



Guia de Estruturação e Administração do Ambiente de Cluster

Versão Beta 0.6

Ministério do Planejamento, Orçamento e Gestão
SLTI – Secretaria de Logística e Tecnologia da Informação
DSI – Departamento de Integração de Sistemas de Informação

Guia de Estruturação e Administração do Ambiente de Cluster

Versão Beta

0.6

Brasília – DF

Presidente da República
Luiz Inácio Lula da Silva

Vice-Presidente da República
José de Alencar Gomes da Silva

Ministro de Estado do Planejamento, Orçamento e Gestão
Paulo Bernardo Silva

Ministro de Estado da Casa Civil
Comitê Executivo de Governo Eletrônico
Dilma Roussef

Secretário de Logística e Tecnologia da Informação
Secretário Executivo de Governo Eletrônico
Rogério Santanna dos Santos

Guia de Estruturação e Administração do Ambiente de Cluster.
Brasília, 2006.
XXX p. : il.

Inclui Bibliografia.

1. Cluster e Grid. 2. Governo Eletrônico. 3. Tecnologias da
Informação e Comunicação. 4. Agregados Computacionais.

“A menos que modifiquemos a nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo”.

Albert Einstein

Coordenação

Secretaria de Logística e Tecnologia da Informação - SLTI
Ministério do Planejamento, Orçamento e Gestão

Colaboração Técnico-Administrativa

Claudete Bezerra da Silva
Diego Sacramento
Fernando Mazoni

Especialistas Convidados

Alice Brito
Adenauer Yamin
Augusto Ovelar
César A. F. De Rose
Daniel Darlen Corrêa Ribeiro
Elizeu Santos-Neto
Fernando Ike
Lucius Trindade Curado e Silva
Marco Sinhareli
Mario Dantas
Philippe O. A. Navaux
Roberto Pires de Carvalho
Reinaldo J. Moreira
Tiarajú Asmuz Diverio
Walfredo Cirne

Consultores Técnicos

Alex Sandro Soares
Elias Otávio de Paula Mussi
Leonardo Rodrigues de Mello

Consultor Responsável

Elias Otávio de Paula Mussi

Coordenação do Projeto de Cluster e Grid

Corinto Meffe
Leonardo Rodrigues de Mello

Coordenação Executiva

Corinto Meffe
José Antônio Borba Soares
Leandro Corte

Coordenação Geral

Rogério Santanna dos Santos

Participação da Sociedade

O Aperfeiçoamento do conteúdo técnico, a inserção de ados e ferramentas ou até a correção de inconsistências técnicas, contou com a participação de várias pessoas.

O intuito de contar com a participação de especialistas desde a primeira versão do Guia surge em função da grande quantidade de tecnologias envolvidas e do grau de complexidade das mesmas.

Não seria possível manter as informações atualizadas e inserir o que há de mais moderno em Cluster e Grid sem a participação da Sociedade.

Contribuições registradas

Adenauer Yamin
Augusto Ovelar
Daniel Darlen Corrêa Ribeiro
Elizeu Santos-Neto
Lucius Trindade Curado e Silva
Marco Sinhoreli
Roberto Pires de Carvalho
Walfredo Cirne

Histórico do Documento

Data	Versão	Autor	Alteração
01/12/05	0.0	Elias Mussi	Estruturação do Sumário
01/05/06	0.1	Trabalhos provindos da equipe interna da SLTI.	Versão inicial de desenvolvimento de conteúdo.
10/02/06	0.1.5	Elias Mussi	Proposta do Sistema de consulta e contribuição on-line para o Guia de Cluster
05/05/06	0.2	Contribuições do Prof. Adenauer Yamin (PPAD) e de Lucius Curado (SSI). Mais trabalhos por parte da equipe da SLTI	Sessões sobre Paralelismo e Sistema de Imagem Única (SSI).
17/06/06	0.3	Elias Mussi	Disponibilização do Sistema de Consulta e Colaboração do Guia de Cluster no endereço http://guialivre.governoeletronico.gov.br/guiaonline/guiacluster/
14/08/06	0.3.5	Equipe SLTI	Expansão do conteúdo do documento, principalmente na parte III.
14/08/06	0.4	Contribuição de Walfredo Cirne e Elizeu Santos-Neto.	Capítulo Grid Computing.
01/09/06	0.4.5	Equipe SLTI	Expansão de conteúdo, principalmente da Parte II.
04/10/06	0.5	Contribuição de Marco Sinhareli, mais trabalhos da Equipe SLTI	Marco, Capítulo sobre Virtualização; SLTI, expansão de conteúdo, principalmente da Parte III
22/10/06	0.6	Contribuição de Roberto Pires de Carvalho, mais trabalhos da Equipe SLTI	Na sessão de Armazenamento Distribuído.

Nota Técnica da Comissão de Redação

Este Guia foi elaborado pela equipe da Gerência de Inovações Tecnológicas (GIT), do Departamento de Integração de Sistemas de Informação (DSI), da Secretaria de Logística e Tecnologia da Informação (SLTI), do Ministério do Planejamento, Orçamento e Gestão (MP).

As diretrizes que compõem este documento têm como base as definições do Governo Eletrônico Brasileiro e respaldo legal no Sistema de Administração dos Recursos de Informação e Informática - SISP instituído através do DECRETO nº1.048, de 21 de janeiro de 1994.

As orientações técnicas são fundamentadas em pesquisadores brasileiros e nas principais tecnologias pertinentes aos ambientes de Cluster e Grid.

A tecnologia de Cluster e Grid, embora recente, possui um amplo acervo de arquiteturas, modelos, ferramentas, middlewares e aplicativos.

Esta Versão Beta (0.6), aberta às contribuições da Comunidade Brasileira, apresenta-se com possíveis inconsistências técnicas, em especial desatualizações ou imprecisões quanto ao estado da arte das soluções apresentadas, que possuem dinâmica acelerada em seu desenvolvimento.

A equipe técnica responsável pela elaboração deste documento conta com a colaboração da Comunidade Acadêmica e de Software Livre para suprir tais lacunas, originadas pela complexidade e pela abrangência do conteúdo do Guia de Cluster.

O lançamento dessa versão 0.6 representa a consolidação dos trabalhos de inserção de conteúdo e a devolução à sociedade do resultado do trabalho até este instante, o qual já conta com importantes colaborações de membros da comunidade

acadêmica brasileira.

Colaborações para este documento podem ser feitas através do sítio <http://guialivre.governoeletronico.gov.br/guiacluster/> e pelo *e-mail*: [<guialivre@planejamento.gov.br>](mailto:guialivre@planejamento.gov.br).

Distribuição

Secretaria de Logística e Tecnologia da Informação. Versão 0.5 e 0.6

Lançamentos Públicos

Versão 0.5	Encontro Mineiro de Software Livre 2006. O Encontro Mineiro de Software Livre 2006, realizado na cidade de Ouro Preto - MG entre os dias 10-12 de outubro de 2006 http://emsl.softwarelivre.org/ .
Versão 0.5	ParGov - SBAC-PAD 2006. "The 18th International Symposium on Computer Architecture and High Performance Computing", realizado na cidade de Ouro Preto - MG entre os dias 17-20 de outubro de 2006 http://www.sbc.org.br/sbac/2006/ .
Versão 0.5	III Fórum Goiano de Software Livre. Realizado na cidade de Goiania - GO entre os dias 27-28 de outubro de 2006.
Versão 0.6	IV CONISLI - Congresso Internacional de Software Livre. IV Congresso Internacional de Software Livre, realizado na cidade de São Paulo - SP entre os dias 03-05 de novembro de 2006 http://www.conisli.org .

Direitos Autorais

Governo Brasileiro – a reprodução em parte ou totalmente é autorizada desde que a fonte seja reconhecida, de acordo com as orientações da CC-GNU GPL¹.



Figura 1: Creative Commons

¹*General Public License* cujo conteúdo está disponibilizado no Apêndice **A**.

Sumário

Sumário	xii
Lista de figuras	xxv
Lista de tabelas	xxix
1 Prefácio	xxxii
1.1 Abreviações e Terminologia	xxxii
1.2 Público	xxxiii
1.3 Autores	xxxiii
1.4 Agradecimentos	xxxiv
I Diretrizes Gerais	1
2 Governo Eletrônico e Novas Concepções Tecnológicas	2
2.1 A Informática Pública Brasileira	2
2.1.1 A Sociedade da Informação e a Inovação Tecnológica	5

2.2	Governo Eletrônico Brasileiro	9
2.2.1	Diretrizes do Governo Eletrônico Brasileiro	10
2.2.2	Padrões de Interoperabilidade de Governo Eletrônico	11
2.2.3	As Diretrizes do Governo Eletrônico e o Software Livre	15
2.2.4	A Arquitetura de Cluster e Grid e as Diretrizes do Governo Eletrônico	18
2.3	As Novas Demandas Computacionais	19
2.3.1	Computação sob Demanda	26
2.3.2	Aproveitamento de Ciclos Ociosos	27
2.4	Dois Paradigmas Computacionais	28
2.4.1	Computação de Grande Porte	29
2.4.2	Computação Distribuída	31
2.4.3	Comparação: Grande Porte e Distribuída	32
II	Aspectos Gerenciais	35
3	Introdução	36
3.1	Vantagens técnicas de utilização de cluster e grid	39
3.2	As Gerações da computação distribuída	40
3.2.1	Tabela Resumo das gerações de computação distribuída	41
3.3	Possibilidades de aplicações práticas de Cluster e Grid	43

3.3.1	Cenários de Aplicação	44
3.4	Arquitetura para sistemas críticos online em N Camadas	54
3.4.1	Definição	54
3.4.2	Camada de Aplicação	55
3.4.3	Camada de Banco de Dados	55
3.4.4	Camada de Armazenamento	55
3.4.5	Diagrama da arquitetura proposta	56
3.4.6	Considerações sobre a arquitetura	57
4	Visão Geral	58
4.1	A sensibilização	58
4.2	Os Recursos Humanos Envolvidos	59
4.2.1	Aperfeiçoamento dos Técnicos	59
4.2.2	Consultoria	59
4.3	O Projeto de Cluster	60
4.3.1	O que deve ser observado	61
5	Processamento Paralelo: Sua Difusão e Uso	71
5.1	Aspectos para a Adoção do Processamento Paralelo	72
5.1.1	Barreiras ao Crescimento da Frequência de Operação dos Processadores	72

5.1.2	Largura de Banda no Acesso à Memória	73
5.1.3	Paralelismo Intrínseco do Mundo Real	74
5.1.4	A Relação Custo-Benefício dos Processadores de Última Geração	74
5.1.5	Aplicações Extremamente Complexas	75
5.1.6	Suporte à Tolerância a Falhas	76
5.1.7	Crescimento Modular	76
5.1.8	Disponibilidade de Software Aplicativo	78
5.1.9	Relação entre a Teoria e a Tecnologia	79
III	Aspectos Técnicos	80
6	Conceitos Básicos	81
6.1	Arquiteturas Computacionais	81
6.1.1	A Classificação de Flynn para Arquiteturas de Computadores	82
6.1.2	Multiprocessadores	88
6.1.3	Multicomputadores	89
6.1.4	Multiprocessadores Simétricos (Symmetric Multiprocessors - SMP)	89
6.1.5	ccNUMA	90
6.1.6	Processadores Massivamente Paralelos (MPP)	90

6.1.7	Sistemas Distribuídos	91
6.1.8	Clusters	92
6.1.9	Grids	92
6.2	Dependabilidade	93
6.2.1	Ameaças	94
6.2.2	Meios	94
6.2.3	Atributos	96
6.3	Escalonamento	97
6.4	Alta Disponibilidade	100
6.5	Balanceamento de Carga	101
6.6	Rede de Comunicações	102
6.6.1	A Importância da Rede de Comunicação	102
6.6.2	Redes de Interconexão Utilizadas em Arquiteturas Paralelas	103
6.6.3	Topologias da Rede de Interconexão	105
6.6.4	Dispositivos de interconexão	109
6.7	Protocolos de Comunicação	113
6.7.1	Frame Relay	113
6.7.2	Asynchronous Transfer Mode	114
6.7.3	FDDI	114

6.7.4	Modelo OSI	115
6.7.5	Protocolo IP	115
6.7.6	Transmission Control Protocol	116
6.7.7	User Datagram Protocol	116
6.7.8	Real-time Transport Protocol	116
6.7.9	Virtual Router Redundancy Protocol - VRRP	117
7	Cluster de Armazenamento	119
7.1	Introdução	119
7.2	Block Devices	120
7.2.1	Arranjo Redundante de Discos (RAID)	121
7.2.2	RAID via Hardware e via Software	122
7.2.3	Distributed Replicated Block Device - DRBD	123
7.2.4	Global Network Block Device - GNBD	127
7.2.5	Internet SCSI (iSCSI)	128
7.3	Sistemas de Arquivos Distribuídos	132
7.3.1	Conceitos Básicos	132
7.3.2	Serviços Oferecidos pelos SADs	135
7.3.3	Algumas Características Desejadas em SADs	139
7.3.4	Network File System - NFS	148

7.3.5	Andrew File System - AFS	152
7.3.6	Constant Data Availability - CODA	156
7.3.7	Lustre	160
7.4	Sistemas de Arquivos Paralelos	161
7.4.1	Parallel Virtual Filesystem Version 1 - PVFS	161
7.4.2	Parallel Virtual Filesystem Version 2 - PVFS2	165
8	Cluster de Aplicação	172
8.1	Linux Virtual Server	173
8.1.1	Nomenclatura e abreviações	174
8.1.2	Tipos de LVS Cluster	175
8.1.3	Algoritmos de escalonamento	179
8.1.4	Casos de uso de LVS	183
8.2	Cluster Tomcat	184
8.2.1	Balanceamento de carga	186
8.2.2	Compartilhamento de sessões	189
8.3	Heartbeat	190
8.4	Zope Cluster	191
9	Cluster de Banco de Dados	195
9.1	Banco de Dados Distribuídos	199

9.2	Replicação de Banco de Dados	200
9.3	PostgreSQL	202
9.3.1	pgpool	202
9.3.2	PGcluster	205
9.3.3	Slony	208
9.4	Mysql	209
9.4.1	Replicação em MySQL	209
9.4.2	MySQL Cluster	213
9.5	Middlewares independentes de Banco de Dados	214
9.5.1	Middleware Sequoia	214
9.5.2	ParGRES	230
10	Alta Capacidade de Processamento (HPC)	232
10.1	Beowulf	232
10.2	Sistema de Imagem Única (SSI)	233
10.2.1	As Principais Características de um Cluster SSI	234
10.2.2	Os principais benefícios de um sistema SSI	236
10.2.3	Memória Distribuída Compartilhada (DSM)	237
10.2.4	OpenMosix	238
10.2.5	Kerrighed	241

11 Ferramentas de Programação Paralela	244
11.1 Troca de Mensagens (Message Passing)	244
11.1.1 PVM	245
11.1.2 Message Passing Interface (MPI)	246
11.2 Relações Entre o Hardware e o Software para Exploração do Paralelismo	248
11.2.1 Relação entre Algoritmos e Arquiteturas Paralelas	249
11.2.2 Propriedades de um Modelo de Programação para o Processamento Paralelo	253
11.3 A Exploração do Paralelismo: Níveis de Abstração e Modelos	258
11.3.1 Modelos nos quais o Paralelismo é Explorado de Forma Totalmente Implícita	258
11.3.2 Modelos com Assinalamento do Paralelismo Explícito	263
11.3.3 Modelos com Assinalamento e Decomposição do Paralelismo Explícitos	266
11.3.4 Modelos com Assinalamento, Decomposição e Mapeamento do Paralelismo Explícitos	268
11.3.5 Modelos com Assinalamento, Decomposição, Mapeamento e Comunicação Explícitos	270
11.3.6 Modelos nos quais o Paralelismo é Explorado de Forma Totalmente Explícita	271
12 Escalonadores de Tarefas	273

12.1 OpenPBS	274
12.2 TORQUE	275
12.3 MAUI	276
13 Grids Computacionais	277
13.1 Introdução	278
13.2 Grids de Serviços	282
13.2.1 Acesso a Serviços	282
13.2.2 Descoberta de Serviços	285
13.2.3 Autenticação e Autorização	288
13.2.4 Privacidade de Dados	289
13.2.5 Composição de Serviço	290
13.2.6 Disponibilização de Serviços	292
13.2.7 Padronização	297
13.3 Grids para Alto Desempenho	302
13.3.1 Plataformas para Processamento Paralelo	302
13.3.2 Execução Remota	308
13.3.3 Escalonamento	308
13.3.4 Imagem do Sistema	321
13.3.5 Segurança	322

13.4 Estudos de Caso	326
13.4.1 Globus	326
13.4.2 MyGrid	335
13.4.3 OurGrid	339
13.4.4 Condor	343
13.5 Tendências em Grids Computacionais	345
14 Virtualização de recursos	347
14.1 Principais tipos de virtualização	347
14.1.1 Virtualização por software	348
14.2 XEN - Xen virtual machine monitor	349
14.2.1 Comparação	350
14.2.2 Sistema Operacional nativo versus virtualização com Xen . .	351
14.2.3 Paravirtualização no Xen	352
14.2.4 Virtualização nativa no Xen	353
IV Apêndices	355
A Licença CC-GNU GPL	356
B Marcas Registradas	366

C	Lista de Abreviaturas	369
D	Tecnologias	371
E	Glossário	374
F	O Ambiente LabCluster	383
F.1	Histórico do LabCluster	384
F.2	Missão do LabCluster	386
F.3	Descrição do Ambiente LabCluster	386
F.4	Infra-estrutura de Hardware	387
F.5	Política de Utilização do Ambiente LabCluster	387
G	Outros Documentos Produzidos	389

Lista de Figuras

1	Creative Commons	xii
2.1	Evolução da carga de processamento e a utilização da computação de grande porte.	30
2.2	Evolução da carga de processamento e a utilização da solução de processamento distribuído.	32
3.1	Evolução da utilização de Arquiteturas de alto desempenho. Fonte Top500.org	37
3.2	Evolução da utilização de S.O. na Top500. Fonte Top500.org	37
3.3	Evolução da utilização por segmentação do mercado. Fonte Top500.org	38
3.4	Esquema do modelo de cluster proposto.	56
5.1	Relação Carga X Custo de investimento, para plataforma Baixa X Alta	77
6.1	Blocos básicos dos computadores sequenciais	81
6.2	Arquitetura genérica de multiprocessador de memória	83

6.3	Arquitetura genérica de multiprocessador de memória compartilhada	85
6.4	Arquitetura genérica síncrona matricial	88
6.5	Alternativas para conectar o processador a rede de interconexão . .	104
6.6	Topologia em barramento	106
6.7	Topologia em malha	107
6.8	Topologia em hipercubo	107
6.9	Topologia em árvore	108
6.10	Esquema de funcionamento de um sistema VRRP	118
7.1	Visão do nível conceitual de funcionamento do DRBD - Trata-se de um <i>driver</i> intermediário entre o <i>block device</i> virtual (<code>/dev/drbd*</code>), o <i>block device</i> real local (<code>/dev/[sh]d*</code>) e os <i>block device's</i> remotos. Todas as transferências são efetuadas pelo protocolo TCP/IP, o que permite sua implementação até mesmo em máquinas geograficamente afastadas.	124
7.2	Fluxo de intercomunicação entre as camadas dos dispositivos Linux - repare que o DRBD não tem como notificar o módulo do sistema de arquivos - mas o oposto ocorre.	126
7.3	Exemplo de cenário GNBD	129
7.4	Exemplo de uma árvore de diretórios	133
7.5	Estrutura de diretórios distribuída	138
7.6	Volumes, VSGs e AVSGs	157
7.7	Visão Geral do PVFS	163

7.8	Clientes acessando o PVFS	164
7.9	Fluxo de dados pelo kernel	164
8.1	Esquema geral de um Linux Virtual Server	174
8.2	LVS-NAT	176
8.3	LVS-DR	178
8.4	LVS-Tun	179
8.5	Visão geral de um cluster Tomcat	185
8.6	Balancamento de carga via DNS Round-Robin	186
8.7	Balancamento de carga via Apache mod_jk	188
8.8	DNS round-robin	193
8.9	ZEO/ZODB + LVS+OCFS2	194
9.1	Sistema de balanceamento de carga	205
9.2	Sistema de balanceamento de carga	206
9.3	Sistema de alta disponibilidade	207
9.4	Princípio do Sequoia	217
9.5	Exemplo de RAIDb-0	219
9.6	Exemplo de RAIDb-1	220
9.7	Exemplo de RAIDb-2	221
9.8	Exemplo de RAIDb-1-0	222

9.9 Exemplo de RAIDb-0-1	222
11.1 Modelo Para Computação Paralela	254
11.2 Números de Fibonacci em Programação Funcional	259
11.3 Fontes de Paralelismo na Programação em Lógica	262
13.1 Acesso transparente a serviços e recursos	278
13.2 Acessando um serviço usando RMI	283
13.3 Acessando um serviço usando CORBA [14]	284
13.4 Interação entre cliente e provedor (Web Services) [343]	285
13.5 Ilustração da arquitetura OurGrid [36]	298
13.6 Relacionamento entre OGSA, OGSi e Globus [343]	300
13.7 Arquitetura multiprocessada	304
13.8 Arquitetura de um MPP	305
13.9 Arquitetura de uma NOW	305
13.10Arquitetura de um Grid Computacional	306
13.11Ilustração de um cenário composto de vários escalonadores	310
13.12Jacobi executando em quatro processadores em um MPP	313
13.13Escalonamento feito pelo Jacobi AppLes	315
13.14Desempenho do WQR	317
13.15Desperdício de ciclos com a replicação	318

13.16	Sumário do desempenho de Storage Affinity comparado com outras heurísticas	319
13.17	Sumário do desperdício de recursos por Storage Affinity comparado com outras heurísticas	320
13.18	Arquitetura do GRAM [133]	329
13.19	Delegação entre escalonadores de aplicação [133]	330
13.20	Exemplo do uso de escalonadores no Globus [133]	331
13.21	Arquitetura do Globus [343]	335
13.22	Arquitetura do MyGrid	337
13.23	Condor protocol [85]	344
A.1	Creative Commons	356

Lista de Tabelas

2.1	Diferenças entre computação de grande porte e distribuída	34
3.1	Tabela de resumo das cinco gerações de computação distribuída . .	42
3.2	Tabela Cenário 1	45
6.1	Formas básicas de tolerância à falhas. Fonte DANTAS [136]	95
6.2	Níveis de Alta Disponibilidade	101
8.1	Exemplos de Sítios que utilizam LVS	184
11.1	Relação entre as características do hardware e do software paralelo	250
13.1	Comparação entre as plataformas de execução para aplicações pa- ralemas	308
13.2	Grid Machine Interface	337
B.1	Tabela de Referência de Marcas Registradas	368
D.1	Tabela de referências de tecnologias	373

F.1 Tabela de Hardware 387

Capítulo 1

Prefácio

1.1 Abreviações e Terminologia

Sempre que possível, na primeira vez em que uma abreviação for usada, será incluída também a versão por extenso. No Apêndice **E** encontra-se um glossário de termos técnicos utilizados.

O Termo *cluster* é utilizado neste documento se referindo as diversas implementações de compartilhamento de recursos computacionais. Tipicamente, um cluster utiliza os recursos de dois ou mais dispositivos de computação¹ em conjunto para um propósito comum. Exemplos de cluster são: Cluster de Processamento de Alto Desempenho ou HPC, Cluster de Balanceamento de Carga e Alta Disponibilidade, Cluster de Banco de Dados e Cluster de Armazenamento. Um outro termo comumente usado é o de *aglomerado de computadores*, utilizado pela comunidade acadêmica brasileira.

Muitas vezes estes ambientes “clusterizados” são construídos a partir de computadores convencionais (desktops), ou seja, vários computadores comuns ligados em rede que se comunicam e trabalham como se fosse uma máquina de grande porte, com capacidade de suportar um ambiente de grande demanda computacional.

¹Estes dispositivos também podem funcionar separadamente

Grid Computacional (The Computational Grid), é uma rede na qual se conecta para obter Serviços Computacionais que agregam recursos sob demanda (ex.: ciclos, armazenamento, software, periféricos, etc).

Os termos *Software* de Fonte Aberta (*Open Source Software*) e *Software* Livre (*Free Software*) tem seus defensores e suas diferenças conceituais e jurídicas. Neste trabalho, usaremos o termo *Software* Livre por se tratar de uma política estratégica do governo e pela intenção de destacar as características que o diferenciam do *Software* de Fonte Aberta, especialmente sua disponibilização na forma da Licença Pública Geral (GPL).

Os termos do Sistema Operacional, como nomes de arquivos, serão apresentados desta forma: Nome de arquivo.

Códigos de programas serão apresentados da forma: Código.

1.2 Público

Este Documento é dirigido aos gerentes e técnicos de Tecnologia da Informação (TI) de todo o Governo Federal Brasileiro, e pode ser utilizado nos outros poderes: Executivo, Legislativo e Judiciário; servindo também como referência para os governos estaduais e municipais que tenham interesse em conhecer e utilizar tecnologias de cluster e grid.

1.3 Autores

Os autores deste documentos são principalmente membros da equipe da Gerência de Inovações Tecnológicas (GIT), do Departamento de Integração de Sistemas (DSI), da Secretária de Logística e Tecnologia da Informação (SLTI) do Ministério do Planejamento, Orçamento e Gestão.

Muitas contribuições de pessoas e instituições externas também foram incluídas neste Guia e estão devidamente registradas na parte inicial deste documento.

1.4 Agradecimentos

Agradecemos a todos as pessoas que participaram da construção deste documento, em especial aquelas que nos enviaram contribuições. A grande maioria dessas pessoas estão citadas na sessão Coordenação e Participação da Sociedade, no início deste documento.

A Coordenação Executiva agradece ao apoio do Secretário de Logística e Tecnologia da Informação, Rogério Santanna dos Santos, pela condição de ser o grande incentivador para a inserção desta tecnologia na Administração Pública Federal (APF); ao Diretor do Departamento de Integração de Sistemas de Informação, Leandro Côrte e ao ex-diretor José Antônio Borba Soares pelo apoio permanente.

Agradecimentos especiais pelos materiais cedidos para o Guia, para os colaboradores: Adenauer Yamin, Daniel Darlen Corrêa Ribeiro, Elizeu Santos-Neto, Lucius Trindade Curado e Silva, Marco Sinhareli, Roberto Pires de Carvalho e Walfredo Cirne.

Parte I

Diretrizes Gerais

Capítulo 2

Governo Eletrônico e Novas Concepções Tecnológicas

2.1 A Informática Pública Brasileira

As primeiras empresas de informática pública surgiram em 1964, inseridas num cenário onde o país ainda buscava desenvolver a economia sustentada no setor agrário. Naquela época, o termo corrente para designar o que hoje conhecemos comumente como informática era “processamento de dados” , termo que não incorporava ainda os recursos de comunicação presentes no cenário da chamada “informática” atual. Ademais, as políticas de informação eram estritamente relacionadas ao tema da segurança do Estado (Torres [134]).

Nos anos seguintes, em especial na década de 70, o Brasil experimentou taxas de crescimento expressivas, apoiadas na forte presença do investimento estatal. Ao final deste período o domínio da tecnologia foi apontado como um fator determinante, dentre outros, para a superação do problema de geração de déficits persistentes, tornando o clima propício para a intensificação dos investimentos públicos em informática, ao lado de uma política protecionista à indústria nacional (Torres [134]).

Um exemplo desta política protecionista foi a Política Nacional de Informática (PNI), Lei 7.232, aprovada em 29 de Outubro de 1984 pelo Congresso Nacional,

com prazo de vigência previamente estabelecido em 8 anos. A Lei tinha como objetivo estimular o desenvolvimento da indústria de informática no Brasil, por meio do estabelecimento de uma reserva de mercado para as empresas de capital nacional.

Apesar da importância neste período do domínio nacional da tecnologia, almejando a utilização de tecnologias consideradas na época “de ponta”, o Estado Brasileiro acabou por se tornar, de certa forma, um ávido consumidor tecnológico. Um grande parque computacional heterogêneo foi estruturado baseado no paradigma da computação de grande porte, num momento em que as tecnologias computacionais eram desenvolvidas por empresas multinacionais e, posteriormente internalizadas no governo, sem um estímulo de pesquisa às universidades brasileiras, bem como ao mercado das empresas nacionais.

Neste paradigma computacional, a grande capacidade de processamento de transações simultâneas e alta disponibilidade estão diretamente relacionadas ao hardware especializado produzido por poucas empresas no mundo. Este modelo amplamente adotado, consolidou a base do processo de automatização e estruturação de sistemas e implementação de serviços, que hoje atinge todos os segmentos do Setor Público.

A falta de padrões abertos transversais e o hardware especializado acabam por tornar o processo de negociação do governo para a aquisição de novos equipamentos e serviços uma atividade limitada e desproporcional. Com poucas empresas capazes de produzir e/ou prestar os serviços para o atendimento das demandas e algumas vezes a ausência de concorrência de empresas na oferta de bens e serviços ao governo, desenvolveram-se diversas relações de dependência tecnológica com os fornecedores. Isto ocorre em função das características deste paradigma computacional, onde as tecnologias de computação de grande porte possuem um elevado custo total de propriedade, sendo utilizadas majoritariamente em grandes projetos e sistemas do governo.

A informática dentro do setor público brasileiro estruturou-se de maneira fragmentada e isolada, tendo criado, diversas ilhas tecnológicas e sistemas sem padrões transversais, o que dificulta e algumas vezes inviabiliza a integração, sendo esta parcialmente realizada muitas vezes através de “pontes”, como por exemplo

SERPRO¹ e DATAPREV², responsáveis pela interligação destas diversas ilhas tecnológicas heterogêneas. Ademais, as iniciativas de governo eletrônico, a pressão e a cobrança da sociedade brasileira pela transparência e otimização do uso de recursos públicos, bem como, o combate à corrupção e à fraude são cada vez maiores, aumentando a necessidade de integração dos sistemas e o poder computacional necessário para realizar análises complexas de imensas bases de dados existentes no governo.

As ações de modernização da máquina pública desde o Plano Nacional de Desburocratização³ até a Reforma Administrativa [175], não foram capazes de atingir os ambientes de tecnologia da informação e comunicação e os sistemas de informação do governo. Isto ocorreu pela dissociação entre a reformulação dos processos administrativos e o modelo de informatização proposto.

Realizar estas mudanças e a necessária otimização da máquina pública, de forma a melhor atender o cidadão, é dificultado ou inviabilizado no paradigma da computação de grande porte, seja por conta dos recursos e investimentos necessários para se estabelecer este processo, seja pela dificuldade para se integrar sistemas, imposto pela falta de padrões. Diante deste cenário se faz necessária a busca por alternativas computacionais inovadoras interoperáveis, plenamente auditáveis, independentes de fornecedor, economicamente sustentáveis para sistemas críticos governamentais e que fomentem o desenvolvimento e pesquisa de novas tecnologias.

Buscando reverter este quadro de dependência tecnológica o governo brasileiro tem investido, através do Ministério da Ciência e Tecnologia e de parcerias entre empresas públicas e universidades, no desenvolvimento de tecnologias de "cluster e grid", baseadas em software livre e voltadas para aplicação de governo eletrônico. Estas tecnologias de "cluster e grid" têm sido largamente utilizadas

¹O Serviço Federal de Processamento de Dados (SERPRO) é a maior empresa pública de prestação de serviços em tecnologia da informação do Brasil, maiores informações em: <http://www.serpro.gov.br/>.

²A Empresa de Processamento de Dados da Previdência Social (Dataprev), ela é a responsável pelo processamento da maior folha de pagamento do país, alcançando mais de 20 milhões de beneficiários/mês. Maiores informações em: <http://www.dataprev.gov.br>.

³O Programa Nacional de Desburocratização da Secretaria de Gestão do Ministério do Planejamento, Orçamento e Gestão, Decreto nº 3335, de 11 de janeiro de 2000, que previa: "Desburocratizar a Administração Pública é fundamental para preparar o país aos novos desafios. É imperativo que o Estado se mostre ágil e competente no atendimento de seus cidadãos, como também é imprescindível que esses não se intimidem ao procurar os serviços públicos e que tenham certeza da boa qualidade e da eficiência do serviço prestado".

em instituições de pesquisa e empresas privadas e estatais, alguns exemplos são: Petrobras, Sistema Nacional de Processamento de Alto Desempenho (SINAPAD), Instituto de Pesquisas Espaciais (INPE), Laboratório Nacional de Computação Científica (LNCC), Google, HP, IBM, Sun, Itautec, Departamento de Defesa Americano (DOD), National Center for Supercomputing Applications (NCSA), entre outros.

É importante salientar que um fator decisivo para a adoção de tecnologias de cluster e grid no governo brasileiro está relacionada à possibilidade de reverter o quadro de consumismo tecnológico desenvolvido ao longo das últimas 2 (duas) décadas e promover o domínio e independência tecnológica do Estado Brasileiro.

Existe uma mudança de paradigma entre as tecnologias de computação distribuída e de computação de grande porte. Na computação distribuída o importante não é a “capacidade de processamento” de um único equipamento, mas sim a “capacidade de processamento coletiva” de um conjunto de equipamentos. Nesta abordagem vários equipamentos com pouca capacidade podem formar um ambiente com grande capacidade de processamento e caso ocorra a falha de um equipamento, o outro assumirá a sua função sem prejuízo para a execução do sistema. Desta forma, elimina-se a necessidade de equipamentos com hardware específico, tolerante a falhas e com redundância.

Com isto, utilizando-se hardware padrão x86 (pc) e a não necessidade de redundâncias e dispositivos especiais no hardware é possível construir sistemas com hardware de baixo custo, compatível com padrões abertos e internacionais, reduzindo a dependência de fornecedores. Com a utilização de soluções baseadas em software livre é possível ainda eliminar a dependência tecnológica e estimular o desenvolvimento de soluções pelos centros de pesquisa, universidades, órgãos de governo e empresas privadas, devido as características de licenciamento do software livre que permitem utilizar o software para qualquer fim, livre distribuição, liberdade de alterar o software e redistribuir a versão alterada.

2.1.1 A Sociedade da Informação e a Inovação Tecnológica

Para a inserção neste novo cenário mundial da economia voltada à informação e tecnologia, cada país desenvolveu estratégias que levou em consideração o seu

grau de desenvolvimento tecnológico conjugado com as suas peculiaridades. No Brasil, o marco inicial desse processo foi a criação do programa “Sociedade da Informação”, por meio do Decreto nº 3.294, de 15 de dezembro de 1999, com o objetivo de “viabilizar a nova geração da Internet e suas aplicações em benefício da Sociedade Brasileira⁴”, estruturado em sete linhas de ação:

- Mercado, trabalho e oportunidades;
- Universalização de serviços para a cidadania;
- Educação na sociedade da informação;
- Conteúdos e identidade cultural;
- Governo ao alcance de todos;
- P&D, tecnologias-chave e aplicações;
- Infra-estrutura avançada e novos serviços.

Esse programa busca contribuir, de forma efetiva, para:

- a construção de uma sociedade mais justa, em que sejam observados princípios e metas relativos à preservação de nossa identidade cultural, fundada na riqueza da diversidade;
- a sustentabilidade de um padrão de desenvolvimento que respeite as diferenças e busque o equilíbrio regional;
- a efetiva participação social, sustentáculo da democracia política.

Com tal esforço, em setembro de 2000, o Governo brasileiro produziu, dentre outros documentos, o chamado “Livro Verde”[135], que identificou o conjunto das ações estabelecidas para impulsionar a Sociedade da Informação no Brasil, contemplando ampliação do acesso à Internet, meios de conectividade, formação

⁴ “O objetivo do Programa Sociedade da Informação é integrar, coordenar e fomentar ações para a utilização de tecnologias de informação e comunicação, de forma a contribuir para que a economia do país tenha condições de competir no mercado global e, ao mesmo tempo, contribuir para a inclusão social de todos os brasileiros na nova sociedade” – disponível em <http://www.socinfo.org.br/sobre/programa.htm>.

de recursos humanos, incentivo à pesquisa e ao crescimento, comércio eletrônico e desenvolvimento de novas aplicações (Guia Livre [151]).

A sociedade da informação não é um modismo. Representa uma profunda mudança na organização da sociedade e da economia, havendo quem a considere um novo paradigma técnico-econômico. É um fenômeno global, com elevado potencial transformador das atividades sociais e econômicas, uma vez que a estrutura e a dinâmica dessas atividades inevitavelmente serão, em alguma medida, afetadas pela infra-estrutura de informações disponível. É também acentuada sua dimensão político-econômica, decorrente da contribuição da infra-estrutura de informações para que as regiões sejam mais ou menos atraentes em relação aos negócios e empreendimentos (Livro Verde [135]).

Na sociedade da informação, o cenário econômico transforma-se de tal modo que a inovação e a conversão de conhecimento em vantagem competitiva passam a constituir importantes diferenciais. Da rapidez na geração e difusão de inovações, decorrem a drástica diminuição da vida útil dos produtos e a necessidade de modernização contínua da produção e da comercialização de bens e serviços. Da conversão do conhecimento surgem as possibilidades de se incorporar os benefícios da tecnologia com maior agilidade.

Para a nova economia, não basta dispor de uma infra-estrutura moderna de comunicação; é preciso competência para transformar informação em conhecimento. A educação é um elemento-chave para a construção de uma sociedade da informação e condição essencial para que pessoas e organizações estejam aptas a lidar com o novo, a criar e, assim, a garantir seu espaço de liberdade e autonomia. O desafio, portanto, é duplo: superar antigas deficiências e criar as competências requeridas pela nova economia.

O governo, nos níveis federal, estadual e municipal, tem o papel de assegurar o acesso universal às tecnologias da informação e comunicação e a seus benefícios, independentemente da localização geográfica e da situação social do cidadão, garantindo níveis básicos de serviços, estimulando a interoperabilidade de tecnologias e de redes. A sociedade civil deve zelar para que o interesse público seja resguardado, buscando organizar-se para monitorar e influenciar, sistematicamente, os poderes públicos e as organizações privadas (Livro Verde [135]).

Papel importante para o êxito do Programa caberá às universidades e demais entidades educacionais, pelo seu envolvimento na formação de recursos humanos e na construção da indispensável base científico-tecnológica. Em particular, nesse contexto, é estratégico deter conhecimento avançado sobre as tecnologias de informação e comunicação que hoje ocupam o centro da dinâmica de inovações e são fatores primordiais de competitividade econômica.

Assim desafios da sociedade da informação demandam cada vez mais uma grande quantidade de recursos computacionais, devido a ampla difusão de serviços e aplicações ao público geral, em especial aos cidadãos. Neste contexto, o "Livro Verde" aponta uma série de tecnologias consideradas chave para o desenvolvimento deste processo, dentre estas tecnologias encontra-se o Processamento de Alto Desempenho, abordado no capítulo 8, que ilustra os seguintes tipos de aplicações: Genoma humano, Dispersão de poluição, Biologia estrutural, Previsão meteorológica, Modelagens de Informação.

Outras aplicações, são possíveis para a nova arquitetura, dentre elas: a prestação de informações ligadas aos serviços públicos, o acompanhamento das ações de governo e condução dos negócios públicos (por ex. compras governamentais), o acesso aos governantes e representantes eleitos são exemplos das possibilidades do uso das tecnologias de informação e comunicação pela máquina administrativa pública. A tecnologia pode ainda ser largamente aplicada para aperfeiçoar a própria gestão do governo - coordenação, planejamento, execução e controle de ações, contabilidade pública etc. - e suas transações comerciais com o setor privado. Conjuntamente essas demandas e as Diretrizes de Governo Eletrônico, de utilização da WEB para prestação da maior parte destes serviços, serviços estes que tem uma grande demanda computacional, com grande quantidade de acesso, usuários simultâneos e alta demanda de processamento, que acabam trazendo a tona as arquiteturas de cluster e grid computacional.

"O setor governamental é o principal indutor de ações estratégicas rumo à sociedade da informação. Primeiramente, porque cabe ao governo definir o quadro regulatório dentro do qual projetos e iniciativas concretas poderão ser formuladas. Segundo, porque como regra o governo é o maior comprador/contratador de bens e serviços em tecnologias de informação e comunicação em um país. Assim, uma decisão do governo em apoio a uma tecnologia ou serviço pode abrir algumas avenidas de atividades ao setor privado, bem como conduzir outras a

becos sem saída. Terceiro, porque o governo, com o uso exemplar de tecnologias de informação e comunicação em suas atividades, pode acelerar grandemente o uso dessas tecnologias em toda a economia, em função da maior eficiência e transparência de suas próprias ações"(Livro Verde [135]).

2.2 Governo Eletrônico Brasileiro

O Governo Eletrônico foi concebido como instrumento de transformação da sociedade brasileira, estabelecendo diretrizes e parâmetros para a criação de uma sociedade digital.

Com o passar do tempo, a chamada "Sociedade da Informação" apresentou novos paradigmas que mereciam igualmente a atenção do Governo Eletrônico.

Assim, em suas diretrizes, foram explicitados:

"[...] O papel do Estado neste mundo em transformação continua fundamental como agente estratégico para o atendimento da demanda de maior participação direta dos cidadãos e, ao mesmo tempo, a tomada de decisões centrais estratégicas e rápidas.

O crescimento das informações em rede, o aumento da transparência, e a conseqüente diminuição da burocracia estatal, aumentarão o controle social sobre o Estado, o que contribuirá para a democratização do processo decisório e para uma maior efetividade da ação governamental.

Neste ambiente de transformações, o Governo Eletrônico pretende ser um agente democrático, estratégico, socialmente justo e ao mesmo tempo eficiente na prestação de serviços aos seus cidadãos.(Vide sítio do Governo Eletrônico [6]) "

Com a preocupação de melhor adequar o País a esse cenário, foram criados, por meio de decreto de 29 de outubro de 2003, comitês técnicos específicos no âmbito do Comitê Executivo do Governo Eletrônico: Implementação do *Software* Livre, Inclusão Digital, Integração de Sistemas, Sistemas Legados e Licenças de *Software*, Gestão de Sítios e Serviços *On-Line*, Infra-Estrutura de Rede, Governo para Governo (G2G), Gestão de Conhecimento e Informação Estratégica.

Segundo o sítio do Governo Eletrônico[6], as principais linhas de ação do Poder Executivo Federal em tecnologia da informação e comunicação estão estruturadas caminhando em direção a um governo eletrônico que procura promover: a universalização do acesso aos serviços, a transparência das suas ações, a integração de redes e o alto desempenho dos seus sistemas. Neste sentido o governo vem atuando em três frentes fundamentais: a interação com o cidadão, a melhoria da sua própria gestão interna, e a integração com parceiros e fornecedores. Neste processo é importante o compartilhamento de recursos do governo, a unicidade e troca de informações entre aplicações, e a responsabilização e credenciamento de gestores da informação, que permita uma integração das redes de governo, com independência, respeitando as peculiaridades setoriais dos órgãos.

2.2.1 Diretrizes do Governo Eletrônico Brasileiro

Em decorrência do Decreto de 29 de outubro de 2003, a implementação do Governo Eletrônico passou a ser realizada segundo sete princípios, que foram assim concebidos⁵:

[...] como referência geral para estruturar as estratégias de intervenção, adotadas como orientações para todas as ações de Governo Eletrônico, gestão do conhecimento e gestão da TI no governo federal[6]:

1. A prioridade do Governo Eletrônico é a promoção da cidadania.
2. A Inclusão Digital é indissociável do Governo Eletrônico.
3. O Software Livre é um recurso estratégico para a implementação do Governo Eletrônico.
4. A gestão do conhecimento é um instrumento estratégico de articulação e gestão das políticas públicas do Governo Eletrônico.
5. O Governo Eletrônico deve racionalizar o uso de recursos.
6. O Governo Eletrônico deve contar com um arcabouço integrado de políticas, sistemas, padrões e normas.
7. Integração das ações de Governo Eletrônico com outros níveis de governo e outros poderes.

⁵ *Oficinas de Planejamento Estratégico. RELATÓRIO CONSOLIDADO. Comitê Executivo do Governo Eletrônico. Maio de 2004. pág 8.*

Nesse novo contexto, a atuação do Governo Eletrônico pretende melhorar a prestação de serviços aos cidadãos, com aumento da transparência e diminuição da burocracia, contribuindo para a democratização do processo decisório, a maior efetividade das ações governamentais e a promoção da inclusão digital.

Para dar suporte a toda demanda computacional que é criada por esses princípios, é que se propõe a utilização de Cluster e Grids no governo, como forma de criar um ambiente computacional robusto, de alto grau de confiança e de baixo custo.

2.2.2 Padrões de Interoperabilidade de Governo Eletrônico

Com a intenção de estruturar mecanismos capazes de promover a eficiência da Administração Pública no contexto da “Sociedade da Informação”, articulada às ações estabelecidas para implantação do Governo Eletrônico, o Governo brasileiro elaborou um conjunto de premissas, políticas e especificações técnicas regulamentadoras para utilização da Tecnologia da Informação e da Comunicação, denominada “Arquitetura e-PING – Padrões de Interoperabilidade⁶ de Governo Eletrônico”.

A “Arquitetura e-PING” define um conjunto mínimo de premissas, políticas e especificações técnicas, que regulamentam a utilização da Tecnologia de Informação e Comunicação (TIC) no Governo Federal, estabelecendo as condições de interação com os demais poderes e esferas de governo e com a sociedade em geral. As áreas cobertas pela e-PING, estão segmentadas em: Interconexão; Segurança; Meios de Acesso; Organização e Intercâmbio de Informações e Áreas de Integração para Governo Eletrônico.

Assim, pela e-PING,

“A existência de uma infra-estrutura de Tecnologia da Informação e Comunicação (TIC) que se preste como o alicerce para a criação dos serviços de governo eletrônico é o pré-requisito para o fornecimento de melhores serviços à sociedade, a custos mais baixos. Um governo moderno e integrado exige

⁶ Os conceitos de interoperabilidade adotados nesta arquitetura estão evidenciados no Documento de Referência, disponível em <http://www.eping.e.gov.br>.

sistemas igualmente modernos e integrados, interoperáveis, trabalhando de forma íntegra, segura e coerente em todo o setor público.

Políticas e especificações claramente definidas para interoperabilidade e gerenciamento de informações são fundamentais para propiciar a conexão do governo, tanto no âmbito interno como no contato com a sociedade e, em maior nível de abrangência, com o resto do mundo. A e-PING é concebida como uma estrutura básica para a estratégia de governo eletrônico, e permite racionalizar investimentos em TIC, por meio do compartilhamento, reutilização e intercâmbio de recursos tecnológicos."

A e-PING apresenta, em cada um dos seus segmentos, políticas técnicas norteadoras para estabelecimento das especificações de seus componentes, que são fundamentadas em algumas políticas gerais. Para este trabalho, as principais políticas gerais levantadas pela e-Ping, que atingem e/ou norteiam o desenvolvimento de sistemas de Cluster e Grid são (e-PING versão 1.9 [2] - pág: 9) :

- **Alinhamento com a INTERNET:** todos os sistemas de informação da administração pública deverão estar alinhados com as principais especificações usadas na Internet e com a *World Wide Web*;
- **Adoção do XML** como padrão primário de intercâmbio de dados;
- **Desenvolvimento e adoção de um Padrão de Metadados do Governo Eletrônico** - e-PMG, baseado em padrões internacionalmente aceitos;
- **Escalabilidade:** as especificações selecionadas deverão ter a capacidade de atender alterações de demanda no sistema, tais como, mudanças em volumes de dados, quantidade de transações ou quantidade de usuários. Os padrões estabelecidos não poderão ser fator restritivo, devendo ser capazes de fundamentar o desenvolvimento de serviços que atendam desde necessidades mais localizadas, envolvendo pequenos volumes de transações e de usuários, até demandas de abrangência nacional, com tratamento de grande quantidade de informações e envolvimento de um elevado contingente de usuários;
- **Adoção Preferencial de Padrões Abertos:** a e-PING define que, sempre que possível, serão adotados padrões abertos nas especificações técnicas. Padrões proprietários são aceitos, de forma transitória, mantendo-se as pers-

pectivas de substituição assim que houver condições de migração. Sem prejuízo dessas metas, serão respeitadas as situações em que haja necessidade de consideração de requisitos de segurança e integridade de informações. Quando disponíveis, soluções em Software Livre são consideradas preferenciais.

Em sua segunda parte, "Especificação Técnica dos Componentes da e-PING", vários pontos são levantados de interesse para novos projetos de sistemas de informática e informação. Principalmente no que se pode caracterizar como computação distribuída, com a utilização de Web Services e de Arquitetura Orientada a Serviços (SOA).

Com a utilização de Web Services para a interligação, integração e interoperabilidade de sistemas. Da sessão "6.1. Interconexão: Políticas Técnicas":

- " 6.1.7. Sempre que possível, deve ser utilizada tecnologia baseada na web em aplicações que utilizaram Emulação de Terminal anteriormente."
- " 6.1.8. A tecnologia de Web Services é recomendada como padrão de interoperabilidade da e- PING."
- " 6.1.9. Os Web Services deverão ser registrados e estar localizados em estruturas de diretório compatíveis com o padrão UDDI. O protocolo de acesso a essa estrutura deverá ser o HTTP."
- " 6.1.10. O protocolo SOAP é recomendado para comunicação entre os clientes e os Web Services e a especificação do serviço deverá utilizar a linguagem WSDL."

Na e-PING, "Web Service" está definido como:

"Os Web Services são aplicações de software, identificadas por uma URI (Uniform Resource Identifier), cujas interfaces e ligações são capazes de serem definidas, descritas e descobertas por artefatos baseados em XML. Além disso, possuem suporte para integração direta com outras aplicações de software, utilizando, como padrão de interoperabilidade, mensagens escritas em XML e encapsuladas em protocolos de aplicação padrão da Internet.

A necessidade de integração entre os diversos sistemas de informação de governo, implementados em diferentes tecnologias, às vezes de forma simultânea e em tempo real, implica na adoção de um padrão de interoperabilidade que garanta escalabilidade e facilidade de uso.

A tecnologia de Web Services é adequada para atender tais necessidades, além de ser independente em relação aos Sistemas Operacionais e às Linguagens de Programação.

O uso de Web Services contempla tanto transferências de documentos entre Instituições, quanto solicitações para execução de serviços remotos."

E em conjunto são recomendados as seguintes especificações:

- Protocolo de troca de informações: SOAP v1.2, como definido pela W3C;
- Infra-estrutura de registro: Especificação UDDI v3.0.2 (Universal Description, Discovery and Integration) definida pela OASIS;
- Linguagem de definição do serviço: WSDL 1.1 (Web Service Description Language) como definido pelo W3C.

Um outro fator importante está levantado na sessão de Integração para governo eletrônico, onde se define as diretrizes técnicas para o segmento, dela (a sessão "10.1 Áreas de Integração para Governo Eletrônico: Políticas Técnicas") se tem:.

"A partir do entendimento de que a materialização do uso de XML Schemas se dá através de serviços interoperáveis:

- Recomenda-se que a Arquitetura Orientada a Serviços - SOA - e as políticas técnicas relacionadas ao Segmento Interconexão sejam observadas no projeto e implementação de aplicações baseadas nos XML Schemas referidos;
- O segmento passa a referenciar a iniciativa "Arquitetura Referencial de Interoperação dos Sistemas Informatizados de Governo - AR", que é um modelo de Arquitetura Orientada a Serviços, adaptado à realidade dos Sistemas Informatizados de Governo e que, oportunamente poderá ser acessada em <http://guialivre.governoeletronico.gov.br/ar/>

Assim com essas políticas de padronização, o governo cria mecanismos para que os projetos em computação distribuída entre os Órgãos do Governo possa ser construído e se obtenham maiores vantagens das arquiteturas de Cluster e Grid. Essas padronizações já são as bases para várias tecnologias já existentes na área, que hoje são maduras e utilizadas pela indústria.

2.2.3 As Diretrizes do Governo Eletrônico e o Software Livre

As diretrizes do Programa Brasileiro de Governo Eletrônico demonstram que a Gestão do Conhecimento e o uso de Padrões Abertos e Software Livre são instrumentos estratégicos de articulação e gestão de políticas públicas porque possibilitam a produção compartilhada e colaborativa de conhecimento, assegurando assim, a habilidade de criar, organizar e compartilhar soluções e conhecimentos estratégicos para o Estado Brasileiro.

O “Guia Livre - Referência de Migração para Software Livre do governo federal”, documento norteador para a migração e utilização de Software Livre na APF, explicita os benefícios obtidos pelo Estado ao se optar por este tipo de tecnologia. Como por exemplo:

“ Nesse cenário, a filosofia do *Software* Livre surge como oportunidade para disseminação do conhecimento e nova modalidade de desenvolvimento tecnológico, em função do novo paradigma que se estabelece na relação de quem produz o *software* (sejam empresas, sejam programadores autônomos) com a tecnologia propriamente dita.”

e

“ Assim, a adoção do *Software* Livre por parte do Estado é amparada principalmente pelos princípios de Impessoalidade, Eficiência e Razoabilidade⁷, visando à melhoria na qualidade dos serviços prestados e à promoção dos desenvolvimentos tecnológico e social.

⁷O artigo 37 da Constituição da República apresenta os Princípios Basílicos da Administração Pública: legalidade, impessoalidade, moralidade, publicidade e eficiência. O princípio da razoabilidade possui fundamentação implícita, sendo evidenciado em algumas Constituições Estaduais.

Portanto, o Estado se beneficia diretamente com a adoção do *Software* Livre, tanto no aspecto de sua estruturação para atendimento às demandas sociais, como no seu papel de promover desenvolvimento. Desse modo, possibilitamos a integração das políticas de modernização administrativa, inclusão social baseadas na Tecnologia da Informação e no desenvolvimento industrial.

A questão do *Software* Livre está contextualizada em amplo cenário integrado, composto por ações de desenvolvimento tecnológico, inserção adequada do País na chamada "Sociedade da Informação", promoção da cidadania, inclusão digital e racionalização de recursos. "

O "Guia Livre" define como principais razões para o uso de software Livre:

- Necessidade de adoção de padrões abertos para o Governo Eletrônico (e-Gov);
- Nível de segurança proporcionado pelo *Software* Livre;
- Eliminação de mudanças compulsórias que os modelos proprietários impõem periodicamente a seus usuários, em face da descontinuidade de suporte a versões ou soluções;
- Independência tecnológica;
- Desenvolvimento de conhecimento local;
- Possibilidade de auditabilidade dos sistemas;
- Independência de fornecedor único.

São apresentados os motivos pelos quais não basta ter acesso ao código aberto, mas é preciso desenvolver comunidades capazes de contribuir para a evolução dos códigos e algoritmos disponibilizados, criando inovações, gerando melhorias e aperfeiçoando os mesmos. As motivações não podem ser apenas econômicas, mas também devem ser orientadas pelas possibilidades de criação e de avanços nas áreas de produção, do conhecimento e de novas tecnologias, assim estimulando o desenvolvimento de todo um conjunto de áreas relacionadas ao *software*, ao conhecimento e à gestão do Estado Brasileiro.

O software livre, por princípio, depende do emprego de padrões abertos. Tal uso vem a facilitar também ações relacionadas com integração de sistemas, otimização de processos e adoção de arquiteturas computacionais abertas.

O compartilhamento da informação e a adoção desse *software* por muitos órgãos públicos e privados contribui para que o produto se mantenha atualizado e ganhe um corpo muito superior ao que cada instituição isoladamente poderia fazer e, sobretudo, se sustenta não apenas por ser uma licença de *software* livre adequada, mas também pela criação de uma comunidade que zela para o seu desenvolvimento, compartilhando saberes e soluções. A comunidade contribui para mantê-lo vivo, corrigindo seus defeitos, aperfeiçoando seu funcionamento, introduzindo inovações e fazendo com que o produto se consolide mais robusto e a cada dia se torne mais conhecido por um conjunto maior de funcionários públicos e privados e por diferentes segmentos da sociedade.

A razão pela escolha preferencial do *software* livre no governo federal é motivado pelos resultados obtidos com o seu compartilhamento junto à sociedade. O *software* livre também possibilita ao cidadão o direito de acesso aos serviços públicos e ao conhecimento sem obrigá-lo a usar uma plataforma específica.

A utilização de *software* livre em soluções de “Cluster e Grid” é uma tendência clara que vem se estabelecendo nos últimos anos:

De acordo com o top500.org⁸, 72% dos computadores mais rápidos do mundo são clusters, e o Linux já está presente em 73% destes.

Os principais desafios de utilização de *software* livre no desenvolvimento de soluções em “Cluster e Grid” para a construção de sistemas críticos governamentais consistem na possibilidade de se aproveitar a grande quantidade de soluções e *softwares* de “Cluster e Grid” disponíveis, bem como na perspectiva de compartilhamento dos sistemas desenvolvidos com outros órgãos e instituições públicas, dentro da perspectiva conceitual do *software* público (vide [270]).

⁸<http://www.top500.org/stats>

2.2.4 A Arquitetura de Cluster e Grid e as Diretrizes do Governo Eletrônico

As principais razões pela escolha preferencial por arquiteturas de cluster e grid no governo federal estão embasadas nas diretrizes de governo eletrônico de utilização de software livre e racionalização de recursos e encontram-se descritas abaixo:

- independência tecnológica;
- independência de fornecedor;
- integração de processos de inovação tecnológica nas estruturas de informática pública, como instrumento de melhoria da qualidade de serviços, competitividade e eficiência;
- estímulo o desenvolvimento de tecnologias nacionais e a política nacional de informática;
- adoção de padrões abertos de hardware⁹ e software;
- interoperabilidade como um fator preponderante no desenvolvimento de sistemas e arquiteturas computacionais no governo;
- aproveitamento dos potenciais disponibilizados pela ampla estrutura de redes computacionais do governo federal.

O presente documento apresenta as possibilidades, tecnologias e cenários de utilização de cluster e grid no governo federal, tendo como objetivo ampliar seu uso interno no governo de maneira a melhor atender as novas demandas computacionais da sociedade da informação que, segundo a diretriz de modernização da máquina pública, encontram-se cada vez mais internalizadas no governo brasileiro.

⁹Também conhecido como hardware commodity, hardware padrão de mercado fornecido por diversas empresas que concorrem entre si para oferecer as melhores condições de suporte, qualidade e preço para o governo

2.3 As Novas Demandas Computacionais

As atividades econômicas que utilizam redes eletrônicas como plataforma tecnológica têm sido denominadas negócios eletrônicos (*e-business*). Essa expressão engloba os diversos tipos de transações comerciais, administrativas e contábeis, que envolvem governo, empresas e consumidores. O comércio eletrônico (*e-commerce*) é a principal atividade dessa nova categoria de negócios.

Os atores institucionais envolvidos nos serviços governamentais são o próprio Governo (*G*), Instituições Externas (*B*, de *business*), e o Cidadão (*C*), que interagem entre si de várias maneiras. Há cinco tipos de relações entre esses atores em aplicações governamentais:

- *B2B (business-to-business)*:
transações entre empresas, exemplos: EDI, portais verticais de negócios;
- *B2C / C2B (business-to-consumer / consumer-to-business)*:
transações entre empresas e consumidores, exemplos: lojas e shoppings virtuais;
- *B2G / G2B (business-to-government / government-to-business)*:
transações envolvendo empresas e governo, exemplos: EDI, portais, compras. Corresponde a ações do Governo que envolvem interação com entidades externas. O exemplo mais concreto deste tipo é a condução de compras, contratações, licitações, etc, via meios eletrônicos;
- *C2C (consumer-to-consumer)*:
transações entre consumidores finais (exemplos: sites de leilões, classificados on-line);
- *G2C / C2G (government-to-consumer / consumer-to-government)*:
transações envolvendo governo e o cidadão (consumidores finais dos serviços do Governo), exemplos: pagamento de impostos, serviços de comunicação). Corresponde a ações do Governo de prestação (ou recebimento) de informações e serviços ao cidadão via meios eletrônicos. O exemplo mais comum deste tipo é a veiculação de informações em um *website* de um órgão do governo, aberto a todos os interessados;

- *G2G (government-to-government)*: transações entre instituições do govern em qualquer nível ou esfera do Poder. Corresponde a funções que integram ações do Governo horizontalmente, exemplo: no nível Federal, ou dentro do Executivo; ou verticalmente, exemplo: entre o Governo Federal e um Governo Estadual.

A informatização de operações internas e de serviços prestados pelo Governo remete à necessidade de se planejar, implementar e operar grandes aplicações de tecnologias de informação e comunicação, envolvendo o desenvolvimento de pacotes de software de grande complexidade, para execução em plataformas usualmente bastante heterogêneas de computadores e redes.

Tais aplicações, especialmente as de escala nacional, que costumam tratar de imensas quantidades de dados, que perpassarão inúmeras gerações tecnológicas, são tão carregadas de variáveis e condicionantes que, tipicamente, são descritos e caracterizados como “sistemas complexos”:

- têm dimensões gigantescas, tais como milhões de usuários, centenas de funções etc.;
- têm especificação dinâmica, isto é, se modifica ao longo do tempo, para acomodar novas necessidades, revisão de prioridades, etc.;
- nunca terminam de ser implementados, como consequência natural das duas características anteriores.

Assim os softwares desenvolvidos pela administração pública têm de optar pelas tecnologias adequadas e compatíveis, seguindo as diretrizes de Governo Eletrônico e os padrões de interoperabilidade já definidos pela e-PING. A adoção de padrões técnicos e sua institucionalização são críticas para assegurar que aplicações governamentais, mesmo resultando de uma miríade de iniciativas descentralizadas e descoordenadas de desenvolvimento, possam interoperar e se integrarem.

A idéia de desenvolvimento em espiral de sistemas é bastante antiga e está na base da idéia de se ter uma seqüência de versões para um serviço. Muitas vezes, as versões são impostas pela evolução tecnológica. Mas especialmente no caso de software, o desenvolvimento em espiral é utilizado como estratégia defensiva para o projeto de sistemas complexos.

Aplicações governamentais, mais do que quaisquer outras, demandam uma abordagem em espiral. Contudo, com demasiada frequência, elas são concebidas na forma de processos lineares com visão demasiadamente simplista, com cronogramas irrealistas e sem um plano de evolução de longo prazo.

Os desafios da “Sociedade da Informação” apresentados no Livro Verde, a inserção do Brasil neste processo Global, a disseminação do acesso a Internet e as pesquisas do meio acadêmico, convergiram em novas possibilidades de aplicação da tecnologia da informação na disponibilização de serviços e informações aos cidadãos.

Existe uma percepção no governo e uma pressão da sociedade em torno da melhoria da qualidade dos serviços prestados aos cidadãos, bem como para o aumento substancial da transparência dos processos governamentais e o combate à fraude e à corrupção. Para atender estas demandas se faz necessário atingir um nível maior de integração entre os sistemas governamentais e criar novos sistemas de inteligência capazes de transformar o imenso volume de dados atual em informação útil e agregada, através da utilização de sistemas de *Business Intelligence* (BI) e de *Enterprise Resource Planning* (ERP) [272].

Além da iminente necessidade de integração e atribuição de valor agregado aos dados transformando-os em informação, é importante salientar que a quantidade de serviços e portais agregadores destas informações e de interação com os cidadãos também deverá crescer conjuntamente com a democratização do acesso à Internet, e sua conseqüente utilização como meio de comunicação entre governo e cidadãos no contexto de desenvolvimento de governo eletrônico.

A ampliação e a melhoria da qualidade dos processos internos e dos serviços prestados pelo governo refletem-se na necessidade de aumento da capacidade computacional do setor público e, por se tratarem de serviços críticos, possuem como características principais de demandas computacionais:

- alta disponibilidade;
- suporte de milhares a milhões de usuários simultâneos;
- alta capacidade de processamento;
- capacidade de trabalhar com bancos de dados da ordem de milhões de re-

gistros;

- tolerância a falhas de *hardware* e *software*;
- facilidade de integração e interoperabilidade;
- adoção de padrões abertos de *hardware* e *software*;
- armazenamento massivo da ordem de TeraBytes de dados;

A necessidade de ampliação da malha computacional, atendendo as características expostas acima, deve superar um conjunto de restrições ou problemas que estão relacionados com a utilização de computação de grande porte para o efetivo atendimento das novas demandas, sendo eles:

- Limitação financeira dos investimentos públicos e a crescente necessidade de racionalização do uso de recursos públicos em TI, que muitas vezes impossibilitam o desenvolvimento ou implementação de um novo sistema diante do custo total de propriedade envolvido na aquisição de *hardware* e *software* para computação de grande porte.
- Dificuldade de aumentar ou diminuir a capacidade computacional de acordo com a demanda atual de cada instituição. Normalmente, servidores de computação de grande porte possuem uma capacidade máxima de expansão limitada por série ou modelo do equipamento. Quando uma instituição atinge a capacidade máxima do modelo que é utilizado, torna-se necessário realizar a troca do equipamento em operação por um modelo de capacidade maior. Geralmente, os custos para a troca de modelo de equipamento por um de maior capacidade são elevados.
- Dependência tecnológica e de fornecedor devido à utilização de *hardware* altamente especializado no modelo da “computação de grande porte”, os quais somente a empresa que comercializa o equipamento ou o *software* é capaz de prestar suporte, realizar manutenção ou a venda de novos componentes. Isso leva ao controle do preço do mercado e enfraquece as relações de negociação entre governo e empresas para a obtenção da melhor solução pelo menor custo possível, influenciada pela menor competição entre empresas no mercado.

- Sistemas e *hardwares* heterogêneos: existe no governo um parque computacional extremamente heterogêneo. Encontram-se em uso *hardwares* e *softwares* dos mais variados fornecedores, muitas vezes incompatíveis entre si, devido ao emprego de padrões fechados ou arquiteturas proprietárias.
- Dificuldade de integração e interoperabilidade entre os sistemas devido à abundância de padrões proprietários, segmentados e divergentes, conjugados com a falta de padrões abertos transversais. A Administração Pública é obrigada a construir ou adquirir *brokers*, que funcionam como pontes entre tecnologias incompatíveis, envolvendo muitas vezes o pagamento de licenças para desenvolvimento das arquiteturas proprietárias utilizadas. Estes fatores dificultam e muitas vezes retardam o processo de integração nas arquiteturas computacionais baseadas em *mainframe* e computadores de grande porte.

Neste cenário o Governo Brasileiro tem desenvolvido diversos projetos objetivando maior transparência nas ações governamentais, otimização do uso de recursos públicos, construção de processos democráticos entre governo, empresas e cidadãos. Alguns exemplos são:

- Sistema eleitoral com votação e apuração através da utilização de urnas eletrônicas;
- Portal de Compras Governamentais¹⁰ e o Sistema Integrado de Administração de Serviços Gerais - SIASG, que são um conjunto de ferramentas para operacionalizar internamente o funcionamento sistêmico das atividades inerentes ao Sistema de Serviços Gerais¹¹, responsáveis pelas compras governamentais¹².

¹⁰<http://www.comprasnet.gov.br>

¹¹O Sistema Integrado de Administração de Serviços Gerais - SIASG, é um conjunto informatizado de ferramentas para operacionalizar internamente o funcionamento sistêmico das atividades inerentes ao Sistema de Serviços Gerais - SISG, quais sejam: gestão de materiais, edificações públicas, veículos oficiais, comunicações administrativas, licitações e contratos, do qual o Ministério do Planejamento, Orçamento e Gestão - MP é o órgão central normativo.

¹²O Governo Federal economizou R\$ 637,8 milhões com a utilização do pregão eletrônico de janeiro a julho de 2006. Esta modalidade foi responsável por 47,3% do total de bens e serviços adquiridos pelo governo durante este período. Um estudo recente realizado pelo Banco Mundial na área de compras públicas eletrônicas demonstra a eficiência do sistema de aquisições eletrônicas do Governo Federal Brasileiro. Segundo a avaliação do Banco Mundial, o Comprasnet obteve pontuação máxima nos indicadores que avaliaram a transparência na divulgação das licitações e de seus respectivos resultados e na utilização de métodos competitivos conforme recomendado pela lei.

- Arrecadação Fazendária: esta é uma das áreas mais avançadas em serviços de governo eletrônico no Brasil. A maioria dos serviços e sistemas de arrecadação fazendária estão disponíveis na Internet. A declaração de imposto de renda de pessoa física disponibilizada por meio eletrônico através da Internet e por disquete foi responsável por 98,2% [256] do total de declarações no ano de 2005.

- Projeto $I^3 - Gov$ ¹³:

O Projeto I^3 -Gov é uma implementação da arquitetura referencial de interoperação de sistemas - e-PING, através dele é possível acessar os Sistemas Estruturadores de Governo, obtendo informações de forma automática e interoperável. São disponibilizadas as seguintes informações e serviços: i) Em Informação, é possível ver, por exemplo, o resultado dos gastos públicos com Saúde, Educação e outros, sob a ótica dos Programas e Ações de Governo¹⁴, ii) Em Inteligência, estão registrados, de maneira padronizada, os macroprocessos de gestão administrativa de Governo. Um exemplo é o fluxo de aquisição de bens e de serviços, iii) Em Integração, ao implementar a Arquitetura Referencial de Interoperação, são organizados os serviços dos Sistemas Estruturadores de Governo. O acesso às informações utiliza metodologia e arquitetura padronizadas que garantem a interoperação transparente e automática¹⁵.

Neste projeto estão envolvidas tecnologias de webservices, datawarehouse, OLAP, ETL e integração de dados/sistemas.

- Projeto de Qualidade de Informações Sociais: O software de gestão de qualidade de dados permite limpar, padronizar e cruzar os cadastros que utilizam dados como nome, data de nascimento, nome de pai e mãe e número de documento de identificação. Também possibilita identificar erros de digitação e fazer comparações por similaridade. Reconhece, por exemplo, a existência de dois cadastros para uma única pessoa que possui um registro com o nome completo e outro com apenas o último sobrenome. Ou no

¹³Integração e Inteligência em Informações de Governo

¹⁴O PPA, plano Plurianual estabelece, de forma regionalizada, as diretrizes, os objetivos e as metas da Administração Pública Federal, e é o principal instrumento de planejamento, por conseguinte, de mudança econômica e social com vistas ao desenvolvimento do País. O PPA organiza a atuação governamental em programas e ações, inserindo na administração pública a orientação do gasto público para resultados na sociedade.

¹⁵Ligação máquina a máquina sem interferência de um operador (pessoa), com periodicidades programadas e, se for o caso, com latência zero. Rotinas administrativas de cadastro e habilitação em serviços disponibilizados máquina a máquina sem interferência de um operador (pessoa), com periodicidades programadas e, se for o caso, com latência zero.

caso de uma mulher que se registrou no sistema com o nome de solteira e, depois, com o nome de casada. As rotinas atuais não consideram essas diferenças e permitem que uma pessoa receba duas vezes o mesmo benefício.

- **Projeto Interlegis:** O Interlegis é um programa desenvolvido pelo Senado Federal, em parceria com o Banco Interamericano de Desenvolvimento (BID), de modernização e integração do Poder Legislativo nos seus níveis federal, estadual e municipal e de promoção da maior transparência e interação desse Poder com a sociedade. Os meios utilizados são as novas tecnologias de informação (Internet, videoconferência e transmissão de dados) que permitem a comunicação e a troca de experiências entre as Casas Legislativas e os legisladores e entre o Poder Legislativo e o público, visando aumentar a participação da população. Em função das finalidades do Programa o cadastro no Portal Interlegis é aberto a qualquer pessoa, dando a oportunidade a essas pessoas adicionarem conteúdo ao site (páginas, imagens, links, notícias, ...), que serão avaliados pelos administradores de conteúdo do Portal, para depois serem divulgados. O link "Ajuda do Portal" foi feito para facilitar a compreensão de como uma pessoa pode se cadastrar, acessar e manusear os conteúdos que o Portal disponibiliza para ela.
- **o Sistema Integrado de Administração de Recursos Humanos (SIAPE¹⁶):** Sistema responsável pela geração e processamento da folha de pagamentos dos funcionários da administração pública federal. Atualmente, este sistema funciona em um computador de grande porte que centraliza o processamento de todas as folhas de pagamento da administração pública federal. Teoricamente, o processamento de uma folha de pagamento de dois funcionários distintos não possui interdependência entre si e poderia ser realizado de forma distribuída utilizando-se tecnologias de processamento paralelo ou "Grid Computing" do tipo *bag-of-tasks*. Esta abordagem poderia utilizar equipamentos dedicados distribuídos ou até mesmo aproveitar os recursos ociosos das estações de trabalho do governo federal, eliminando a necessidade e os custos envolvidos na aquisição e manutenção do *main-frame* utilizado atualmente por este sistema.
- **Sistema Integrado de Administração Financeira do Governo Federal - SIAFI¹⁷:** O SIAFI é o principal instrumento utilizado para registro, acompa-

¹⁶<http://www.siapenet.gov.br>

¹⁷<http://www.tesouro.fazenda.gov.br/SIAFI/>

nhamento e controle da execução orçamentária, financeira e patrimonial do Governo Federal. Atualmente, este sistema é executado em *mainframe* e encontra-se em andamento no Serviço Federal de Processamento de Dados (Serpro) um projeto[201] de migração para baixa plataforma, adoção de *software* livre e tecnologia de Cluster de balanceamento de carga.

Estes projetos desenvolvidos pelo governo possuem um ou mais das seguintes características:

- Necessidade de Alta Capacidade de Processamento;
- Suporte a Milhares de Usuários Simultâneos;
- Alta Capacidade de Armazenamento;
- Alta Disponibilidade;
- Segurança;
- Utilização de padrões abertos;
- Necessidade de integração de sistemas e bases de dados.

Os “grandes” sistemas utilizados pelo governo em sua grande maioria, como alguns dos descritos anteriormente, são desenvolvidos para a utilização em sistemas baseados em “Computação de Grande Porte”. A seção 2.4 apresenta uma descrição sobre o paradigma computacional atualmente utilizado e defesa de utilização do paradigma computacional de cluster e grid para atingir os mesmos resultados com maiores vantagens para a administração pública.

2.3.1 Computação sob Demanda

Vários sistemas de governo possuem demandas flutuantes, ou seja, durante um momento convivem com pouca ou nenhuma carga de processamento e em outro momento específico possuem uma elevada carga de processamento.

Um exemplo deste perfil é o sistema de declaração de imposto de renda. Durante um período do ano é ativada a entrada de dados no sistema para a realização de

declarações de imposto de renda. Quanto mais se aproxima o final deste período, maior a quantidade de declarações e, conseqüentemente, a capacidade computacional necessária para o funcionamento do sistema. O mesmo ocorre com o SIAPE, só que neste caso a utilização para processamento e entrada de dados no sistema ocorre com maior concentração durante alguns dias do mês.

A arquitetura da computação de grande porte não possui capacidade para que facilmente se aumente ou diminua seu poder computacional, sem esbarrar nas barreiras impostas por seu *hardware* especializado e proprietário, por conta de seu alto custo total de propriedade e a dificuldade de aquisição destes equipamentos. Em função desta relação de dependência, a administração pública é obrigada a adquirir um *hardware* com maior capacidade de processamento para atender a demanda de pico de processamento do sistema. Durante quase toda a sua vida útil, o equipamento adquirido possuirá capacidade de processamento ociosa, devido às características de demandas flutuantes de processamento desses sistemas governamentais.

Em resumo, a administração alocará na maior parte do tempo recursos financeiros em um equipamento subutilizado, quando este recurso poderia ser utilizado em áreas com demandas mais urgentes.

Para equacionar questões como essas, a administração pública está em busca de alternativas computacionais baseadas em "Cluster e Grid" que auxiliem na resolução desse desafio de computação sob demanda, minimizando a capacidade de processamento ociosa existente dentro da administração pública, bem como a alocação de recursos financeiros desnecessários.

2.3.2 Aproveitamento de Ciclos Ociosos

Estima-se que a administração pública direta possua um parque computacional em torno de 300 mil estações de trabalho, que adotam um padrão de uso comum. Este padrão consiste numa maior utilização destes equipamentos durante os horários de expediente comercial de trabalho. Entretanto, até mesmo durante os horários de maior utilização destes equipamentos existem grandes períodos de ociosidade do uso de processamento dos mesmos. Este imenso recurso computacional ocioso poderia ser amplamente aproveitado através do emprego de

paradigmas de computação probabilística e Grid Computing. Alguns possíveis usuários desta capacidade de processamento seriam: o governo através do processamento e execução de sistemas internos, e as universidades e centros de pesquisa através de projetos de pesquisa científica [272].

Existem diversos projetos mundiais de aproveitamento de recursos ociosos de processamento que demonstram o poder da computação probabilística. Alguns exemplos são: SETI@home, que posteriormente deu origem ao BOINC¹⁸, uma infra-estrutura aberta de computação em rede; e o distributted.net¹⁹, um projeto de computação distribuída para finalidades genéricas criado em 1997. Nestes projetos são utilizados os ciclos ociosos de processamento de máquinas interligadas na internet, não dedicadas exclusivamente à rede de processamento e dispersas mundialmente.

Uma lista extensa porém incompleta de projetos de aproveitamento de ciclos de processamento ociosos pode ser encontrada na página:

"http://en.wikipedia.org/wiki/List_of_distributed_computing_projects"

Existem no Brasil diversas atividades de pesquisa em andamento e soluções desenvolvidas em universidades brasileiras que poderiam ser utilizadas para aproveitar a capacidade de processamento ocioso das milhares de estações de trabalho do governo brasileiro. Algumas dessas soluções são: o vCluster[147] e o Ourgrid[62], desenvolvidos respectivamente pela Pontifícia Universidade Católica do Rio Grande do Sul e pela Universidade Federal de Campina Grande.

2.4 Dois Paradigmas Computacionais

O modelo atualmente adotado pelas empresas de informática governamentais e por muitas grandes empresas, para resolver o paradigma exposto na sessão anterior, é caracterizado principalmente pelos sistemas heterogêneos de grande porte, com a utilização de mainframes e supercomputadores.

A evolução das soluções de Cluster e Grid vem criando uma nova alternativa

¹⁸Berkeley Open Infrastructure for Network Computing
<http://boinc.berkeley.edu/>

¹⁹<http://distributted.net>

para os ambientes computacionais de grande porte. A utilização destes ambientes para computação de alta performance vem crescendo rapidamente e já é quase predominante para as grandes máquinas utilizadas nos dias de hoje. Segundo a lista Top500²⁰, em sua publicação de número 27 (06/2006), sistemas de Cluster já são responsáveis por 72,80% dos sistemas integrantes da lista, com 364 sistemas.

2.4.1 Computação de Grande Porte

A computação de grande porte é definida como sistema de alta capacidade de computação, também conhecida como “Alta plataforma”, esta é caracterizada pela utilização de Mainframes e supercomputadores.

Mainframes são sistemas de computação, dedicados normalmente ao processamento de um volume grande de informações e transações. Os mainframes são capazes de oferecer serviços de processamento a milhares de usuários através de milhares de terminais conectados diretamente ou através de uma rede. São computadores que geralmente ocupam um grande espaço físico e necessitam de ambiente especial para seu funcionamento. Os mainframes são capazes de realizar operações em grande velocidade e sobre um volume muito grande de dados. Os mainframes nasceram em 1946 e foram constantemente aperfeiçoados. Em 7 de abril de 1964, a IBM apresentou o “System/360”, mainframe que, na época, foi o maior projeto de uma empresa. Desde então, outras empresas, como a HP e a Burroughs (atual Unisys) lançaram seus modelos de mainframe.

Supercomputador é um computador com altíssima velocidade de processamento e grande capacidade de memória, empregado em pesquisas científicas e militares. Este termo é geralmente confundido com cluster, um tipo de supercomputador criado a partir da cooperação de vários computadores convencionais. Os primeiros supercomputadores foram criados na década de 1960 por Seymour Cray. Seymour Cray fundou sua própria empresa, a Cray Research, em 1970 e dominou o mercado da supercomputação durante 25 anos (1965-1990).

²⁰ Top500[362] é uma lista dos 500 maiores sistemas computacionais do mundo, que é gerada a cada 6 (seis) meses, essa lista é mantida por causa dos interesses, tanto de pesquisadores, como de vendedores de sistemas, mas principalmente para os usuários e futuros usuários de sistemas de alta performance. A lista é classificada pela performance computacional do sistema realizada pelo benchmark LINPACK.

A distinção entre supercomputadores e mainframes não é clara e direta, mas genericamente são diferenciados pelas tarefas submetidas, os supercomputadores são utilizados na solução de problemas em que o tempo de cálculo é um limite (processamento), enquanto os mainframes são utilizados em tarefas que exigem alta disponibilidade e envolvem alta taxa de transferência de dados (internos ou externos ao sistema)(Wikipedia[386]).

A Figura 2.1, representa o problema de escalabilidade e dimensionamento decorrente da utilização da computação de grande porte. A área mais escura(em azul) reflete uma possível evolução da carga²¹ de processamento em um período de tempo.

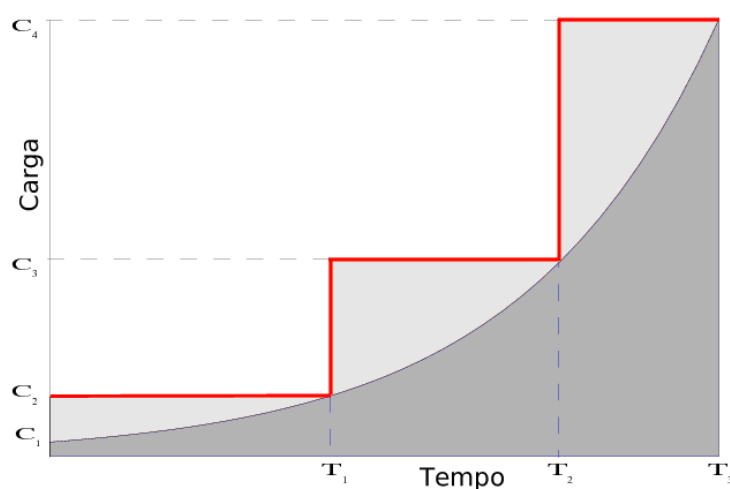


Figura 2.1: Evolução da carga de processamento e a utilização da computação de grande porte.

²¹A palavra carga é utilizada nesta seção para representar o uso de recurso computacional, seja de processamento, rede e armazenamento.

Inicialmente, para responder à uma demanda de carga de processamento C_1 é necessário que a Administração adquira uma solução com capacidade de processamento superior à exigência inicial. Essa medida justifica-se pelo já citado alto custo do equipamento envolvido: uma vez que recursos financeiros serão empregados na utilização de máquinas multiprocessadas, é interessante que essas máquinas operem no maior tempo possível. Dessa forma, para satisfazer a demanda de processamento destacada em C_1 , adquire-se uma solução com capacidade C_2 , superior, que irá garantir o funcionamento do ambiente até que a exigência de processamento atinja seu limite máximo. Na figura 2.1, a área mais clara(em laranja) representa a capacidade ociosa de processamento do equipamento utilizado em função do tempo.

Quando o limite de processamento do equipamento for alcançado, torna-se necessário a realização de um *upgrade*, que geralmente caracteriza-se pela substituição do equipamento original por um novo, ou pela incorporação de novo *hardware* ao equipamento original. Qualquer uma das alternativas exigirá elevado custo financeiro. Assim, passa-se à utilização de um equipamento com capacidade de processamento C_3 , para novamente garantir a operacionalização do ambiente por mais um período de tempo (T_1 até T_2), inaugurando um ciclo constante de atualizações determinado pela carga de processamento a ser suportada.

No caso da utilização da computação de grande porte, percebe-se que as soluções adquiridas operam a maior parte do tempo com carga de processamento inferior à sua capacidade, devido ao alto custo de *hardware* envolvido associado à dificuldade de dimensionamento do ambiente, conforme representado pela área em mais clara na Figura 2.1 e normalmente quando atingem a carga máxima, sofrem dificuldades de expansão do hardware(capacidade de processamento).

Portanto, em última instância, a Administração aloca recursos financeiros em uma solução cuja capacidade de processamento não será plenamente exigida na fase inicial de alocação de recursos computacionais.

2.4.2 Computação Distribuída

Por utilizar componentes físicos comuns em sua arquitetura, um ambiente *cluster* apresenta facilidade de dimensionamento da capacidade de processamento. O

ambiente pode ser concebido de acordo com a exigência inicial da carga de processamento do ambiente. À medida que a carga aumenta, novos componentes físicos podem ser facilmente alocados no ambiente para suprir a necessidade de processamento. Como nestes ambientes podem ser empregados *hardwares* mais abertos, ou utilizados pelo mercado, consegue-se realizar um rápido dimensionamento com custo reduzido, maximizando a capacidade de processamento do ambiente.

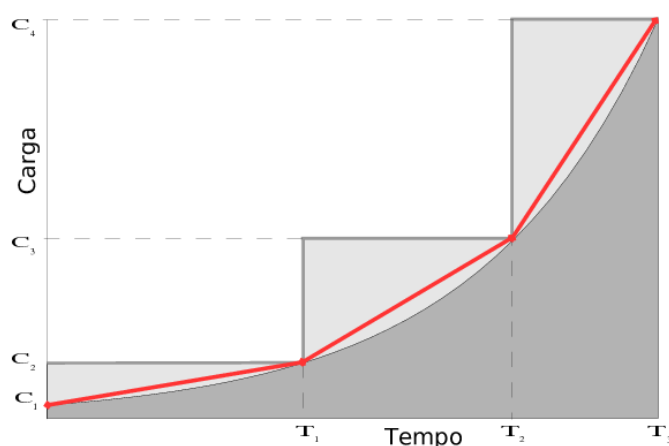


Figura 2.2: Evolução da carga de processamento e a utilização da solução de processamento distribuído.

A Figura 2.2 apresenta o resultado da utilização do ambiente *cluster*. Para efeito de ilustração foram utilizados os mesmos parâmetros da Figura 2.1 (relativa à adoção da computação de grande porte). As linhas em vermelho representam a evolução do dimensionamento da carga de processamento do ambiente *cluster*.

2.4.3 Comparação: Grande Porte e Distribuída

Existem grandes similaridades entre as arquiteturas de computação de grande porte e a computação distribuída.

Algumas dessas similaridades são:

- Alto Poder de Processamento;
- Alta Disponibilidade;
- Suporte a Milhares de Transações e Usuários simultâneos;
- Contingenciamento de recursos;
- Grande Capacidade de Armazenamento.

Informações detalhadas sobre como implementar estas funcionalidades em arquiteturas distribuídas podem ser encontradas nos capítulos: Cluster de processamento (capítulo 10), Cluster de Aplicação (capítulo 8), Cluster de Banco de Dados (capítulo 9), Cluster de Armazenamento (capítulo 7), Grid computing (capítulo 13) e Virtualização de Recursos (capítulo 14) neste documento.

A tabela 2.1 apresenta as principais diferenças entre as duas abordagens tecnológicas tratadas na seção 2.4.

Grande Porte	<i>Cluster e Grid</i>
- Alto custo de implantação; - Dependência de fornecedor único; - Utilização de <i>hardware</i> específico; - Alto custo de manutenção; - Dificuldade de redimensionamento do ambiente; - Utilização parcial da capacidade de processamento; - Grande custo total de propriedade; - Tecnologia estabelecida no mercado.	- Baixo custo de implantação; - Independência de fornecedores – facilidade de negociação; - Utilização de <i>hardware</i> comum – padrão PC; - Baixo custo de manutenção; - Facilidade de redimensionamento do ambiente; - Maximização da capacidade de processamento; - Baixo custo total de propriedade; - Tecnologia inovadora.

Tabela 2.1: Diferenças entre computação de grande porte e distribuída

Parte II

Aspectos Gerenciais

Capítulo 3

Introdução

As tecnologias de Cluster e Grid tem sido amplamente utilizadas nos últimos 20 anos, principalmente em aplicações de pesquisa, algumas das áreas de maior utilização destas tecnologias são: pesquisa genética, bioinformática, física, química, engenharia, climatologia, petroquímica, pesquisa espacial e resolução de equações e métodos matemáticos.

Apesar das tecnologias de clusters serem recentes, sua utilização é crescente e já domina a lista das máquinas mais rápidas do mundo. A figura 3.1 mostra a evolução percentual das arquiteturas de sistemas de alto desempenho no mundo, os dados são da organização "top500.org" [362].

Assim como as arquiteturas de cluster vem crescendo, a utilização de software livre (GNU/Linux) também vem crescendo de forma acentuada. A figura 3.2 mostra a evolução de sua utilização nos últimos anos.

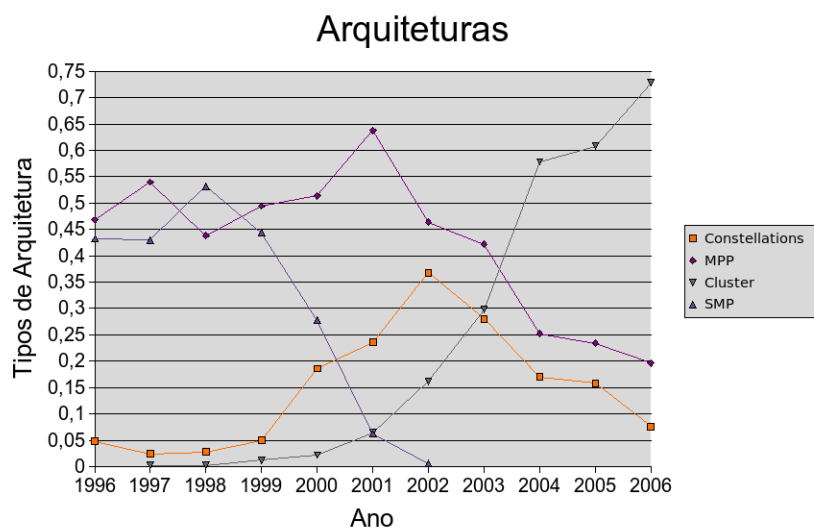


Figura 3.1: Evolução da utilização de Arquiteturas de alto desempenho. Fonte Top500.org

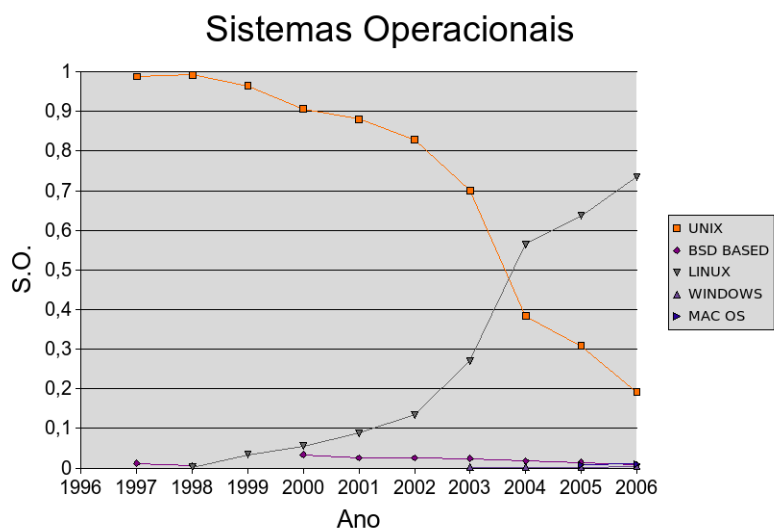


Figura 3.2: Evolução da utilização de S.O. na Top500. Fonte Top500.org

O mercado corporativo tem percebido as vantagens e possibilidades de negócios relacionadas a utilização de tecnologias baseadas em Cluster e Grid, sendo observado um crescimento da adoção destas tecnologias no mercado corporativo. Este fato pode ser analisado pelo investimento para desenvolvimento destas tecnologias realizado pelas maiores empresas de tecnologia do mundo, como Oracle, Google, IBM, Intel, AMD, Sun, entre outras. Além disso, uma pesquisa realizada

no ano de 2004 pelo instituto Forrester Research¹ constatou que 37% das grandes empresas do mercado corporativo estão em alguma fase de adoção/desenvolvimento de projetos baseados em tecnologias de Grid Computing.

A organização Top500 também mantém os dados sobre os segmentos corporativos que utilizam as máquinas de maior capacidade computacional do mundo, a figura 3 mostra a evolução no tempo desses segmentos de utilização.

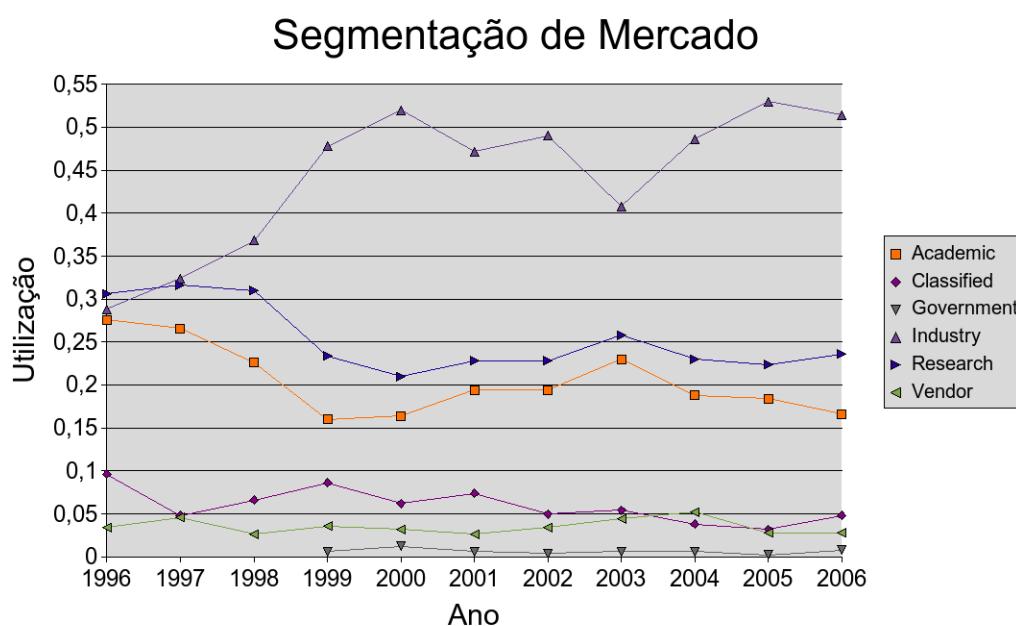


Figura 3.3: Evolução da utilização por segmentação do mercado. Fonte Top500.org

A utilização de computação distribuída no governo federal tem sido pequena, devido principalmente a dificuldade de encontrar documentação em português sobre estas tecnologias, com isso, esta parte do documento irá abordar as principais tecnologias de Cluster e Grid e suas possibilidades de utilização no ambiente do Governo Federal, com o objetivo de estimular a utilização destas tecnologias no governo.

¹Forrester, 2004. <http://www.forrester.com/go?docid=34449>

3.1 Vantagens técnicas de utilização de cluster e grid

A sessão 2.3 apresenta algumas demandas e desafios computacionais do Governo Brasileiro e a possibilidade de utilização de tecnologias baseadas em cluster e grid para auxiliar no atendimento destas demandas. De forma que utilização de cluster e grid nos ambientes do governo possui as seguintes vantagens técnicas:

- Utilização de hardware padrão de mercado em sistemas críticos através da transferência do gerenciamento das funções de alta disponibilidade, tolerância à falhas e balanceamento de carga do hardware para o software. Diminuindo a necessidade de hardware especializado, aumentando a concorrência entre as empresas fornecedoras e propiciando ao governo a independência tecnológica de fornecedores de hardware.
- Em geral, as tecnologias de Cluster e Grid possuem como base padrões abertos e interoperabilidade. Facilitando a integração de sistemas baseados nestas tecnologias, em oposição a sistemas em computação de grande porte que utilizam em sua grande parte tecnologias proprietárias e padrões fechados.
- Disponibilidade de soluções baseadas em software livre que permitem a implementação de sistemas de cluster e grid sem a necessidade de ônus de licenças de software, além de permitir a melhoria, alteração, distribuição e compartilhamento de soluções, segurança, transparência e possibilidade de auditoria plena do sistema. Somente as categorias de Clustering² e Computação Distribuída³ do site de projetos Sourceforge.net possuem juntas um total de 1363⁴ projetos de tecnologias de cluster, grid e computação distribuída.
- Maior Facilidade de aumentar ou diminuir a capacidade computacional de acordo com a demanda existente. Grids e clusters computacionais. Esta questão foi abordada no relatório de avaliação de ferramenta de mineração, que encontra-se disponível para download no endereço: <http://guialivre.governoeletronico.gov.br/labcluster/tamandua.pdf>

²http://sourceforge.net/softwaremap/trove_list.php?form_cat=141

³http://sourceforge.net/softwaremap/trove_list.php?form_cat=308

⁴Último acesso: 28/09/2006

- Possibilidade do desenvolvimento de sistemas e serviços que utilizem os conceitos de computação sob demanda, com o objetivo de aproveitar da melhor maneira possível os sistemas e recursos computacionais existentes no governo.
- Possibilidade de realizar o aproveitamento de ciclos ociosos de computadores já existentes na infra-estrutura de TI atual. Estima-se que o governo federal possua atualmente um total de aproximadamente 300 mil computadores. A maior parte destes equipamentos possui um padrão de utilização semelhante, onde são grandes os períodos de tempo de inatividade ou ociosidade destes equipamentos. Esta imensa capacidade computacional poderia ser utilizada para a execução de sistemas de governo ou para o processamento de projetos de pesquisa técnica-científica desenvolvidos nas universidades brasileiras e empresas públicas brasileiras.

3.2 As Gerações da computação distribuída

Durante os últimos 20 anos, a computação distribuída passou por um processo intenso de mudanças e revoluções. Este processo foi marcado por 5 gerações computacionais descritas a seguir:

- **Primeira Geração de Computação distribuída:**
A primeira geração, também conhecida como host-based computing, é baseada na utilização de terminais burros que servem apenas como meio de visualização de aplicações, softwares e dados que encontram-se no computador central. Os recursos computacionais de processamento e armazenamento utilizados nesta geração são exclusivamente do computador que hospeda as aplicações.
- **Segunda Geração de Computação distribuída:**
Na segunda geração, passam a ser utilizados computadores clientes com uma capacidade um pouco maior, capazes de suportar a emulação de terminal, entretanto as aplicações continuam sendo armazenadas e executadas em um servidor remoto.
- **Terceira Geração de Computação distribuída:**
A terceira geração é caracterizada pela utilização do paradigma de cliente

e servidor, as aplicações são desenvolvidas para serem executadas em um computador cliente, terem uma interface com o usuário e interagirem com servidores de aplicação.

- Quarta Geração de computação distribuída:

A quarta geração é caracterizada pela utilização de aplicações multi-camadas com regras de negócio, interface de usuários e dados separadas entre ambiente de interação do usuário e várias camadas de servidores.

- Quinta geração de computação distribuída:

A quinta geração, também conhecida como grid computing, é caracterizada pela existência pela utilização por parte do cliente de recursos computacionais alocados em um pool virtual, de forma que o cliente utiliza capacidade computacional de acordo com a sua necessidade sem precisar ter maiores detalhes ou controle de onde são os recursos utilizados.

3.2.1 Tabela Resumo das gerações de computação distribuída

A tabela 3.1 apresenta um resumo das cinco gerações da computação distribuída

Da mesma forma que em cada uma destas revoluções aconteceu mudanças estruturais nas relações entre as pessoas (usuárias ou desenvolvedores de tecnologias) e os sistemas computacionais, o mesmo irá ocorrer na adoção de tecnologias em Cluster e Grid. Não existe tecnologia pior ou melhor do ponto de vista global, cada tecnologia possui seu nicho de utilização e aplicação, cabe ao gestor do sistema ou aplicação realizar a devida análise e verificar quais procedimentos devem ser realizados. Ao menos que a mudança tecnológica seja transparente para o usuário do sistema, como em todas as mudanças poderá vir a ser necessário realizar alterações nos sistemas, treinamento e capacitação dos usuários e desenvolvedores, desenvolvimento de novas aplicações baseadas no novo paradigma.

Cinco Gerações de Computação Distribuída	
Geração	Características
Primeira (Computação baseada em Host)	• Terminal Burro • Um servidor • Aplicações Monolíticas
Segunda (Acesso Remoto)	• Um cliente suportando somente funções de emulação de terminal • Um Servidor
Terceira (Cliente/Servidor)	• Um cliente suportando regras de processamento, bem como interfaces de usuário • Até dois servidores
Quarta (Multi-camadas)	• Um cliente suportando regras, bem como interfaces de usuário • Mais de duas camadas de servidores
Quinta (Grid)	• Ambiente virtual onde todos os sistemas são considerados um pool de recursos • N-camadas • Arquiteturas orientadas a serviço

Tabela 3.1: Tabela de resumo das cinco gerações de computação distribuída

3.3 Possibilidades de aplicações práticas de Cluster e Grid

Adoção de tecnologias em Cluster e Grid muitas vezes poderá impactar nas relações entre as pessoas (usuárias ou desenvolvedores de tecnologias) e os sistemas computacionais, da mesma forma que qualquer outra mudança tecnológica. Os gestores, juntamente com os técnicos, deverão definir qual tecnologia será adotada, quando e sobre qual metodologia/procedimento através de uma análise prévia dos possíveis benefícios obtidos com a mudança tecnológica e os riscos/impactos envolvidos.

O Governo Brasileiro possui várias aplicações e demandas computacionais distintas. Entretanto, existem necessidades ou características que são comuns a todas as aplicações:

- **Alta Disponibilidade:** a indisponibilidade de alguma aplicação de governo acarretará desde uma interrupção na execução das atividades internas, até mesmo, uma impossibilidade de serviços prestados aos cidadãos. Por este motivo, uma demanda transversal e comum dos sistemas e aplicações de governo é que eles possuam algum tipo de sistema de alta disponibilidade e tolerância à falhas. Na computação de grande porte tais sistemas são implementados em hardware altamente especializado.
- **Segurança:** a demanda de segurança deve ser entendida como:
 - **Confidencialidade:** o acesso a informação deverá ser realizado tão somente às entidades legítimas, ou seja, àquelas autorizadas pelo proprietário da informação.
 - **Integridade:** a informação manipulada deve manter todas as características originais estabelecidas pelo proprietário da informação, incluindo controle de mudanças e garantia do seu ciclo de vida (nascimento, manutenção e destruição).
 - **Disponibilidade:** a informação deve estar sempre disponível para o uso legítimo, ou seja, por aqueles usuários autorizados pelo proprietário da informação.

- **Utilização de Padrões Abertos:** crescente necessidade de utilização de padrões abertos e comuns entre as diferentes arquiteturas de aplicações com o objetivo de facilitar a integração de sistemas, bases de dados e diminuir a dependência de tecnologias proprietárias. A e-ping⁵ define os padrões adotados, recomendados e em transição que deverão ser utilizados pelo governo brasileiro.
- **Aderência à Legislação:** sistemas e aplicações de governo necessariamente devem estar de acordo com a legislação vigente no país.

A definição de soluções tecnológicas, ou até mesmo a realização de uma análise do ambiente computacional do governo, é complicada devido esta heterogeneidade e diversidade tecnológica existente. Desta maneira, é preciso realizar uma análise de cada cenário e aplicação, para poder definir qual tecnologia será capaz de atender as demandas da instituição com o melhor custo-benefício.

A adoção de tecnologias de Cluster e Grid para aplicações críticas no governo, pode representar uma redução nos custos, viabilização da implementação de novos sistemas e ampliação da capacidade computacional dos sistemas existentes, devido principalmente a utilização de hardware commodity, de software livre e a questão estratégica da independência tecnológica e de fornecedor.

Existem tecnologias de Cluster e Grid para as mais variadas finalidades, sendo necessário realizar uma análise, e conseqüente verificação de quais tecnologias são capazes de atender as demandas computacionais existentes na instituição, com o melhor custo-benefício e o menor risco possível à continuidade do negócio.

3.3.1 Cenários de Aplicação

Para efeitos didáticos e de esclarecimento das possibilidades de utilização de tecnologias de Cluster e Grid no governo federal, serão definidas 3 cenários diferentes onde serão apresentadas características das aplicações, demandas computacionais e uma pequena análise sobre quais tecnologias poderão ser utilizadas. Para informações técnicas sobre as tecnologias de Cluster e Grid abordadas neste documento, leia a parte **III**.

⁵www.eping.e.gov.br

Cenário 1

Neste cenário, a instituição da administração pública possui um portal web de serviços voltados aos cidadãos, sendo utilizado basicamente para realizar consultas e obter informações, possuindo conteúdo estático (armazenado localmente em sistema de arquivos) e conteúdo dinâmico (armazenado remotamente através da utilização de um servidor de banco de dados).

O portal precisa estar disponível para acesso e consulta dos usuários na maior parte do tempo possível, questões como integridade, confidencialidade e autenticidade são essenciais, pois as informações contidas no portal não devem ser alteradas e disponíveis para pessoas que não possuam a devida autorização.

A tabela 3.2 apresenta o mês, a quantidade de acessos simultâneos ao portal e a carga de processamento utilizada no computador que hospeda a aplicação.

Mês	Acessos Simul-tâneos	Carga Utili-zada
01	250	50%
02	350	70%
03	450	90%
04	500	100%

Tabela 3.2: Tabela Cenário 1

Neste exemplo, após o servidor atingir sua capacidade de processamento máxima em 4 meses, caso seja possível, será necessário realizar otimizações na aplicação, para continuar utilizando o mesmo servidor por mais tempo ou realizar um upgrade na capacidade computacional do servidor. Após atingir o limite técnico de expansão do servidor deverá ser adquirida uma máquina com maior capacidade de processamento e expansão.

Normalmente, quanto maior a capacidade de processamento de um único servidor maior será o preço, sendo que o custo envolvido na aquisição não cresce de

maneira linear e o preço de uma máquina com 4 processadores é maior que preço de duas máquinas de 2 processadores cada. Apesar disso, uma máquina de 8 processadores terá um desempenho menor que duas máquinas de 4 processadores devido as características e limitações físicas da arquitetura x86_32 e x86_64. Sabe-se que nestas arquiteturas computacionais a melhor relação custo/performance são obtidas em máquinas de 4 processadores.

Caso a demanda continue crescendo, em pouco tempo, uma única máquina na arquitetura x86_32 e x86_64 não será capaz de atender as necessidades computacionais, neste caso existirão duas possibilidades: Trocar a arquitetura da máquina utilizada, ou distribuir o atendimento da demanda em mais de um servidor.

A abordagem tradicionalmente utilizada seria a primeira e tem apresentado alguns problemas tais como: alto custo, dependência tecnológica e de fornecedor e conseqüente dificuldade de administrar o ambiente de TI.

Para se realizar a ampliação da capacidade computacional de acordo com a segunda possibilidade, distribuindo o atendimento da demanda em mais de um servidor, deverão ser utilizadas tecnologias e soluções para a implementação de Cluster ou Grid.

Neste cenário de portal ou aplicação web, poderão ser utilizadas tecnologias de alta disponibilidade(para garantir que o nível de serviço exigido) e Balanceamento de Carga (para distribuir as requisições dos usuários entre os servidores). Estas tecnologias podem ser utilizadas em conjunto ou separadamente de acordo com a necessidade da instituição. Exemplos de possibilidade são:

- Implementar somente um sistema de alta disponibilidade com duas máquinas:

A capacidade de processamento deverá ser suportada trivialmente por apenas uma máquina. Uma das tecnologias mais utilizadas nesta possibilidade é o HeartBeat⁶

- Implementar somente um sistema de balanceamento de carga para várias máquinas:

As requisições dos usuários será balanceada entre as diversas máquinas.

⁶<http://www.linux-ha.org>

Entretanto, um usuário irá receber uma mensagem de erro, caso sua requisição seja redirecionada para uma máquina defeituosa. Uma tecnologia amplamente utilizada para balanceamento de carga é o DNS Round-Robin.

[

- Implementar um sistema de alta disponibilidade e de balanceamento de carga:

As requisições de usuários serão distribuídas em um conjunto de servidores e caso ocorra falha em algum dos servidores, o mesmo não receberá mais requisições de usuários. As tecnologias mais utilizadas para implementar soluções deste tipo são : lvs+ldirectord.

Assim, é preciso garantir que a informação será a mesma não importando qual servidor ela estará sendo acessada. De maneira que todo o conteúdo, seja ele estático ou dinâmico, publicado deverá estar disponível, sincronizado e idêntico em todos os servidores.

Informações mais detalhadas sobre estas tecnologias serão apresentadas no capítulo XIII

Cenário 2 - Mineração de Dados

Nos últimos anos a informatização dos meios de produção e as novas formas de comunicação possibilitaram a geração de grandes volumes de dados nas mais diversas instituições, sejam públicas ou privadas. Grande parte das informações armazenadas nessas bases de dados permanece ainda desconhecida, uma vez que as ferramentas tradicionais de extração não permitem uma completa visualização de todas as correlações possíveis, tornando o processo demorado, dispendioso e pouco automatizado.

O aproveitamento de tais informações armazenadas permite o desenvolvimento de estratégias que possibilitam aumentar a competitividade das organizações. Especificamente no público, a produção de conhecimento a partir das bases de dados propicia melhor suporte à tomada de decisão, tendo por consequência a promoção da eficiência da Administração.

Nesse contexto, a automatização dos processos de análise de dados, com a uti-

lização de software específico, diretamente conectado à massa de informações, tornou-se uma grande necessidade, a qual aplicação de técnicas de mineração de dados propõe-se a dirimir. Mineração de dados é compreendida como a exploração e a análise, por meio automático ou semi-automático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativos⁷.

O processo de mineração de dados tem como principais objetivos descobrir relacionamentos, geralmente não triviais, entre dados armazenados, fornecer subsídios para que seja possível realizar previsão de tendências futuras com base nos registros acumulados, bem como estabelecer parâmetros para análise e auditoria das informações coletadas.

A realização de um processo de mineração de dados, geralmente emprega algoritmos de alta complexidade os quais podem realizar tarefas de classificação, estimativa, associação, segmentação ou agrupamento de informações, com possibilidade de consultas à bases de dados não estruturadas.

Dessa forma, à medida que a base de dados aumenta, as possibilidades de relacionamentos e combinações de tarefas cresce exponencialmente.

Por essa razão, existe o desafio de promover um ambiente computacional favorável ao processo de mineração de dados para que os resultados possam ser obtidos em curto espaço de tempo.

Existem 2 tipos de Cluster que podem auxiliar na resolução deste problema:

- Cluster de Processamento de Alto Desempenho: Implementação dos algoritmos de manipulação dos dados utilizando-se de técnicas de exploração de paralelismo e sistemas de processamento distribuído de alto desempenho, com o objetivo de melhorar o tempo de mineração dos dados através da distribuição das operações realizadas entre um conjunto de servidores. As principais tecnologias utilizadas para esta finalidade são: MPI, PVM e OpenMosix.
- Cluster de Banco de Dados: A idéia aqui é clusterizar a camada de banco de dados da aplicação, fornecendo a aplicação uma maior capacidade de

⁷ BERRY, M. J. A.; LINOFF, G. Data mining techniques – for marketing, sales, and customer support. John Wiley & Sons, New York, 1997.

armazenamento disponível e um acesso aos dados mais rápido. Estas questões são obtidas através da utilização de tecnologias de banco de dados distribuído, replicado, ou paralelização de consultas SQL. As soluções mais conhecidas para auxiliar na resolução deste problema são: Mysql Cluster, PgCluster, slony, Sequoia e Pargres.

O governo federal financiou o desenvolvimento do tamanduá, um software de mineração de dados em Cluster, este software foi licenciado sobre os termos de licenciamento GPL. Permitindo a sua utilização, distribuição, alteração e distribuição de versão alterada do software. O tamanduá é um serviço web de mineração de dados, possui uma arquitetura escalar, modular e realizar o processamento da mineração através de algoritmos de processamento paralelo baseados na Biblioteca de Processamento paralelo PVM.

Maiores informações sobre o software de mineração tamanduá podem ser encontradas na página oficial do projeto:

<http://tamandua.speed.dcc.ufmg.br/>

Cenário 3 - Processamento de Alto Desempenho

Aplicações que demandam uma grande quantidade de recurso de processamento podem obter ganhos expressivos se utilizarem paradigmas de programação paralela e sistemas de distribuição do processamento em Cluster. Alguns exemplos são: aplicações de processamento científico, simulações, análises e comparações de dados/informações, entre outras.

Atualmente existem dois principais tipos de cluster para processamento de alto desempenho, sendo que cada um possui suas próprias características:

- **Bibliotecas de Programação Paralela:** Neste tipo de cluster de processamento de alto desempenho é necessário adaptar o software para utilizar as bibliotecas de programação paralela que compõem o cluster. As bibliotecas mais utilizadas são o MPI e o PVM. Não é simples portar aplicações existentes para utilizar este tipo de cluster, pois a lógica computacional normalmente utilizada no desenvolvimento de sistemas ou aplicações é sequencial,

sendo preciso antes de mais nada realizar uma análise da aplicação com o objetivo de encontrar pontos no sistema de maior demanda computacional e possibilidades de paralelização, utilizando técnicas de programação paralela e exploração do paralelismo de forma a obter o melhor desempenho possível em um cluster de processamento de alto desempenho.

- Sistema de imagem única (SSI): Neste tipo de cluster de processamento de alto desempenho, o sistema de imagem única simula uma única máquina com todos os recursos computacionais das máquinas presentes no cluster. Isto acontece geralmente de maneira transparente para as aplicações e dependendo do sistema utilizado, teoricamente, se a aplicação é capaz de utilizar mais de um processador ela será capaz de ser utilizada no cluster sem a necessidade de realizar alterações na aplicação.

Os sistemas de SSI mais utilizados são: Mosix, Openmosix, OpenSSI e Kherighed. Cada um destes sistemas realiza a migração de processos de maneira diferenciada, não sendo possível atualmente realizar a migração de qualquer tipo de aplicação ou processo, devido as limitações de cada sistema. Entretanto, existem muitos casos onde a adaptação do sistema para execução em um cluster de processamento baseado em bibliotecas de programação paralela pode ser custoso ou inviável e que é possível executar a aplicação em um cluster SSI, obtendo um desempenho melhor que o da aplicação serial, entretanto não tão otimizado quando se a aplicação fosse programada para ser paralela.

Um exemplo de aplicação que poderia envolveria um grande esforço de adaptação e migração para um cluster de bibliotecas de programação paralela e que poderia ser utilizado em um cluster ssi facilmente é: um programa de simulação que recebe a entrada de diversas condições iniciais e demora 10 dias para retornar o resultado da simulação. Em um cluster SSI, poderiam ser executadas várias cópias do programa com arquivos de entrada e saída diferentes que seriam automaticamente distribuídos no conjunto de máquinas do cluster. Este tipo de aplicação não teria nenhum ganho no tempo de processamento de uma cópia do programa pois o programa não é paralelo, entretanto ao final do prazo de execução teria-se um acervo maior de resultados para posterior análise e utilização.

As tecnologias de processamento de alto desempenho devem ser utilizadas nas seguintes situações:

- Se a aplicação puder obter ganhos na utilização do sistema de imagem única sem necessidade de alteração. No caso da instituição não possuir recursos ou permissão para realizar alterações na aplicação.
- Se a melhoria de performance e resultados da paralelização da aplicação justificar a necessidade de adaptação e re-desenvolvimento da aplicação explorando as possibilidades de paralelismo.

Atualmente a Petrobras é a maior usuária de processamento de alto desempenho em cluster do Brasil, sendo que possui 3 dos 4 supercomputadores da América do Sul que encontram-se atualmente na lista 500 supercomputadores mais rápidos do Mundo⁸. Estes 3 supercomputadores são clusters de processamento de alto desempenho utilizados para a execução de seus sistemas de exploração petrolífera.

Cenário 4 - Alta Disponibilidade

Atualmente, sistemas de tecnologia da informação são responsáveis pela execução das mais variadas atividades administrativas, financeiras, de gestão de pessoas e até mesmo de comunicação na maioria das instituições públicas. Neste ambiente dependente de tecnologias, uma possível falha ou indisponibilidade em algum servidor ou serviço, acarretará a impossibilidade de realizar alguma atividade ou até mesmo prejuízo financeiro para a instituição.

Um fator importante a ser levado em consideração na preparação de infraestrutura para qualquer serviço ou aplicação é o fato de que todo e qualquer hardware, software ou aplicação estão sujeitos a falhas. Sejam por conta de problemas nos componentes eletrônicos, problemas de desenvolvimento do software/aplicação ou até mesmo erros resultantes de uma interação errada dos usuários ou administradores do ambiente.

Existem basicamente duas alternativas, do ponto de vista do servidor que hospeda uma determinada aplicação, para se conseguir um maior nível de disponibilidade:

⁸Top 500: <http://www.top500.org>.

1. Implementação de mecanismos de redundância e tolerância à falhas no hardware. Alguns exemplos são: Fontes Redundantes, Espelhamento de discos, Utilização de Tecnologias HotSwap para troca de componentes do servidor, entre outras. Esta abordagem normalmente é responsável por aumentar os custos associados à aquisição dos equipamentos de informática.
2. Implementação de mecanismos de tolerância à falhas via software. Esta abordagem consiste em utilizar um sistema de alta disponibilidade em software, capaz de gerenciar um conjunto de servidores de forma que a falha em algum dos servidores não afete a execução da aplicação que é controlada pelo sistema de alta disponibilidade. Devido as questões relacionadas a alta disponibilidade serem tratadas em software, em geral, podem ser adquiridos hardwares commodity, normalmente com custo menor que os hardwares utilizados na alternativa 1. Exemplos destes sistemas são: HeartBeat, Mon, Carp, entre outros.

O sistema de alta disponibilidade com maior maturidade e mais utilizado no sistema operacional linux é o Heartbeat⁹. Este sistema pode ser utilizado para controlar a parada e o início de "serviços" ou a execução de comandos em um conjunto de máquinas. Em conjunto com o Heartbeat é necessário utilizar alguma tecnologia que seja responsável por replicar e garantir a integridade dos dados entre os dois ou mais servidores, em geral o software mais utilizado é o drbd¹⁰.

A figura ?? é um diagrama onde 4 clientes estão acessando uma aplicação que encontra-se hospedada em um conjunto de alta disponibilidade. A aplicação encontra-se ativa somente no servidor primário e todos os dados salvos no disco do servidor primário são replicados para o servidor secundário. Em caso de falhas no servidor primário, o heartbeat será responsável por tomar as ações necessárias para que o servidor secundário passe a executar a aplicação e servi-la aos clientes. Os clientes enchem apenas um servidor através de um endereço ip compartilhado entre os dois servidores.

Esta solução é estável, simples de implementação simples e utilizada em produção em todo o mundo. Entretanto uma requisição enviada ao servidor primário antes de sua falha que envolva alguma atividade de escrita (email, banco de dados, servidor de arquivos, etc.) e não tenha sido gravada no disco do servidor

⁹<http://linux-ha.org>

¹⁰<http://www.drbd.org>

primário, será perdida quando o servidor secundário assumir. Pois, o servidor secundário só irá possuir as informações que tiverem sido gravadas no disco do servidor primário e replicadas em seu disco.

Recomenda-se utilizar uma solução baseada em heartbeat+drbd+mon em sistemas de email, servidor de arquivos, servidor web e banco de dados. Sendo que devem ser tomados os devidos cuidados no sistema gerenciador de banco de dados para que não sejam perdidas requisições de transação.

Cenário 5 - Banco de Dados

A camada de banco de dados é uma das partes críticas da maioria dos sistemas. Uma falha, indisponibilidade ou problemas de integridade na camada de banco de dados pode ser responsável pela indisponibilidade de um sistema inteiro ou até mesmo pela perda de dados que encontravam-se armazenados. Por conta desse motivo, esta camada deve ser avaliada, desenvolvida e implementada com cuidado.

Existem Diversas tecnologias de cluster para banco de dados, sendo que as principais podem ser classificadas em:

- Alta Disponibilidade: Nesta categoria encontram-se as tecnologias de replicação e alta disponibilidade. Para replicação normalmente é utilizado alguma solução própria ou específica para o sistema de banco de dados utilizado. No caso do postgres pode ser utilizado o slony que provê replicação do tipo ativo/passivo. O mysql possui um recurso próprio para replicação de tabelas e dados entre servidores. Para alta disponibilidade pode ser utilizado por exemplo o Heartbeat+DRBD.
- Paralelização de Consultas SQL: Nesta categoria encontram-se as tecnologias de paralelização de consultas SQL, cujo objetivo é aumentar a velocidade de processamento e execução de consultas sql complexas, particionando-a e distribuindo em um conjunto de servidores. Uma solução independente de plataforma de banco de dados é o Pargres¹¹, desenvolvido para ser utilizado principalmente em aplicações OLAP e DataWareHouse.

¹¹<http://http://pargres.nacad.ufrj.br/>.

- Distribuição do banco e balanceamento das requisições: Este tipo de tecnologia é utilizada normalmente em grandes aplicações transacionais, onde é necessário aumentar a performance e disponibilidade. As soluções mais conhecidas são o pgCluster, específico para o postgres e o Sequoia¹², solução em java independente de sistema de gerenciamento de banco de dados.

Maiores informações sobre as tecnologias disponíveis para cluster de banco de dados podem ser encontradas no capítulo 9.

3.4 Arquitetura para sistemas críticos online em N Camadas

3.4.1 Definição

A Arquitetura para sistemas críticos online em N camadas é um conceito que vem ganhando credibilidade no mercado. Em sistemas mais voltados para serviços web, suas principais vantagens são o fato de ser baseada totalmente em padrões abertos e ser uma arquitetura extremamente modular, capaz de se adaptar aos mais diversos cenários computacionais.

Tal arquitetura é composta por N Camadas e pode ser composta:

- Camada de Aplicação Web
- Camada de Banco de Dados
- Camada de Armazenamento

Cada uma destas camadas será melhor exemplificada nas subseções abaixo:

¹²<http://www.continuent.org>

3.4.2 Camada de Aplicação

Esta camada é responsável pelos serviços web disponíveis no cluster. É nela que se encontram os servidores de aplicação e é a única camada acessada externamente pelos usuários dos serviços.

Nesta Camada são utilizadas tecnologias de Cluster de Aplicação, Balanceamento de Carga e Alta Disponibilidade. A tabela **D** apresenta uma lista das tecnologias utilizadas.

As principais características são: Alta Disponibilidade, Escalabilidade, Balanceamento de Carga, Alta Performance.

3.4.3 Camada de Banco de Dados

Esta camada é responsável pelos bancos de dados que são acessados pelos serviços disponíveis no Cluster. É nesta camada que se encontram os servidores de Banco de Dados.

Nesta Camada são utilizadas tecnologias de Cluster de Banco de Dados e Replicação de Banco de Dados. A tabela **D** apresenta uma lista das tecnologias utilizadas.

As principais características são: Alta Disponibilidade, Balanceamento de Carga, Alta Performance e Integridade dos Dados.

3.4.4 Camada de Armazenamento

Esta camada é responsável pela consolidação do armazenamento do Cluster, ela deve simular/funcionar como um storage externo. É nesta camada que se encontram os servidores de Armazenamento.

Nesta Camada são utilizadas tecnologias de "Distributed Mass Storage", Sistemas de arquivos compartilhados, Dispositivos de Blocos em rede, entre outras. A tabela **D** apresenta uma lista das tecnologias utilizadas.

As principais características são: Tolerância à falhas, Alta Disponibilidade, Integridade dos Dados, Alta Performance e Arquivos Distribuídos.

3.4.5 Diagrama da arquitetura proposta

A figura abaixo apresenta um diagrama da arquitetura proposta na seção 3.4

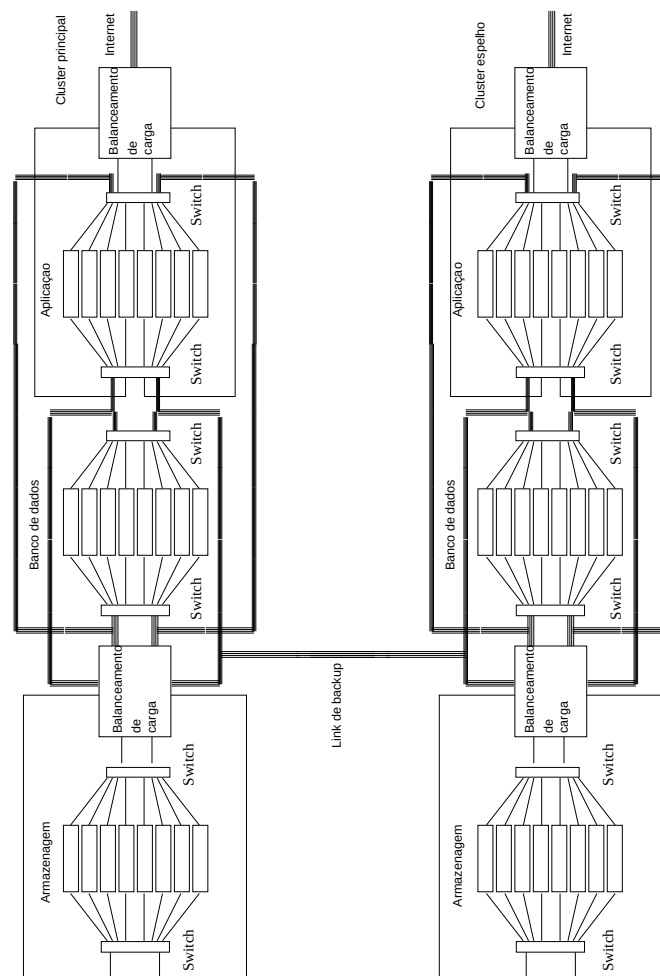


Figura 3.4: Esquema do modelo de cluster proposto.

3.4.6 Considerações sobre a arquitetura

Esta arquitetura foi pensada para que fosse o mais geral e modular possível, alguns exemplos de utilização desta arquitetura podem ser:

- Implementação da camada de aplicação através de tecnologias de cluster, Implementação da camada de banco de dados através de uma máquina RISC de grande Porte, Implementação da camada de armazenamento em Cluster.
- Implementação da camada de aplicação através de máquina RISC grande porte e/ou Mainframe, Implementação da camada de banco de dados em Cluster, Implementação da camada de armazenamento através do uso de Storage Externo.
- Implementação da camada de aplicação, banco de dados e armazenamento em Cluster.

Capítulo 4

Visão Geral

4.1 A sensibilização

A escolha por desenvolver um sistema crítico para ser utilizado em cluster é uma decisão complicada e tem a sua sustentabilidade em muitos aspectos, não apenas técnicos, mas também estratégicos e econômicos e devem estar contextualizada em uma política tecnológica. A decisão sobre o desenvolvimento e o uso de Clusters sofre também influências de caráter cultural, e estas podem ser mais limitadoras do que o próprio emprego da tecnologia.

Mudar sistemas, alterar soluções e plataformas, em geral, são tarefas complexas. Ao considerarmos que toda mudança capaz de modificar o comportamento e as rotinas das pessoas aumenta o grau de dificuldade das tarefas, podemos afirmar que, ao se falar em inovação, a atenção dos Administradores não pode se concentrar exclusivamente na parte técnica. A inovação exige também esforço de mudança cultural, o que nas organizações se retrata diretamente no que se concebe como Cultura Organizacional.

4.2 Os Recursos Humanos Envolvidos

4.2.1 Aperfeiçoamento dos Técnicos

Antes de capacitar a equipe técnica e de desenvolvimento, é preciso reuni-los e explicar os motivos da mudança de plataforma. É importante conquistar todos envolvidos no projeto e conscientizá-los sobre os motivos que levaram a instituição a escolher essa nova tecnologia e as melhorias que essa mudança irá ocasionar. Esta é uma atividade essencial na chamada Sensibilização.

Adotar uma nova tecnologia, como a de clusters, necessita de um plano de capacitação que contenha profissionais especializados para viabilizar a difusão do conhecimento dessa tecnologia entre os funcionários da instituição, para que estes aceitem mais facilmente a implantação, absorvam o conhecimento nas regras de negócio do sistema e possam manter todo o ambiente operacional sem a necessidade e a dependência de agentes externos. Identificar os perfis adequados com a tecnologia de clusters que será implantada, como por exemplo: sistemas distribuídos, sistemas paralelos, banco de dados distribuídos, Grid e depois formar uma programa de treinamentos, antes da implantação, e de acordo com essas áreas, que são necessários para a formação e compreensão dos envolvidos.

4.2.2 Consultoria

A utilização de Cluster não é simples, e deve ser bem conhecida em todos os níveis da organização de TIC da instituição. A opção e o projeto utilizando desta tecnologia é um estudo complexo que precisa ser bem feito e avaliado por todas as esferas, para que possa ter sucesso e trazer benefícios à instituição.

A contratação de uma consultoria especializada pode diminuir os riscos de erros e gastos excessivos no projeto. Consultores especializados podem, em conjunto com funcionários da empresa, participar do planejamento, da avaliação das possibilidades de utilização de tecnologias de clusters dentro do ambiente, analisar a situação atual, indicar os pontos críticos do projeto. Esse processo de consultoria

tem de prever uma interação dos “conhecedores do problema”¹ com os especialistas em tecnologias de cluster para que em conjunto possam obter a melhor solução para os Sistemas que irão rodar no ambiente clusterizado.

4.3 O Projeto de Cluster

Quando se prepara para projetar ou planejar um sistema que vai rodar em cluster para ambientes empresariais (ou ambientes de produção, que não são flexíveis como os ambientes acadêmicos) é aconselhável se preparar para poder responder e respaldar várias tecnologias, metodologias e informações. A opção por uma ou outra tecnologia pode ser o marco divisor entre o sucesso ou não do projeto. Assim, alguns cuidados tem de ser tomados na hora de reconhecer o ambiente no qual se está optando.

Várias dicas são dadas no artigo: “Ten Tips for Building Your First High-Performance Cluster” ou em tradução livre “Dez macetes para construir seu primeiro cluster de Alto-desempenho” escrito por Joseph D. Sloan e publicado em “12/29/2004” na http://www.linuxdevcenter.com/pub/a/linux/2004/12/29/lxclstrs_10.html, que tem foco apenas em sistemas de alto desempenho.

Aumentado a complexidade e procurando expandir a idéia pensando em estruturas empresariais mais complexas, que podem ser conhecidas como “enterprise”, e com base apenas em tecnologias abertas.

Algumas características que devem ser observadas nesse modelo são:

- Escalabilidade do ambiente: Capacidade sob demanda para atender os requisitos de recursos de infra-estrutura;
- Balanceamento de carga: Capacidade de realocar as cargas de trabalho no caso de falhas dos sistemas, tanto de Hardware como de software, e também aumentar o desempenho global do sistema;
- Permitir separação de ambientes e ou aplicativos dentro de uma partição,

¹Pressupõe-se nesse ponto que a instituição detém todo o conhecimento das regras de negócios do seu ramo de atuação, tendo capacidade em modelar os sistemas necessários a ela.

para eliminar a contenção e alocar desempenho para sistemas específicos;

- Gerenciamento da carga de trabalho, permissão para estabelecer critérios de desempenho para cargas de trabalho específicas com base em suas regras de negócios e ajustar os recursos de processamento para atingir estas metas.

Essas opções não só estimulam a eficiência econômica, mas também permitem maior visibilidade de como os recursos de computação devem ser alocados para apoiar processos de negócio estratégicos em tempo real, eliminando a subutilização e os custos indiretos dela decorrente.

4.3.1 O que deve ser observado

A construção deste tipo de sistema é complexa e é necessário conhecer muito bem o problema para propor a melhor solução. Clusters de alto-desempenho provêm freqüentemente o modo de melhor custo benefício para acelerar cálculos, ou em outras palavras, a computação de alto desempenho, mas a construção do primeiro cluster pode ser uma experiência difícil. Os desafios para a construção de uma infra-estrutura deste tipo é grande e muitas variáveis tem de ser observadas e trabalhadas para se obter, além do melhor desempenho, um menor custo de investimento. O pensamento é semelhante quando em sistemas “enterprise”, mas com um grau de complexidade maior, pois estamos tratando de um ambiente que tem de ser estável, robusto, escalável e capaz de responder por toda a carga de processamento projetada.

O tempo de vida² das aplicações desenvolvidas para clusters “enterprise” tem a tendência de ser maior, podendo ter ciclos de mais de 10 anos de operação. Nestes casos a escolha das tecnologias a serem usadas terão grande importância para as bases do projeto.

Entre muitas outras informações e detalhes de projetos, alguns considerados mais importantes são levantados e discutidos nas próximas sessões, a listar:

1. Estabeleça metas realistas

²Ciclo de vida e de desenvolvimento dos sistemas

2. Projeto piloto
3. Utilização hardware idêntico
4. Sistemas diskless
5. A importância da rede de comunicações
6. Minimize mas não sub-dimensione seu hardware
7. Isole seu cluster
8. Use ferramentas de cluster
9. Supere o desejo do mais recente
10. Plano para expansão e reposição desde o princípio
11. Documente seu cluster
12. Segurança
13. A aplicação
14. Banco de dados

Estabeleça metas realistas

O primeiro passo na construção de um cluster de alto-desempenho é realizar um planejamento visando o que se pretende e decidir quais as metas a serem atendidas, tanto no curto como no longo prazo. É preciso selecionar o hardware apropriado e determinar de quais softwares precisarão seus usuários. Claramente, estas decisões afetarão seu planejamento. Por exemplo, para cálculos intensivos em dados, é necessário um subsistema de I/O de grande capacidade e desempenho, mas em ambientes WEB a resposta dos servidores WEB pode ser a métrica e para banco de dados a quantidade de transações suportadas.

Não é aconselhável iniciar o desenvolvimento da aplicação e nem do cluster, antes de conhecer seu problema. Faça um levantamento de seus sistemas e tenha pessoas que conheçam ambas as áreas para participar do projeto do cluster.

Em caso de aplicações existentes, que se queira portar para este ambiente, pesquise as possibilidades pois, certamente, o porte de aplicações mais antigas (legados) custará muito mais caro do que o desenvolvimento de uma nova aplicação em tecnologias mais recentes. Mas ainda sim, a avaliação de cada uma aplicação precisa ser feita. Podem ocorrer casos de aplicações (neste caso, problemas) que nem mesmo com o emprego de tecnologias mais recentes seja possível clusterizar.

As metas de desempenho desejadas (o crescimento deste) a longo prazo também são uma métrica a ser utilizada, podendo até mesmo se tornar a linha mestra para o projeto de todo o sistema. As capacidades tem de ser bem planejadas e conhecidas desde o início do desenvolvimento do projeto, pois estas que indicarão as possíveis tecnologias a serem adotadas para se obter uma equalização de desempenho e custo total de implantação. Ressalta-se que não se pode pensar neste momento do projeto apenas nas capacidades imediatas do sistema. Deve-se também programar o crescimento de demanda a longo prazo, picos de utilização do sistema, que em alguns momentos pode explodir a capacidade instalada, sistema de recuperação de desastres, entre outras variáveis.

Projeto piloto

O planejamento de um projeto ou cluster pode ser difícil se não há conhecimento e experiência em clusters. Só com a prática e experiência que poderemos ser capazes de entender as capacidades e limitações de um cluster. Iniciar com um projeto pequeno pode ser a melhor forma para evitar enganos e erros, e conseqüentemente, gastos excessivos.

Construir um sistema de teste com um número pequeno de máquinas e com o modelos do seu sistema, permitirá determinar o que é mais preciso antes de assumir compromissos que podem ser equivocados. Isto provavelmente irá economizar tempo e dinheiro, já que corrigir enganos em um cluster de grande porte e em produção pode ser muito demorado e principalmente ter um custo muito elevado.

O domínio da tecnologia também é importante, e um projeto piloto pode ser utilizado para várias outras aplicações, como plataforma de desenvolvimento de sistemas, testes de configurações, projeção de estresse de sistemas e homologia-

ção de aplicações, entre muitas outras coisas.

Utilização de hardware idêntico

Certamente há exceções a esta regra. Você certamente precisará de um nó frontal mais rápido para seu sistema, da mesma maneira que também irá querer ter discos de maior capacidade em servidores de I/O.

No entanto, a utilização do mesmo hardware em cada máquina do cluster traz muitas facilidades:

- Simplificar a instalação e a configuração de seus clusters, já que poderá ser usado imagens idênticas do sistema para cada máquina.
- Simplificar a manutenção do cluster, pois todos os sistemas têm a mesma configuração básica. Assim, deve ser preciso manter poucas peças sobressalentes que poderão ser trocadas rapidamente, caso o hardware do equipamento apresente defeitos.

Existe também a idéia de Cluster segmentado, ou cluster de N camadas, no qual se teriam vários clusters que, trabalhando em conjunto, seriam responsáveis pela infra-estrutura "enterprise". Por exemplo, uma camada apenas para ser responsável pelo armazenamento (cluster storage, SAN), uma camada apenas para banco de dados, uma camada apenas para aplicações, uma camada para firewall e proxy, e assim modelar o ambiente para atender as especificidades dos sistemas utilizados no ambiente. Mais detalhes deste tipo de ambiente pode ser melhor visto na sessão 3.4. Seguindo assim essa característica de camadas de cluster, cada uma delas tenderia a ter um único tipo de hardware.

Sistemas diskless

Sloan, em seu artigo [332], aconselha a se evitar a utilização de sistemas que não utilizam disco rígidos; no entanto, a experiência e a evolução destes sistemas vêm mostrando que a sua eficiência pode ser muito bem aproveitada, além de

se facilitar o gerenciamento de todo o sistema de forma global. Mesmo assim, a utilização deste tipo de tecnologia precisa ser muito bem avaliada para verificar os prós e contras de uma implementação baseada em tecnologia sem disco.

A importância da rede de comunicações

Uma reflexão tem de ser feita antes de começar a pensar em redes: um dos grandes erros em projetos de clusters, para qualquer que seja a sua finalidade, é acreditar que o processamento, ou alta capacidade de processamento, é baseado apenas em computadores, ou em seus processadores. Um cluster tem de ser visto como um organismo completo, onde cada peça tem sua importância e pode ser o responsável por um melhor desempenho do sistema globalmente.

E é exatamente nesse aspecto que a rede de comunicações tem sua importância na hora de projetar o cluster. A rede é normalmente o gargalo de desempenho para clusters que utilizam hardware não proprietário, ou hardware de prateleira. Assim é importante tomar cuidado e colocar a camada de rede como uma das partes de grande importância no projeto. A criação de camadas separadas para dados e para gerenciamento dos sistemas, a possível utilização de várias interfaces de rede, ou outras tecnologias de comunicação, precisam ser avaliadas para as características que se deseja obter. Com os baixos preços das placas Gigabit Ethernet, a sua utilização vem se tornando freqüente nesses tipos de sistemas.

Minimize mas não sub-dimensione seu hardware

Ao especificar o hardware dos nós computacionais de um cluster, há muita coisa a ser observada. Dependendo do ambiente e do número de máquinas que o cluster irá conter, decisões como utilizar ou não "HACKS" serão de grande importância, assim como é uma tendência em ambientes de grande porte a utilização deste tipo de infra-estrutura para melhorar utilização do espaço físico e facilidade de manutenção, para pequenos ambientes será um gasto desnecessário.

Na especificação do hardware dos nós, certamente, não precisará de coisas como placas de som, aceleradoras 3D, mouses, teclados e monitores. Uma idéia criativa é montar, em um carrinho, um monitor, teclado, e mouse que possam circular

pelo ambiente para fazer a checagem de máquinas individuais quando problemas surgirem. Se estiver comprando equipamento, minimizar o hardware pode baixar seu custo total de forma a permitir comprar mais máquinas.

Mas existe um limite à esta “minimização” das especificações de hardware, certamente uma placa de vídeo será necessária no caso de que se precise dar manutenção em alguma máquina, assim como um CD-Rom pode fazer falta para a instalação de softwares e do sistema operacional. Portanto, ter esses equipamentos ou possibilidades de solução para esses problemas é de extrema importância para o ambiente, não pode ser obrigatória a necessidade de se abrir máquinas para adicionar uma placa de vídeo ou cd-rom para poder resolver qualquer tipo de problema, e o custo envolvido não é tão grande para se evitar a utilização destes equipamentos.

Isole seu cluster

Não existe nenhuma razão para todos os nós do cluster estarem visíveis em sua rede local. A existência de uma rede separada para o cluster tem por razão a segurança das informações e dos sistemas que nele são executados. Com esse isolamento, pode-se principalmente preocupar com o desempenho do sistema, afrouxando as preocupações com correções de segurança. Repare que não está sendo dito que o isolamento do cluster evita problemas de segurança, mas sim que muitas falhas de segurança em servidores em redes públicas são críticos, em um ambiente isolado e sob controle não terão as mesmas repercussões, possibilitando assim melhoras na disponibilidade dos sistemas.

Se for preciso obter acesso à rede do cluster, este pode ser provido através de conexões seguras por um firewall e por uma máquina de entrada, responsável pelo disparo de processos no sistema. O controle de acesso e conexões ao cluster são temas que têm de ser bem estudados, e dependendo do tipo e principalmente do valor desta informação, o controle tem de ser mais acurado. Nesse ponto, o controle de acesso iria muito além dos firewalls, proxies e servidores de autenticação, mas também passaria pelo estabelecimento de conexões seguras e autenticadas, certificados digitais, entre outras tecnologias.

Use ferramentas de cluster

A utilização de ferramentas, que automatizam as tarefas de instalação e administração de cluster podem ser uma solução agradável para os administradores de sistemas, mas é preciso dominar e entender essas ferramentas com profundidade.

Ferramentas como o OSCAR³, Rocks⁴ e XCAT⁵, simplificam a instalação e o gerenciamento de clusters, neste caso cluster de processamento de alto desempenho. Qualquer um destes pacotes provavelmente instalará e configurará boa parte das suas necessidades básicas.

Assim como estes pacotes, existem outros que facilitam a utilização de clusters, como o RHCS⁶ que é apontado para sistema de HA.

Supere o desejo pelo mais recente

Tenha como meta a ser alcançada um cluster em funcionamento, que atenda de melhor forma as necessidades levantadas. Sendo assim, é muito improvável que não se precise da versão mais recente de distribuições Linux. Equipamentos de cluster estão baseado nas distribuições de Linux mais comuns, mas leva tempo para se adaptarem aos novos lançamentos. Na maioria dos clusters, os usuários notarão diferenças apenas no nó de entrada do sistema. Para os nós de trabalho (ex., o tamanho do cluster), não importa qual distribuição de Linux está executando, contanto que execute a tarefa para a qual foi projetado.

Plano para expansão e reposição desde o princípio

O ciclo de vida de equipamentos de informática é curto, e isto fica muito evidente quando se começa a pensar na evolução do cluster, de como gerenciar toda a necessidade de expansão com a viabilização tecnológica e técnica necessária. Vários

³Mais informações podem ser vistas em <http://oscar.openclustergroup.org/>

⁴Mais informações podem ser vistas em <http://www.rocksclusters.org/>

⁵Mais informações podem ser vistas em <http://www.xcat.org/>

⁶Mais informações podem ser vistas em <https://www.redhat.com/solutions/clustersuite/>

pontos têm de ser levantados, como escalabilidade, compatibilidade, custo, assim como os motivos para a expansão.

A evolução do cluster para aumentar as capacidades, a escalabilidade da solução utilizada tem de ser observada, para saber, no mínimo, quais as limitações da arquitetura que pretende-se implantar. Outros pontos também precisam ser levantados para a expansão planejada, como a aderência de hardwares de modelos diferentes, espaço físico, capacidade de refrigeração, etc. A vantagem, no caso de expansão ou na troca de todo o cluster, é que boa parte do equipamento pode ser reciclada para outras tarefas dentro do sistema.

No caso de estar trabalhando na expansão de um cluster em funcionamento, o estudo deste será de grande importância para saber como a aplicação é executada no ambiente existente, fornecendo um grande volume de informação valiosa para os novos projetos.

Documente seu cluster

Documentação bem feita e completa é a chave para o aperfeiçoamento do uso de seu cluster atual e para projetos futuros. Se estiver instalando equipamentos, mantenha o registro da informação de configuração, de rede, dados históricos de tráfego, utilização e principalmente de problemas.

Muitas vezes passamos por problemas que já foram resolvidos em ocasiões anteriores, mas por falta de um histórico de ocorrências, não sabemos como resolver o problema, o que obriga a todo um retrabalho de pesquisa por soluções dos problemas apresentados. As documentações são de extrema importância nesses momentos, mas as principais documentações ainda são as relacionadas ao próprio cluster. Deve-se conhecer como as conexões de rede e energia estão feitas, quais as configurações e todos os detalhes técnicos da implementação, para ajudar a prever problemas, bem como facilitar em muito o processo de resolução de qualquer incidente.

Segurança

Muitos projetos esquecem de trabalhar com a segurança dos ambientes de uma forma integrada, ou seja, a segurança pensada tanto na interface de hardware como na de software. A ISO 17799 "Tecnologia da Informação - Código de Prática para Gestão da Segurança de Informações" aborda vários aspectos da segurança que tem de ser observados para uma boa prática desta. Entre outros itens, ela aborda:

- Política de Segurança;
- Segurança Organizacional;
- Segurança Relacionada ao Pessoal;
- Segurança Física e Ambiental;
- Gerenciamento de Comunicações e Operações;
- Controle de Acesso;
- Desenvolvimento e Manutenção de Sistemas, que levanta itens como:
Requisitos de Segurança nos Sistemas,
Análise e Especificação dos Requisitos de Segurança,
Segurança em Sistemas Aplicativos,
Validação dos Dados de Entrada,
Controle do Processamento Interno,
Segurança de Arquivos do Sistema,
Controle de Software Operacional.
- Gerenciamento da Continuidade do Negócio;
- Obediência a Exigências.

A aplicação

No projeto e desenvolvimento da aplicação devem estar detalhados todos os processos e tecnologias que serão utilizados, uma aplicação bem projetada terá sucesso na sua execução em um ambiente de cluster.

Banco de dados

Conhecer bem as demandas de acesso aos dados pelos sistemas que serão executados no cluster permitirá uma melhor escolha da arquitetura de banco de dados necessária para suprir as exigências do sistema. Mais informações podem ser obtidas no capítulo [9](#)

Capítulo 5

Processamento Paralelo: Sua Difusão e Uso

Um problema central em computação paralela, segundo SKILLICORN [330], é o desencontro entre as necessidades do software paralelo e as propriedades das arquiteturas paralelas sobre as quais eles serão executados. Este distanciamento entre o hardware e o software paralelo oscila com rapidez, isto porque o tempo de vida de uma arquitetura paralela é, via de regra, medido em anos, enquanto que o tempo de vida desejável para qualquer software de grande porte é medido em décadas. Dentro de uma visão tradicional, o procedimento é, no espaço de alguns anos, reescrever o software à medida que uma nova tecnologia de arquitetura paralela é disponibilizada no mercado. A rescrita de código, dentre outros problemas, introduz custos.

Isso hoje é causado principalmente por causa da evolução rápida desta área da computação. Apesar da área de pesquisa já ser antiga, a sua aplicação em ambientes empresariais é recente e vem evoluindo muito rapidamente.

5.1 Aspectos para a Adoção do Processamento Paralelo

Nesta seção serão tratados aspectos que podem ser vistos como estímulos para adoção do processamento paralelo, seja a partir do ponto de vista da exaustão das arquiteturas seqüenciais em função dos limites impostos pela tecnologia atual, seja considerando as necessidades dos diferentes segmentos usuários.

5.1.1 Barreiras ao Crescimento da Frequência de Operação dos Processadores

Como regra geral, quando maior a frequência de operação do processador (*clock*), maior desempenho terá o computador. Porém é importante ter presente que, uma vez mantidos aspectos como conjunto de instruções, arquitetura, etc., o desempenho efetivo (registrado pelos *benchmarks* tradicionais, tais como MIPS, MFLOPS, SPECmarks) não irá crescer na mesma razão do *clock*. Outros aspectos precisariam acompanhar o crescimento do *clock*, como por exemplo a eficiência (largura de banda) de acesso à memória. Independente desta posição não otimista para a relação desempenho/*clock*, considerando o nível em que se encontra atualmente a velocidade de operação dos processadores, a mesma já enfrenta entraves tecnológicos para seu aumento.

Destacam-se os seguintes aspectos como limitadores para o crescimento do *clock* (HWANG [222]): **O consumo de energia e a conseqüente dissipação térmica:** os componentes projetados para *clocks* elevados (tais como SRAM ou ECL) apresentam índices de consumo e dissipação térmica bem mais elevados que os similares (DRAM, CMOS) para *clocks* mais modestos. **A dimensão do processador e seus componentes acessórios:** limitados pela velocidade da luz, os elétrons podem percorrer distâncias menores à medida que a duração do pulso de *clock* diminui. Um *clock* de 1 GHz (1000 MHz) limita a distância máxima de deslocamento dos elétrons à grandeza de centímetros (o valor exato depende das características do substrato semiconductor). Deste modo, para operar nesta velocidade se fazem necessários componentes eletrônicos altamente densos, bem como cuidados extremos com o comprimento dos condutores que os interligam. Esta elevada

densidade de integração e a restrição nas dimensões globais dificulta o fluxo do elemento resfriador (ar, água, etc.), o que, por sua vez, introduz severas dificuldades no processo de dissipação térmica. Além disto, é preciso considerar o fato que em frequências elevadas ficam potencializados os fenômenos de capacitâncias e indutâncias parasitas, os quais dificultam sobremaneira os níveis de integração dos semicondutores.

Estes dois aspectos independentes se interrelacionam no equilíbrio entre desempenho e possibilidade de operação estável e confiável. As arquiteturas de alto desempenho são utilizadas, quase sempre, em missões críticas e pelo seu custo não é usual mais do que uma unidade em cada instalação.

5.1.2 Largura de Banda no Acesso à Memória

O aproveitamento do crescente poder computacional dos modernos processadores esbarra no de fato que o fluxo de dados possível entre os mesmos e a memória não cresce na mesma proporção. Este comportamento foi denominado *Gargalo de Von Neumann*, e caracteriza que o poder de processamento disponibilizado para computação de um problema é limitado em função da taxa de transferência de dados e instruções entre a memória e o processador.

O uso de vários processadores na solução do problema faculta, com a soma de suas taxas de transferência individuais, a superação do *Gargalo de Von Neumann*. Existem diferentes formas de interligar diversas memórias a vários processadores na definição uma máquina paralela.

Cada estratégia de interconexão (vide item 6.6.2) tem implicações diretas em aspectos operacionais, tais como: emprego genérico (possibilidade de uso com desempenho desta máquina paralela a um número maior de naturezas de problemas), na sua escalabilidade e no seu custo, dentre outros (CULLER [128]).

5.1.3 Paralelismo Intrínseco do Mundo Real

Os fenômenos naturais são inerentemente paralelos. Deste modo, seria natural e direto expressar as computações pertinentes ao mundo real de forma paralela, ou ao menos de uma forma que não impeça o paralelismo. Escrever programas sequenciais, via de regra, implica impor uma ordem as ações que são independentes e que poderiam ser executadas concorrentemente.

Na programação sequencial, é inevitável arbitrar uma particular ordem na qual as ações são colocadas. Isto pode tornar o computador um impecilho para a percepção de novos conceitos. Some-se a isto o fato que situações nas quais a ordem de execução das ações é importante para o melhor entendimento do problema real, são difíceis de diferenciar daquelas nas quais a ordem de execução praticamente não importa (SKILLICORN [331]).

5.1.4 A Relação Custo-Benefício dos Processadores de Última Geração

Mesmo que a recente tendência histórica de crescimento da velocidade dos processadores se mantenha, a computação paralela possibilita para muitas aplicações, uma relação custo/benefício melhor do que a conseguida ao utilizar equipamentos com um só processador de última geração. Isto ocorre, em grande parte, devido aos custos de projeto e fabricação de cada nova geração de processadores.

A cada novo processador mais poderoso, o preço da geração anterior cai consideravelmente; desde modo, agrupar em um equipamento paralelo, processadores mais antigos provê uma alternativa computacional de custo competitivo. Tendo em vista que cada nova geração introduz um acréscimo de desempenho com magnitude da ordem de décimos, mesmo modestos agrupamentos de processadores não tão atuais, são viáveis no que diz respeito ao desempenho global. Este aspecto se potencializa ainda mais se a escolha tecnológica do hardware para interligação não apresentar custo elevado.

Esta tendência é, em parte, responsável pela popularidade das estações de trabalho em rede de alta velocidade (100 Mbps no mínimo) como alternativa de equi-

pamento para processamento paralelo (CULLER [128]). E ainda mais reforçada com as quedas de preços das interfaces de comunicação Gigabit e seus componentes como switches.

5.1.5 Aplicações Extremamente Complexas

Existem aplicações que demandam elevadíssimo poder computacional. Por mais poderoso que possa ser determinado processador, dentro do atual estado tecnológico, a combinação de vários destes em uma arquitetura para processamento paralelo, torna disponível maior capacidade de processamento que a possível com um único.

Como exemplo de aplicações que atualmente demandam grandes recursos computacionais destacam-se:

- inteligência artificial, incluindo redes neurais, robótica e reconhecimento de padrões;
- análise de elementos finitos, onde aparecem diversos tipos de equações diferenciais aplicadas a mecânica estática, eletromagnetismo, e dinâmica dos fluidos;
- simulação, onde se sobressaem as técnicas de Monte Carlo;
- processamento de sinais, envolvendo FFT (*Fast Fourier Transformation*) sobre grandes volumes de dados, processamento de imagens e processamento sísmico;
- algoritmos básicos em ciência da computação: classificação, busca e processamento de árvores e grafos;
- grandes bancos de dados com resposta em tempo real (OLTP *On Line Transaction Processing*);

Freqüentemente é sugerido que os equipamentos paralelos, sobretudo os de grande porte, são comprometidos com propósitos especiais. A idéia inerente a

esta afirmação é que somente um pequeno conjunto de aplicações poderia ser executado eficientemente em um hardware paralelo. A lista de aplicações acima indica exatamente o contrário; a ineficiência do processamento paralelo tem muito mais relação com as “dimensões do problema” do que com as particularidades de um domínio específico do conhecimento humano. Nos últimos dez anos os computadores paralelos tem sido programados com eficiência tanto para aplicações do mundo comercial como para o da pesquisa e desenvolvimento (MORSE [280]).

5.1.6 Suporte à Tolerância a Falhas

Muitas aplicações críticas (controle de tráfego aéreo, sistemas de controle industriais, automações bancárias, etc.) exigem um regime de operação sem interrupções. A existência de redundância de hardware, inerente às arquiteturas paralelas, oferece um suporte natural às técnicas de tolerância a falhas. Alguns processadores podem monitorar e registrar a operação do sistema, no momento que for detectado alguma disfunção, as partes envolvidas podem ter suas funções continuadas por outras. Deste modo, no caso de falhas, o equipamento paralelo pode manter a computação corrente, possivelmente ocorrendo tão somente uma diminuição no desempenho na prestação dos serviços (HWANG [222]).

5.1.7 Crescimento Modular

Esta característica diferencia fortemente as arquiteturas paralelas e distribuídas dos equipamentos de grande porte (*mainframes*) tradicionais. Nas arquiteturas com um único processador, o usuário no momento do crescimento da plataforma, precisa prever sua demanda no mínimo a médio prazo. Isto leva a um crescimento feito aos saltos. Logo após a expansão, é comum a instalação como um todo apresentar uma relação custo/benefício ruim. Essa relação pode ser vista na figura 5.1.7 que mostra a escalada dos custos ao longo do tempo para as duas plataformas (alta plataforma (mainframe) e baixa plataforma (cluster e máquinas padrões de mercado)) em relação a capacidade de carga do sistema. Pode-se ver nessa figura claramente os saltos dados, pelo excesso de capacidade de processamento. O arco cinza escuro na figura 5.1.7 mostra a demanda de processamento

ao longo do tempo, a linha vermelha mostra a linha de crescimento de custos (C1) para o ambiente em baixa plataforma e por ultimo os degrais cinza claro (C2) mostram o crescimento de custos para a plataforma alta.

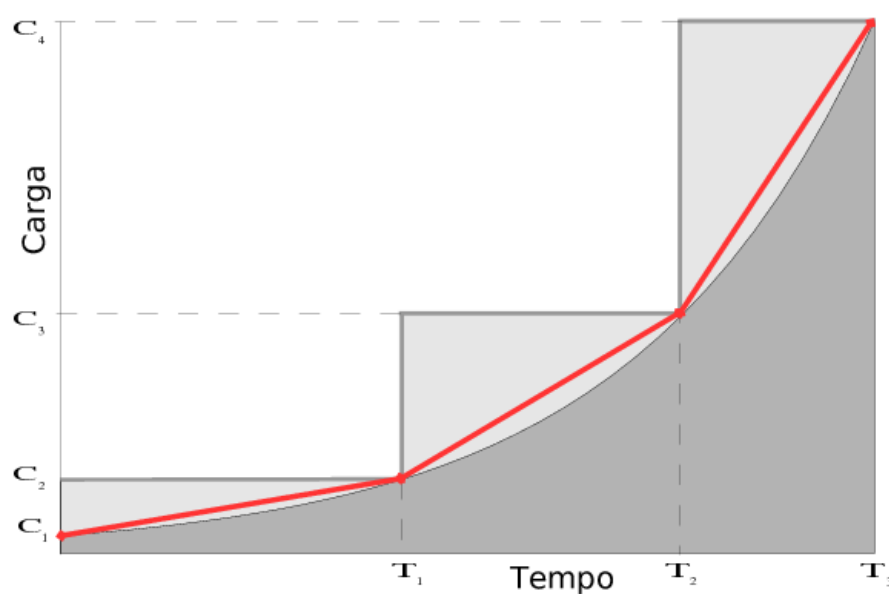


Figura 5.1: Relação Carga X Custo de investimento, para plataforma Baixa X Alta

Tanto para o fornecedor quanto para o usuário, é muito oportuno que a arquitetura possa ser expandida gradualmente através da adição de módulos. Esta possibilidade permite uma melhor adequação da curva investimentos & produtividade, uma vez que o equipamento poderá crescer dentro de uma determinada faixa, tendo como regulador a demanda de serviço real (MORSE [280]).

5.1.8 Disponibilidade de Software Aplicativo

É exatamente a disponibilidade de software de terceiros com qualidade (um número típico para as diferentes marcas seria 1500 aplicações) que tem potencializado o mercado das estações de trabalho de elevado desempenho. Por sua vez, a ausência de aplicações disponíveis no mercado tem sido um dos fatores a restringir a adoção de equipamentos paralelos por parte das empresas em geral. Poucas empresas, à exceção das instituições de pesquisa e ensino, não se intimidam ante os esforços para portar e/ou desenvolver software para exploração do paralelismo. Este aspecto acaba sendo significativo no momento de decidir pela não adoção de um equipamento paralelo. Tentando traçar um perfil, é possível dizer que no binômio “fazer & comprar”, o software paralelo que uma empresa poderia necessitar, ainda está muito polarizado no fazer.

Do ponto de vista de uma empresa que desenvolve e comercializa software, a decisão de investir no mercado de processamento paralelo ou em outras frentes (por exemplo, melhoramentos em produtos já amplamente utilizados) é influenciada por alguns fatores (MORSE [280]):

- **pequena base instalada:** o mercado de equipamentos paralelos é pequeno, independente do referencial que for utilizado. Os equipamentos maiores estão nos laboratórios governamentais, os quais, via de regra, tem sua própria equipe de desenvolvimento. Outro grupo de equipamentos está em universidades, utilizados principalmente para pesquisa e ensino. Por sua vez, as empresas que fazem desenvolvimento tecnológico de seus produtos com o suporte de computadores paralelos (empresas químicas, automóveis, aviões), por questões de propriedade intelectual, também tem seu próprio grupo de programação.
- **elevado custo de conversão:** atualmente, para uma empresa migrar seu produto de software de uma arquitetura tradicional para uma plataforma paralela, terá de ter uma equipe de desenvolvimento conhecedora do hardware paralelo utilizado. Em função deste hardware, poderão ser necessárias modificações no *layout* de dados, no fluxo de controle, e até mesmo nos algoritmos básicos utilizados. O ganho de desempenho, principal razão de ser da adoção do hardware paralelo, poderá ser prejudicado com a não observância criteriosa destas modificações quase sempre indispensáveis.

- **validação:** testar o quão correto está o porte de um software para uma máquina paralela pode se mostrar uma tarefa bastante complexa, até mesmo porque os resultados das implementações seqüencial e paralela podem apresentar diferenças. Isto se potencializa para códigos numéricos, nos quais a convergência, a precisão e o erro acumulado, são fortemente influenciados pelo tamanho do problema. A decisão por uma arquitetura paralela, normalmente contempla problemas com dimensões bem maiores que aquelas possíveis de serem computadas em equipamentos com um só processador. Apesar dos matemáticos garantirem que o resultado de uma soma de números não depende da ordem de sua realização (propriedade associativa da soma), o hardware de ponto flutuante pode apenas se aproximar desta abstração. Considerações similares a esta fazem da validação do software paralelo uma atividade complexa e tratada com muita cautela pelos desenvolvedores de software, até mesmo porque incorreções na versão paralela podem lançar dúvidas sobre a qualidade da versão seqüencial já disponível.

5.1.9 Relação entre a Teoria e a Tecnologia

A teoria para o processamento paralelo foi desenvolvida após a tecnologia e ainda se encontra imatura em muitos aspectos. Deste modo, a teoria historicamente não vem sugerindo caminhos ou até mesmo limites para exploração tecnológica. Como resultado, ainda não estão disponíveis na abrangência necessária, representações abstratas da computação paralela, lógicas para avaliação da mesma, ou até mesmo algoritmos paralelos que sejam comprovadamente eficientes nas diversas arquiteturas reais (SKILLICORN [331]).

Parte III

Aspectos Técnicos

Capítulo 6

Conceitos Básicos

6.1 Arquiteturas Computacionais

A Arquitetura de computadores pode ser definida como a estrutura e a organização dos hardwares e se refere ao funcionamento interno de um computador, ou seja, a organização interna de todos os periféricos necessários para a montagem de um sistema computacional.

As arquiteturas serão caracterizadas a partir de componentes cuja nomenclatura é apresentada na figura 6.1. Estes também são os três principais blocos básicos das arquiteturas seqüenciais.



Figura 6.1: Blocos básicos dos computadores seqüenciais

6.1.1 A Classificação de Flynn para Arquiteturas de Computadores

A inclusão da proposta feita por Flynn (taxonomia de Flynn) em 1966 é, em primeiro lugar, um compromisso com a classificação mais difundida para arquiteturas de computadores.

A proposta se baseia nas combinações possíveis entre uma ou mais seqüências de instruções, atuando sobre uma ou mais seqüências de dados. Decorre daí, quatro classes de computadores:

- Seqüência de Instruções, uma Seqüência de Dados (**SISD**-*Single Instruction stream over a Single Data stream*): corresponde aos computadores seqüenciais convencionais, nos quais só existe uma única unidade de controle que decodifica seqüencialmente as instruções, que operam sobre um único conjunto de dados.
- Seqüência de Instruções, Múltiplas Seqüências de Dados (**SIMD**-*Single Instruction stream over a Multiple Data stream*): corresponde aos processadores matriciais. Nestas arquiteturas, diversos elementos processadores são ativados por somente uma unidade de controle. Esta unidade está submetida a um único programa, cujas instruções repassam aos elementos processadores. Os processadores executam, concorrentemente, uma mesma instrução sobre os dados que têm na sua memória local.
- Múltiplas Seqüências de Instruções, uma Seqüência de Dados (**MISD**-*Multiple Instruction stream over a Single Data stream*): não existem computadores construídos que se enquadrem nesta categoria.
- Múltiplas Seqüências de Instruções, Múltiplas Seqüências de Dados (**MIMD**-*Multiple Instruction stream over a Multiple Data stream*): nesta classe se enquadram os multiprocessadores.

Arquiteturas de Memória Distribuída

Neste tipo de arquitetura, cada nó tem seu processador, sua unidade de controle e sua memória local (MIMD). Assim, cada nó pode executar, de forma assín-

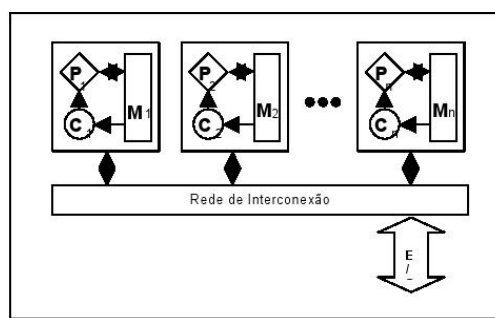


Figura 6.2: Arquitetura genérica de multiprocessador de memória

crona, um processo independente sobre seus próprios dados (vide figura 6.2). Os equipamentos que implementam este tipo de arquitetura também são conhecidos como multicomputadores ([128], [280]).

A rede de interconexão (vide item 6.6.2) é crítica para o desempenho deste tipo de equipamento. As diversas possibilidades de implementação da rede de interconexão (topologia, latência, contenção, etc.) neste tipo de arquitetura constituem um dos aspectos responsáveis pela falta de padrão no mercado de arquiteturas paralelas.

A configuração deste tipo de arquitetura é variável. O número de processadores, por exemplo, pode oscilar da casa das dezenas a alguns milhares. Em alguns casos, o crescimento ocorre em potências de 2 (16, 32, 64, 128, etc.) (vide item 6.6.3).

Principais aspectos positivos

Tecnologia de compilação: uma vez que cada nó da arquitetura é uma unidade de processamento autônoma, é possível aproveitar toda esta tecnologia de compilação disponível para programação seqüencial, agregando à mesma os recursos de uma biblioteca para troca de mensagens entre os nós processadores. São propostas usuais que, utilizando desta premissa, exploram conhecidas linguagens seqüenciais como C ou FORTRAN para programação paralela.

Possibilidade de emular outras arquiteturas: resguardadas as restrições inerentes ao desempenho, é possível à arquitetura de memória distribuída, emular outros paradigmas de controle e de organização de memória. Uma possibilidade usual é o emprego de DSM (*Distributed Shared Memory* [276], [304]), através da

qual o software aplicativo tem a visão de uma memória comum a todos os nós processadores.

Compartilhamento de uso: este tipo de arquitetura permite, de forma bastante flexível, o particionamento e a alocação de subgrupos de processadores à tarefas/usuários.

Principais aspectos negativos

Custo das comunicações: em função das características da rede de interconexão utilizada (vide item 6.6.2), alguns algoritmos podem ter seu desempenho diminuído. Assim, o processo de planejamento, codificação e geração de código (com a contribuição explícita ou não do programador), precisa considerar aspectos de localidade das comunicações e granulosidade das tarefas, para otimizar a possibilidade de seu particionamento e distribuição aos processadores.

Custo de sincronização: apesar de poderem trabalhar frequentemente de forma assíncrona, determinados momentos da execução paralela podem exigir um estado conhecido comum, para um grupo de processadores. Para minimizar o possível tempo de espera nos momentos de sincronização, o desenvolvimento de software deve contemplar uma distribuição de carga o mais equânime possível (o que nem sempre é viável), e com isso potencializar a utilização dos processadores e aumentar o desempenho global do processamento.

Uso ineficiente da memória: três aspectos concorrem para a sobre-ocupação da memória em arquiteturas de memória distribuída. O primeiro decorre da necessidade de armazenamento temporário das mensagens recebidas até que o processo responsável pela computação possa fazer o devido tratamento. Dependendo do tamanho e da frequência destas mensagens, um considerável volume de memória terá de ser destinado para isto. O segundo é consequência da necessidade de cópia local do código executável. O terceiro decorre, em função da busca de desempenho, de se fazer a cópia local também das estruturas de dados globais que o algoritmo possa necessitar.

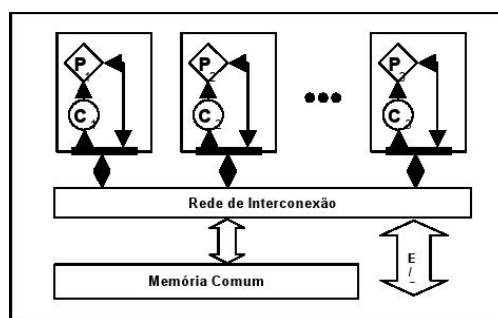


Figura 6.3: Arquitetura genérica de multiprocessador de memória compartilhada

Arquiteturas de Memória Compartilhada

Neste tipo de arquitetura todos os nós têm acesso uniforme a uma única memória comum (vide figura 6.3). São também denominadas de multiprocessadores simétricos ([128], [280]). Uma das razões do sucesso comercial deste tipo de arquitetura MIMD, é decorrente da sua flexibilidade de uso. Cada processador da arquitetura pode ser visto como uma máquina seqüencial tradicional; a existência de outros processadores, bem como da memória comum, pode ser abstraída. Uma consequência desta característica é a possibilidade de utilizar o software seqüencial já existente, praticamente sem nenhuma modificação. Deste modo, o paralelismo além de ser utilizado para melhorar o desempenho de um determinado programa, também pode ser empregado para executar simultaneamente diversos programas seqüenciais do usuário.

Como a memória é um recurso compartilhado, para que a mesma não se transforme em um ponto de estrangulamento da operação da arquitetura, o número de processadores varia, tipicamente, entre 4 e 20.

Uma estratégia para aumentar o desempenho do sistema de memória compartilhada é o uso de uma memória cache entre o processador e a memória comum. A busca de um sistema eficiente para manutenção da coerência de memória neste tipo de arquitetura é um tema complexo e originou diversos trabalhos de pesquisa.

A utilização destes sistemas trás vários aspectos positivos como:

Abstração da localidade do processador: neste tipo de arquitetura o programa-

dor pode abstrair a localidade do processador. Deste modo, a troca de mensagens é sincronizada por um mecanismo de escrita em variáveis comuns. Como a memória é fisicamente compartilhada, isto pode ser feito com elevado desempenho (via de regra maior que os obtidos com as políticas de DSM - *Distributed Shared Memory*).

Semelhante às arquiteturas convencionais: os multiprocessadores de memória compartilhada usualmente oferecem ambiente de programação e sistema operacional bastante semelhante aos das máquinas seqüenciais, o que facilita a adoção da arquitetura enquanto o software está sendo adequado para uma execução efetivamente paralela.

Facilidade de uso múltiplo: os processadores podem ser alocados individualmente ou em grupos, para diferentes programas/usuários.

Maior compartilhamento dos recursos: a memória comum facilita o compartilhamento de estruturas de dados globais. Por sua vez também, os recursos de entrada/saída e de memória virtual podem ser aproveitados por todos os nós processadores.

Mas também trás como problema da **pouca escalabilidade**, a política de acesso uniforme à memória faz com que este tipo de arquitetura tenha como limite um número de processadores ao redor de 20. O constante aumento do poder computacional dos processadores, e a conseqüente necessidade destes de maior banda-passante com a memória, contribui para potencializar este aspecto. Esta arquitetura também está sujeita ao custo de sincronização, que afeta as arquiteturas de memória distribuída (vide item 6.1.1). Porém, como o número típico de processadores não é grande, e as comunicações tem um desempenho elevado, assim como a sincronização como um todo pode ser melhor administrada.

Arquiteturas Síncronas Matriciais

Neste tipo de arquitetura, todos os processadores obedecem a uma única unidade de controle. Esta unidade busca e decodifica as instruções do programa e as transmite para os diversos processadores que as executam, utilizando sua própria memória local (SIMD). Assim, a cada ciclo, todos os processadores (menos os que

estão intencionalmente inibidos) executam sincronamente uma mesma instrução sobre sua parte dos dados. O paralelismo se dá, portanto, pela manipulação simultânea de diferentes partes do conjunto de dados. Daí sua denominação estar associada aos termos: arquiteturas síncronas e paralelismo de dados ([128], [280]).

Este tipo de arquitetura exige uma estrutura densa para a rede de interconexão, a fim desta suportar a difusão das instruções a partir do controlador para a matriz de processadores. Esta rede de interconexão também é utilizada para distribuir dados e recolher resultados.

O ambiente para geração de código neste tipo de arquitetura, usualmente, fica localizado em uma estação de trabalho que atua como intermediária (*front-end*) para a arquitetura. Esta estação acumula as funções de gerenciamento de contas de usuário, o escalonamento das diversas requisições de processamento e o acesso através da rede local de computadores.

As arquiteturas síncronas se mostram vocacionadas para algumas áreas de aplicação, nas quais oferecem uma excelente relação entre desempenho/custo. Destacam-se as áreas de processamento de sinais e imagens, nas quais a aglutinação de chips, cada um contendo dezenas de processadores simples e as respectivas memórias (de pequeno tamanho), podendo trazer excelentes resultados.

A Sincronização inerente entre processadores; a natureza do modelo de controle (único e centralizado) garante uma operação passo-a-passo, e os processadores estão conseqüentemente sempre sincronizados.

Diferentemente do que ocorre com as arquiteturas que têm controle distribuído (sejam de memória compartilhada ou não), estas ficam sujeitas as necessidades eventuais de sincronização, as quais costumam introduzir períodos de ociosidade na operação dos processadores.

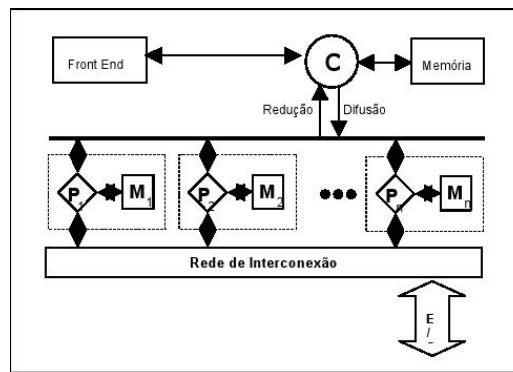


Figura 6.4: Arquitetura genérica síncrona matricial

Uso eficiente da memória: a única memória que precisa acomodar programas é a memória associada ao controlador central; as memórias dos processadores podem ser dedicadas totalmente para dados.

Alguns aspectos negativos desta abordagem são: **Escalabilidade**; quando o tamanho da matriz de processadores cresce, podem surgir dificuldades de garantir, através de uma rede de interconexão de grandes dimensões, a operação totalmente síncrona dos processadores (este aspecto se potencializa com o crescimento constante do *clock* dos processadores). A **Ineficiência ante desvios condicionais**; os desvios condicionais dependentes de dados, precisam ser processados independentemente, um após o outro. Esta situação é vista como um dos principais pontos de perda de desempenho desta arquitetura. E a **Dificuldade de compartilhamento**; uma vez que existe somente uma unidade de controle, a arquitetura somente pode ser utilizada por um programa/usuário de cada vez. Alguns fornecedores facultam a existência de mais de um controlador central (com o decorrente acréscimo de custo), o que permitiria o compartilhamento de recursos.

6.1.2 Multiprocessadores

A arquitetura de multiprocessadores é conhecida como fortemente acoplada, uma vez que os processadores e a memória estão fortemente interligados através de um sistema local de interconexão.

Essa arquitetura é caracterizada pelo compartilhamento global de memória pelos diversos processadores do ambiente e é esse compartilhamento global de memória que se torna o gargalo da escalabilidade do ambiente. A escalabilidade em

uma configuração multiprocessada varia até em algumas centenas de processadores.

6.1.3 Multicomputadores

A arquitetura de multicomputadores é conhecida como fracamente acoplada, uma vez que os processadores têm suas próprias memórias locais. Essa arquitetura é caracterizada por ter até milhares de processadores. Não há um compartilhamento forte, sendo as comunicações entre processos feitas por troca de mensagens entre os processos que estão sendo executados nos processadores.

Um exemplo de uma configuração de multicomputadores é a criação de um cluster com PCs convencionais usando uma rede local ethernet. Diferente da configuração de multiprocessadores em que é necessário à utilização de um comutador especial, esse tipo de cluster utiliza peças encontradas em qualquer loja de informática.

6.1.4 Multiprocessadores Simétricos (Symmetric Multiprocessors - SMP)

Estes ambientes são conhecidos como arquiteturas de compartilhamento total, são caracterizadas por até dezenas de processadores compartilhando os mesmos recursos computacionais e rodando um único sistema operacional. Os processadores são considerados simétricos porque tem os mesmos custos para acesso a memória principal.

A utilização de SMP é mais popular do que se imagina, a utilização deste tipo de máquina já é cotidiano em grande parte das organizações de hoje e também vem ganhando espaço em áreas menores, reflexo da alta redução de custos destes equipamentos.

Um problema desta arquitetura é sua escalabilidade, pois com o aumento do número de processadores a taxa de colisão de acesso à memória também cresce, sendo necessário a utilização de soluções de memórias de cache e globais, que

serão vistos à frente.

6.1.5 ccNUMA

Na arquitetura SMP não temos uma boa escalabilidade, pois, como se utiliza normalmente sistemas de interconexão na forma de barramento, que torna o gargalo do sistema rapidamente. Assim outras opções de interconexão podem ser utilizadas, como a utilização de interconexão comutada, que utiliza comutadores (switches) como elemento de ligação entre os processadores. Também existem outras soluções de interconexão que podem aumentar a largura de banda, mas é importante deixar claro que qualquer uma destas soluções agrega além de um custo altíssimo, retardo de comunicações entre os processadores e a memória.

Na teoria, uma arquitetura de Acesso Não-Uniforme à Memória (Non-Uniform Memory Access - NUMA) é conhecida por sua capacidade de escalar até centenas de processadores.

Máquinas NUMA preservam o modelo de programação simples de configurações SMP, mas com o acréscimo de tempo para acesso a memória global.

A implementação prática de uma máquina NUMA é conhecida como Acesso Não-Uniforme à Memória com Coerência de Cache (Cache Coherence Non-Uniform Memory Access - ccNUMA), pois estas já tratam vários problemas de acesso à memória desta arquitetura, como as diferenças de velocidades de acesso a memórias locais e globais, implementando sistemas de tratamento de coerência de cache.

Aplicações como banco de dados, ERP e CRM são aplicações candidatas a rodarem nessa plataforma.

6.1.6 Processadores Massivamente Paralelos (MPP)

Máquinas com configuração massivamente paralelas (Massive Parallel Processors - MPP), são arquiteturas fracamente acopladas. Computadores que seguem

este paradigma são usualmente classificados como multicomputadores, e usualmente um nó deste é um multiprocessador.

Essa arquitetura é caracterizada por milhares de nós computacionais interligados por uma rede de interconexão de alta velocidade. Cada nó pode ser composto por um ou mais processadores, possuindo cache e memórias locais. Cada nó possui também seu próprio sistema operacional, onde as aplicações rodam localmente e se comunicam por sistemas de trocas de mensagens (11.1).

A escalabilidade de um MPP é maior do que arquiteturas de memória compartilhada. Os maiores computadores classificados na lista TOP500 [362] usam este paradigma.

Uma parte importante na configuração MPP é o sistema de interconexão que liga seus vários nós. Entre os principais fatores em consideração na construção destes sistemas de interconexão são, segundo DANTAS [136]:

- Topologia;
- Algoritmo de roteamento;
- Estratégia de comutação;
- Controle do fluxo entre nós.

A escalabilidade de um MPP é maior do que de arquiteturas de memória compartilhada. Os maiores computadores classificados na lista TOP500 [362] usam este paradigma.

6.1.7 Sistemas Distribuídos

Os sistemas distribuídos, sob o aspecto da arquitetura de máquinas, para a execução de aplicativos, podem ser vistos como configurações com grande poder de escalabilidade, pela agregação dos computadores existentes nas redes convencionais em um sistema único, onde a homogeneidade ou heterogeneidade do conjunto de máquinas, cada uma com seu próprio sistema operacional, permite

a formação de interessantes configurações de SMPs, clusters, MPPs e grids computacionais. ([136])

Vários aspectos na utilização de ambientes distribuídos têm de ser cuidados, aspectos como segurança, confiabilidade, retardo da rede de comunicações, compatibilidades de pacotes de softwares, entre outros.

6.1.8 Clusters

As configurações de clusters em termos de arquitetura computacional, podem ser entendidas como uma agregação de computadores de forma dedicada para a execução de aplicações específicas. Normalmente formados por computadores do tipo PC, pertencentes a uma única unidade (ex: laboratório).

A escalabilidade é um fator diferencial nestes ambientes, pois os recursos podem crescer conforme estiverem disponíveis. ([136])

6.1.9 Grids

Grids computacionais são uma nova forma de agregar ambientes geograficamente dispersos, com objetivos claros de especificação de qualidade de serviços. Atualmente, a Internet com uma configuração distribuída de recursos é conhecida como o ambiente que melhor pode demonstrar esse tipo de ambiente. Em outras palavras, diferentes tipos de aplicativos com diferentes tipos de requerimentos de qualidade (exemplos são a largura de banda, retardo de comunicações e *jitter*¹) são tratados de maneira igual. Em adição, os “serviços WEB” que oferecem serviços para a execução de tarefas de usuários finais ainda são poucos. Desta forma, o objetivo destas configurações é voltar toda a potencialidade de recursos e serviços disponíveis para o processamento de tarefas dos usuários pertencentes à configuração de grid (DANTAS [136]). Grids computacionais são amplamente discutidos no capítulo 13 deste trabalho.

¹Jitter é uma variação estatística do retardo na entrega de dados em uma rede, ou seja, pode ser definida como a medida de variação do atraso entre os pacotes sucessivos de dados

6.2 Dependabilidade

Dependabilidade é um termo traduzido literalmente do inglês *dependability* que reúne diversos conceitos que servem de medida tais como confiabilidade (*reliability*), disponibilidade (*availability*), segurança (*safety*), manutenibilidade (*maintainability*), comprometimento do desempenho (*performability*), e testabilidade (*testability*). Estas são medidas usadas para quantificar a dependabilidade de um sistema. ([302])

Assim pode-se dizer que a dependabilidade é uma propriedade dos sistemas computacionais que define a capacidade dos mesmos de prestar um serviço no qual se pode justificadamente confiar (DANTAS [136]).

Confiabilidade é a medida mais usada em sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis. Confiabilidade é uma medida probabilística que não pode ser confundida com disponibilidade. Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço. *Performability* está relacionada à queda de desempenho provocado por falhas, onde o sistema continua a operar, mas degradado em desempenho. *Maintenability* significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo e a colocação em operação. Finalmente, testabilidade é a capacidade de testar atributos internos ao sistema ou facilidade de realizar certos testes. Quanto maior a testabilidade, melhor a manutenibilidade, por consequência, menor o tempo de indisponibilidade do sistema devido a reparos.

A caracterização de dependabilidade envolve ainda um conjunto de conceitos que alguns autores dividem em três grupos: os atributos, os meios pelos quais será alcançada e as ameaças. Nas próximas sessões estes três grupos serão melhores vistos.

6.2.1 Ameaças

Um defeito é definido como um desvio da especificação, ou a transição de estado do serviço de um sistema de correto para um serviço incorreto. Deve ser evitado que o sistema apresente defeitos, pois estes não podem ser tolerados.

Define-se falha (ou falta) como a causa física ou algorítmica do erro. Falhas estão associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário. Assim um chip de memória, que apresenta uma falha em um de seus bits (falha no universo físico), pode provocar uma interpretação errada da informação armazenada em uma estrutura de dados (erro no universo da informação) e como, resultado o sistema pode negar autorização de embarque para todos os passageiros de um voo (defeito no universo do usuário).

$$\Rightarrow FALHA \Rightarrow ERRO \Rightarrow DEFEITO \Rightarrow$$

O entendimento da relação de dependência entre falhas, erros e defeitos é a base para o conhecimento da "patologia da falha". Essa relação, como mostrado acima, pode ser utilizada em outros componentes, não apenas físicos, e a sua utilização recursiva ajuda na análise de sistemas em diferentes níveis de abstração. Informações mais detalhadas sobre este tópico podem ser obtidas em DANTAS [136].

6.2.2 Meios

Os meios da classificação de dependabilidade nos ajudam a trabalhar na prevenção, tolerância, remoção ou previsão das falhas. E tem como objetivo tratar as falhas que podem levar a erros, e que em sua propagação causam defeitos.

Prevenção De Falhas

A prevenção de falhas tem por objetivo aumentar a confiabilidade dos sistemas, empregando técnicas de controle de qualidade em projetos e desenvolvimento. Falhas não são facilmente previsíveis, então é preciso existir procedimentos para

que, caso ocorram, existam formas de reparo a fim de restaurar as condições de serviços. Um exemplo de falha imprevisível é a falha de um componente de hardware.

Tolerância à Falhas

O paradigma de tolerância à falhas é definido como a capacidade de um sistema apresentar um comportamento bem definido na ocorrência de falhas. As formas básicas de tolerância à falhas são:

Propriedade do Sistema	Operacionalidade Garantida
Operacionalidade não garantida	Segurança garantida
Mascaramento	Defeito seguro (fail safe)
Segurança não garantida	Sem mascaramento, Não tolerância

Tabela 6.1: Formas básicas de tolerância à falhas. Fonte DANTAS [136]

A primeira forma se caracteriza pela segurança e operacionalidade garantida, é a que realiza o mascaramento, empregado para encobrir ou ocultar falhas. Neste item o serviço apresentado pelo sistema não deverá ser modificado pela ocorrência de falhas, ou seja, o sistema como um todo não deverá apresentar defeito. Logo o sistema deverá permanecer operacional e em um estado seguro para os usuários e para o meio ambiente. Está é a forma mais completa de tolerância à falhas, a mais desejada e a de maior custo. Todas as demais formas modificam o serviço prestado pelo sistema na ocorrência de falhas ([136]).

A tolerância a falhas consiste, basicamente, em ter hardware redundante que entra em funcionamento automaticamente após a detecção de falha do hardware principal.

Este texto não tem a intenção de estender demasiadamente a discussão sobre este tema podendo ser melhor visto em DANTAS [136].

Remoção de Falhas

Uma solução para a obtenção da dependabilidade é a opção conhecida como remoção de falhas. Esta técnica pode ser aplicada tanto na fase de desenvolvimento como durante o ciclo de vida do sistema. A remoção de falhas na fase de desenvolvimento é realizada através das etapas de verificação, diagnóstico e correção. A verificação dos mecanismos de tolerância à falhas é um importante aspecto de remoção de falhas ([136]).

Previsão De Falhas

A previsão de falhas é o último meio utilizado para se alcançar a dependabilidade. Esta opção considera uma avaliação do comportamento do sistema com relação à ocorrência e ativação de falhas. Esta abordagem pró-ativa pode subsidiar as modificações para melhorias, tanto estruturais como para melhor eficiência/eficácia dos sistemas.

6.2.3 Atributos

Os atributos de dependabilidade têm naturezas não determinísticas das circunstâncias dos atributos que são: Disponibilidade, Confiança, Segurança, Confiabilidade, Integridade, Reparabilidade. Esses atributos usam medidas probabilísticas para gerar seus pesos relativos. Esses pesos são medidos dependendo da aplicação/serviço considerado, assim estes pesos variam sempre, não existindo um padrão ([136]).

- Disponibilidade:

Disponibilidade instantânea é o atributo, definido como a probabilidade de um sistema apresentar um serviço correto, num determinado instante de tempo t . Na análise de disponibilidade estamos interessados no comportamento de um sistema ao de determinados períodos de tempo, ou seja, estamos preocupados em observar a alternância de períodos de funcionamento correto e períodos que o sistema está de reparo. O fator importante é saber

a fração de tempo na qual o sistema deverá ter condições de apresentar o serviço de forma correta.

- **Confiabilidade:**

É a métrica que avalia, o quanto um sistema pode apresentar um serviço correto continuamente durante um intervalo de tempo t , ou seja, a probabilidade de o sistema não apresentar defeito durante o intervalo de tempo considerado.

- **Segurança:**

É considerada sob dois aspectos: contra catástrofes e convencional. Contra catástrofes é a probabilidade do sistema apresentar defeito que acarrete consequências catastróficas para seus usuários em um intervalo de tempo. E segurança convencional é a probabilidade obtida através da combinação dos atributos: disponibilidade, confidencialidade e integridade, ou seja, a probabilidade de que não ocorra acesso ou manipulação indevida no estado do sistema no intervalo de tempo.

- **Confidenciabilidade:**

É a probabilidade de não ocorrer divulgação indevida de informação no intervalo de tempo.

- **Integridade:**

É a probabilidade de não ocorrer alterações impróprias de estado em um sistema no intervalo de tempo.

- **Reparabilidade:**

Esta métrica avalia o quanto um sistema pode ser restaurado, retornando ao estado de serviço correto em determinado tempo, dado que o mesmo apresentou defeito.

6.3 Escalonamento

Escalonamento é um processo de tomada de decisões que se preocupa com a alocação de recursos limitados para tarefas ao longo do tempo, e possui como meta a otimização de uma ou mais funções-objetivo. As tarefas podem ser operações em um processo de produção, execução de software em um sistema de computação,

entre outros. As funções-objetivo também podem ser diversas, como a minimização do tempo médio gasto por uma atividade de montagem de peças em uma máquina de uma linha de produção ou a minimização do tempo de execução de uma tarefa computacional.

Escalaonadores de tarefas são componentes de software comumente integrados a sistemas operacionais paralelos e/ou distribuídos e que têm como função a distribuição de trabalho computacional para as unidades de processamento integrantes do sistema, de modo a maximizar o desempenho global do processamento realizado, isto é, promover o balanceamento de carga entre as unidades de processamento envolvidas.

Em sistemas homogêneos, o problema de balanceamento de carga pode ser reduzido a uma divisão de um determinado trabalho computacional em N porções iguais e que possam ser distribuídas e executadas por N unidades de processamento do sistema, supostamente idênticas. Neste caso, o problema está fortemente relacionado à maneira de como representar o trabalho computacional a ser processado e a melhor maneira de dividi-lo em várias partes iguais.

Em sistemas heterogêneos, o problema de balanceamento de carga é consideravelmente mais complexo e, nestas circunstâncias, o escalonador de tarefas ganha especial importância. Para que o conjunto heterogêneo de unidades de processamento possa ser utilizado de maneira eficiente, questões como predição e monitoramento de desempenho, passam a integrar o problema de balanceamento de carga.

Isso significa que um bom compromisso, entre o tempo de processamento despendido na busca por uma solução e a qualidade da solução encontrada, deve ser satisfeito, e é no contexto deste compromisso que as principais linhas de desenvolvimento de escalonadores ganham forma: a dos escalonadores estáticos e a dos dinâmicos.

Um importante aspecto dos escalonamentos estáticos é que seu cálculo se faz de maneira totalmente independente da distribuição das tarefas. O escalonamento é feito em duas etapas: na primeira etapa o cálculo do escalonamento é realizado, ou seja, a atribuição das tarefas às unidades de processamento é definida; no segundo momento, um mecanismo de distribuição de tarefas deve entrar em ação para promover a distribuição previamente calculada.

Uma importante consequência deste modelo de escalonamento é a necessidade de se ter informações precisas sobre o sistema considerado. Assim, o bom funcionamento de um escalonamento de tarefas estático requer uma estimativa precisa do desempenho do sistema em questão e a qualidade deste escalonamento é um resultado direto da precisão com que estas estimativas são obtidas. Nestas circunstâncias, estimativas imperfeitas ou ocorrências de eventos inesperados, que afetem o desempenho do sistema durante a execução das tarefas previamente escalonadas podem fazer com que seu desempenho global sofra significativos decréscimos.

Apesar desta aparente limitação, o escalonamento estático é largamente utilizado em sistemas paralelos reais, uma vez que sua simplicidade de implementação lhe confere grande robustez e facilidade de manutenção. Além disso, nestes sistemas a ocorrência de eventos que afetem significativamente o desempenho do escalonamento é rara e os resultados são freqüentemente satisfatórios.

Em oposição a esta técnica está a dos escalonadores dinâmicos. O escalonamento dinâmico pode ser entendido como a aplicação de sucessivos escalonamentos estáticos sobre estados intermediários de execução da aplicação, à medida que ela é executada. Os momentos em que cada um desses escalonamentos é realizado varia de escalonador para escalonador, mas o aspecto mais importante dos escalonadores dinâmicos, é o que justifica o emprego do termo "dinâmico", e o fato de o escalonamento ser feito concorrentemente à distribuição e execução das tarefas das aplicações.

Ao produzir-se um escalonamento com essas características, beneficia-se da habilidade em lidar com grande parte das decisões de escalonamento em tempo real, o que eliminam muitos dos problemas do caso estático. Embora as decisões ainda se baseiem em estimativas de desempenho do sistema e, conseqüentemente, estimativas imprecisas ainda podem significar um escalonamento ineficiente. Com isso, as consequências de um mau escalonamento não são tão impactantes quanto seriam no caso estático. Assim, atrasos ou adiantamentos no tempo de conclusão de uma determinada tarefa podem ser utilizados em tempo real para o reescalonamento das tarefas restantes a serem executadas. Uma vantagem adicional do fato de seu processamento ser realizado concorrentemente a execução da aplicação escalonada e que isso pode significar economia de tempo global com relação ao caso estático.

Entretanto, os escalonadores dinâmicos possuem seus inconvenientes. Em contrapartida aos escalonadores estáticos, a implementação dos escalonadores dinâmicos é trabalhosa e requer a manipulação e gerência de estruturas de dados freqüentemente complexas. Esse fato torna este tipo de escalonador pesado, sob o ponto de vista da implementação e execução, e menos robusto já que, na eventual ocorrência de uma falha, um grande trabalho de recuperação de estado deverá ser feito.

Outro importante aspecto no projeto de escalonadores de tarefas é o paradigma de operação adotado. A existência de diferentes paradigmas advém do fato da implementação do escalonador de tarefas estar diretamente vinculado às maneiras de como as unidades de processamento do sistema distribuído em questão estejam conectadas entre si, tanto física quanto logicamente. Assim, existe um grande número de paradigmas de balanceamento de carga, emergidos de diferentes topologias de interconexão, cada qual adaptado às características do ambiente computacional no qual foi concebido.

6.4 Alta Disponibilidade

Um sistema computacional é composto por diversos componentes eletrônicos que podem falhar impedindo o acesso a informação. A crescente demanda por sistemas que possam deixar informação disponível para ser acessada, modificada, armazenada pelo maior tempo possível levou fabricantes de hardware e desenvolvedores de software a pensarem em maneiras de como contornar esses problemas de paradas de sistemas, sejam elas causadas por falhas internas (causadas por mal funcionamento de hardware, erros introduzidos por softwares ou outras razões de natureza imprevisível, como interferência magnética) ou mesmo paradas programadas para manutenção.

O conceito de alta disponibilidade é caracterizado por um sistema desenhado para impedir perda de um serviço por ele disponibilizado, reduzindo ou gerenciando falhas (mais detalhes em [6.2.2](#)) bem como minimizando tempo de desligamento planejado para manutenção.

Este conceito não se resume a um software específico, mas a um conjunto de

Disponibilidade (%)	Downtime/ano	Downtime/mês
95	18 dias 6:00:00	1 dias 12:00:00
96	14 dias 14:24:00	1 dias 4:48:00
97	10 dias 22:48:00	0 dias 21:36:00
98	7 dias 7:12:00	0 dias 14:24:00
99	3 dias 15:36:00	0 dias 7:12:00
99,9	0 dias 8:45:35.99	0 dias 0:43:11.99
99,99	0 dias 0:52:33.60	0 dias 0:04:19.20
99,999	0 dias 0:05:15.36	0 dias 0:00:25.92

Tabela 6.2: Níveis de Alta Disponibilidade

mecanismos e técnicas que tem por objetivo detectar, contornar e mascarar falhas que venham a ocorrer ocasionando perda de acessibilidade.

É senso comum na literatura caracterizar a disponibilidade pela probabilidade de um sistema estar acessível em determinado período de tempo.

A Tabela 6.4 ilustra um dos termos de comparação geralmente utilizado na avaliação de soluções HA: níveis de disponibilidade segundo tempos de indisponibilidade (downtime). Excluídos desta tabela, os tempos de downtime estimados (geralmente para manutenção ou reconfiguração dos sistemas) que são alheios às soluções e muito variáveis.

Quanto maior a disponibilidade desejada ao sistema, maior a redundância e custo das soluções, tudo depende do tipo de serviço que se pretende disponibilizar e de outras variáveis intrínsecas ao sistema. Há casos em que o custo do sistema indisponível é muito maior que o custo de desenvolvimento de um ambiente de alta disponibilidade para o mesmo. Informações mais detalhadas sobre este assunto podem ser obtidas na sessão 6.2 deste documento, em DANTAS [136] e softwares que trabalham a disponibilidade de sistemas serão discutidos no capítulo 8 também neste documento.

6.5 Balanceamento de Carga

Quando se projeta um sistema computacional, sabe-se a carga média e máxima que este irá suportar. Apartir do momento em que a carga de utilização do sis-

tema começa a se tornar excessiva, é preciso se buscar uma solução para o aumento de capacidade do sistema, que pode ser basicamente: i) aquisição de uma máquina de maior capacidade computacional, ii) melhoria de performance do sistema e iii) utilização de sistemas de balanceamento de carga.

Os sistemas de balanceamento de carga são em geral a repartição da execução do serviço por várias máquinas. Estas soluções podem se especializar em pequenos grupos sobre os quais se faz um balanceamento de carga: utilização da CPU, de armazenamento, ou de rede. Qualquer uma delas introduz o conceito de clustering, ou server farm, já que o balanceamento será, provavelmente, feito para vários servidores.

Informações sobre a implementação de algumas soluções de balanceamento de carga em Software Livre serão discutidos no capítulo 8 deste documento.

6.6 Rede de Comunicações

6.6.1 A Importância da Rede de Comunicação

Em cluster a eficiência do sistema da rede de comunicação entre os nós é de extrema importância e criticidade. Se a rede falha ou tem baixo desempenho, o cluster inteiro sentirá esse problema e, por consequência, o desempenho do sistema como um todo será atingido.

Assim é comum se projetar redes para cluster pensando não apenas no desempenho e latência desta, mas também na alta-disponibilidade da rede.

É importante considerar que uma rede é um elemento bastante seguro a nível físico: dificilmente uma vez instalada, a rede fisicamente irá falhar.

Outro tópico importante da rede é a sua eficiência: uma rede congestionada destrói o desempenho do cluster. Assim, dependendo do tamanho do cluster, e da quantidade de nós pertencentes a este, a rede poderá ser a culpada diretamente pela baixa eficiência computacional do cluster. É por isto que o investimento em uma rede tecnologicamente moderna é habitual nestes tipos de sistemas.

6.6.2 Redes de Interconexão Utilizadas em Arquiteturas Paralelas

Não importando o tipo da arquitetura, todo computador paralelo necessita de uma rede de interconexão para criar canais de comunicação entre os seus diversos recursos de processamento, armazenamento e entrada/saída. Considerando a diversidade das alternativas tecnológicas, esta seção vai explorar aspectos centrais pertinentes ao tema, a partir dos quais, podem ser entendidas as várias alternativas possíveis para as redes de interconexão.

Alternativas para Interligar o Processador à Rede de Interconexão

Do ponto de vista da organização do hardware, existem duas possibilidades para o posicionamento das chaves de interconexão (vide figura 6.5) apresentada por MORSE ([280]):

- **chave associada ao processador:** neste caso, a maioria das vezes a chave está localizada no mesmo circuito integrado (*chip*) do processador. Nesta implementação é possível para o processador enviar e/ou receber múltiplas mensagens concorrentes, o que em determinadas situações pode ser oportuno para exploração do paralelismo. Como exemplo, temos o emprego desta tecnologia nas arquiteturas SIMD (vide item 6.1.1) CM-1, CM-2 e AMT DAP, e também nas arquiteturas MIMD (vide item 6.1.1) nCube, Transputer, iWarp e KS-1.
- **chave independente do processador:** nesta implementação, o processador tem um único canal com a sua chave de interconexão. A principal vantagem deste caso é a maior flexibilidade para criação de nós heterogêneos na arquitetura. Os nós responsáveis pela entrada/saída poderiam utilizar a mesma chave de interconexão que os nós processadores. Embora não seja uma prática comum, esta segunda estratégia também faculta que possam ser trocados os processadores e mantida a rede de interconexão. As arquiteturas SIMD não utilizam esta segunda opção de chave. Alguns exemplos de arquiteturas MIMD que a empregam seriam o Intel Paragon, a CM-5 e o Cray T-3D. Uma desvantagem decorrente da dissociação entre o proces-

sador e a chave de interconexão é o prejuízo do nível de integração (mais circuitos integrados, mais conexões, etc.).

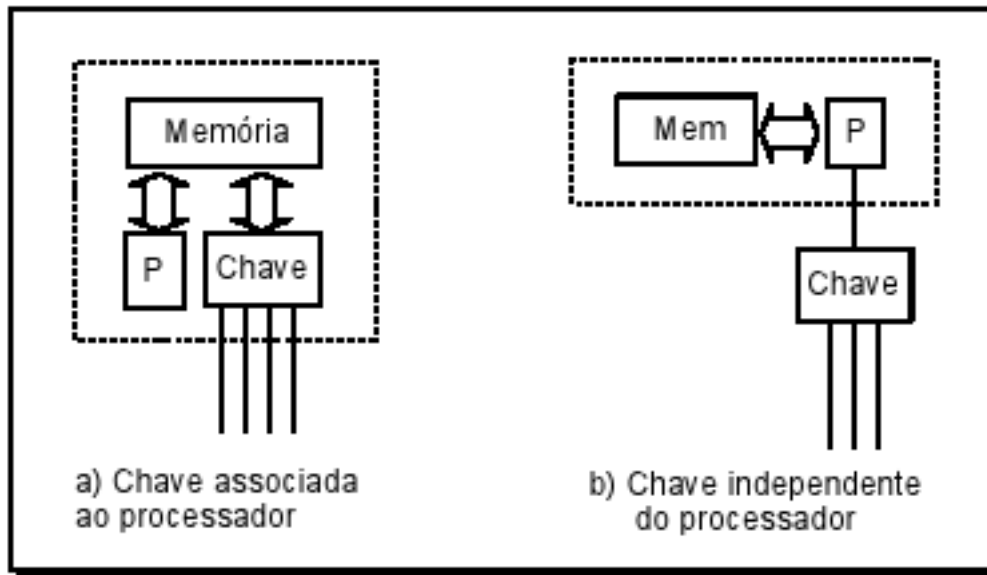


Figura 6.5: Alternativas para conectar o processador a rede de interconexão

Características que Definem o Desempenho de uma Rede de Interconexão

Além da topologia da rede de interconexão, as outras características que se destacam na definição do seu desempenho são:

- **largura de banda do canal:** número de bytes por segundo que pode fluir entre dois nós com conexão direta. Via de regra, a largura de banda é dependente do número de pulsos por segundo da arquitetura (*clock*) e do número de bits possíveis de serem enviados por pulso.
- **latência de comutação:** tempo inerente à operação da chave de comutação. Se dois processadores precisam trocar dados, e não existe um canal interligando os dois diretamente, as chaves de comutação intermediárias precisam propagar a mensagem através da rede de interconexão. As latências elevadas trazem prejuízo ao desempenho da arquitetura paralela, sobretudo quando a mensagem necessita passar por diversas chaves.

- **independência de processador:** caracteriza se o processador precisa ou não ser interrompido, para auxiliar na atividade de comunicação. Muitas das atuais implementações de redes de interconexão permitem que o processador continue sua computação enquanto uma mensagem está sendo transmitida, recebida ou roteada. Isto minimiza o custo introduzido pela necessidade de comunicação entre processadores.
- **contenção:** pode ocorrer a recepção praticamente simultânea de duas mensagens por uma determinada chave, e ambas podem necessitar usar o mesmo canal de saída. Uma obrigatoriamente terá de aguardar. O atraso na computação do processador que aguarda a mensagem retida pode resultar em perda de desempenho. Uma possibilidade é o hardware de comutação prever uma política de tempo compartilhado para as portas das chaves; isto dividiria o custo de espera entre os dois processadores destinatários, porém introduziria custos de comutação (vide latência de comutação).

6.6.3 Topologias da Rede de Interconexão

Uma vez que a interconexão direta de todos os processadores entre si não é viável quando o número dos mesmos aumenta, como regra geral é utilizado um padrão para definir as ligações. Este padrão é denominado de topologia da rede de interconexões. Três parâmetros podem ser utilizados para caracterizar o possível desempenho de uma topologia. Os mesmos são: a largura da bisseção, o diâmetro e o grau (BAKER [71]).

A **largura da bisseção** indica quantas mensagens simultâneas podem ser trocadas entre duas metades da rede de interconexão. É um indicador da largura de banda possível para as comunicações através da rede. O **diâmetro** indica qual o menor número de nós intermediários que precisam ser envolvidos, para que dois processadores, o mais distantes possível, se comuniquem.

O **grau** indica o número máximo de mensagens que podem ser manipuladas (enviadas ou recebidas) simultaneamente por cada um dos processadores.

Topologia em Barramento

Nesta topologia, todos os processadores estão conectados em um único barramento compartilhado. Quando um processador necessita comunicar-se com outro, ele aguarda que o barramento esteja livre e propaga no mesmo a mensagem; o destinatário, por sua vez, identifica que a mensagem é para si e a recebe (vide figura 6.6).

No caso de duas transmissões simultâneas, o software detector de colisões interrompe as transmissões e os processadores voltam a tentar novamente após um período de tempo determinado aleatoriamente.

Assim sendo, a sua largura da bisseção é 1. Isto significa que esta topologia não permite mais do que um par de processadores em comunicação simultaneamente.

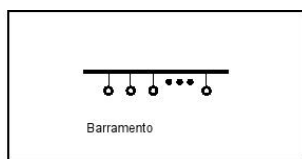


Figura 6.6: Topologia em barramento

Do ponto de vista do desempenho, esta topologia somente é viável para um pequeno número de processadores e/ou classes de problemas cujos algoritmos implementem pouca comunicação. Esta topologia é bastante usual em pequenos agrupamentos (*clusters*) de estações de trabalho interligadas por redes locais.

Topologia em Malha

Os processadores nesta topologia tem um canal de comunicação direto com o seu vizinho (a). Uma variação que é utilizada consiste em interligar as extremidades da grade, criando uma configuração denominada malha toroidal (b), a qual reduz o diâmetro da malha por um fator de 2 (vide figura 6.7).

A largura da bisseção de uma malha é $\sqrt{N}$ onde N é o número de processadores.

A largura da bisseção dobra para a malha toroidal. O diâmetro da topologia em malha é $2(\sqrt{N} - 1)$, e o seu grau é fixo e de valor 4.

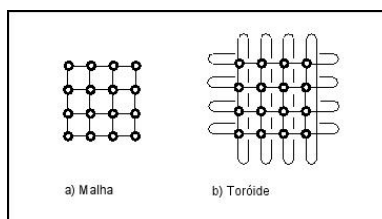


Figura 6.7: Topologia em malha

O hardware para este tipo de tecnologia é de simples construção e expansão. A malha se adapta bem a algoritmos utilizados em cálculos científicos, onde se destaca a manipulação de matrizes.

Uma arquitetura que utiliza esta topologia é o Intel Paragon.

Topologia em Hipercubo

Toda rede de interconexão hipercúbica está alicerçada sobre uma estrutura multi-dimensional baseada em endereços binários.

Os tamanhos do hipercubo são definidos por potências de 2; $N=2^D$ onde D é a dimensão do hipercubo e N o número de processadores.

Em função disto, todos os nós podem ser identificados por um número binário. Cada nó é conectado a todos os seus vizinhos; isto faz com que o hipercubo tenha grau variável e de valor D (vide figura 6.8).

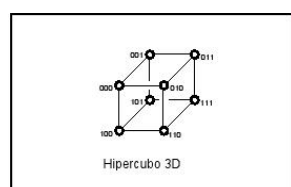


Figura 6.8: Topologia em hipercubo

A topologia hipercúbica confere boas propriedades à rede de interconexão; a largura da bisseção é $N/2$, e o diâmetro é $\log_2 N$. Apesar de apresentar bom desempenho para muitos padrões de comunicação, sua eficiência se mostra bastante dependente do algoritmo de roteamento a ser empregado.

Um aspecto inconveniente do hipercubo é sua escalabilidade, o número de processadores sempre cresce em potência de 2. Além disso, como o grau de cada nó é em função do tamanho do cubo, toda expansão no número de processadores implica em adicionar mais um canal de comunicação a todos os nós. Para cubos maiores, estas características podem trazer inconvenientes para a administração do custo/benefício quando da expansão da arquitetura. Um equipamento que emprega esta topologia é o Ncube 2.

Topologia em Árvore

A clássica árvore binária, com processadores nas suas folhas, tem se mostrado uma boa opção de topologia para arquiteturas paralelas. O diâmetro de uma árvore completa é $2\log_2((N+1)/2)$, bastante similar ao do hipercubo (N é o número de processadores). A largura da bisseção, por sua vez, é somente 1, o que pode introduzir um severo gargalo quando processadores de uma metade da árvore precisarem se comunicar com os da outra metade.

A solução para pequeníssima largura da bisseção da árvore é utilizar uma variante denominada árvore larga. Em uma árvore larga (vide figura 6.9), a largura dos ramos (canal) cresce a medida em que se sobe das folhas em direção à raiz.

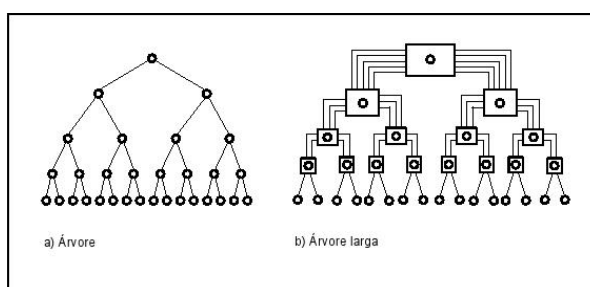


Figura 6.9: Topologia em árvore

A largura da bisseção da árvore larga plena é N e o seu diâmetro proporcional a

$2(\log N)$. A arquitetura da CM-5 da *Thinking Machines* utiliza uma versão modificada da árvore larga.

6.6.4 Dispositivos de interconexão

Já estão disponíveis comercialmente dispositivos de interconexão que propiciam a criação de ambientes similares a multicomputadores ou multiprocessadores, utilizando computadores convencionais.

Existem atualmente duas grandes classes de dispositivos de interconexão para alto desempenho. Uma primeira classe é formada por dispositivos cuja solução é baseada em programação por troca de mensagens entre processadores no nível de placa de rede, esta solução permite a criação de multicomputadores. Exemplos de equipamentos desta classe são: Myrinet, Gigabyte System Network e Giganet, sistemas que utilizam rede Gigabit ethernet também são encontrados, mas com desempenho de rede mais baixo. Não se pode confundir as tecnologias Gigabit ethernet, Gigabyte System Network e Giganet. A Gigabit ethernet é a mais conhecida, utilizada e de menor custo, todavia o seu desempenho é muito menor comparado com as outras soluções.

A segunda classe é formada por interconexões e tem como peculiaridade uma solução que cria a abstração de uma memória virtual única (multiprocessador) entre todos os computadores interligados no dispositivo. Exemplo desta são o Quadrics Network (QSNET) e Dolphin SCI.

Gigabit Ethernet

O padrão Gigabit Ethernet é uma extensão dos padrões 10 Mbps Ethernet e 100 Mbps Fast Ethernet para interconexão em redes. Esse padrão surgiu da necessidade criada pelo aumento da largura de banda nas "pontas" das redes (ex.: servidores e estações de trabalho) e também pela redução constante dos custos entre as tecnologias compartilhadas e comutadas, juntamente com as demandas das aplicações atuais.

O padrão Gigabit Ethernet tem como principais vantagens a popularidade da tec-

nologia Ethernet e o seu baixo custo. Trata-se de uma tecnologia padrão, protegendo o investimento feito em recursos humanos e em equipamentos. Não há nenhuma nova camada de protocolo para ser estudada, conseqüentemente, há uma pequena curva de tempo de aprendizagem em relação à atualização dos profissionais. Graças às vantagens trazidas por essa tecnologia, ela tornou-se também outra possibilidade para a interconexão em clusters.

Apesar da alta velocidade, o padrão Gigabit Ethernet não garante o fornecimento de QoS (Qualidade de Serviço), que é um dos pontos mais fortes da tecnologia ATM. Desta forma, ele não pode garantir o cumprimento das exigências de aplicações, como a videoconferência com grande número de participantes, ou mesmo uma transmissão de vídeo em tempo-real de um ponto para muitos outros. O fato do custo por porta da Gigabit Ethernet ser mais alto, quando comparado com a Myrinet, e devido ao uso de cabos de fibra-ótica. Contudo, uma queda substancial no custo da Gigabit Ethernet agora é aparente. Os custos baixos estão impulsionando a Gigabit Ethernet para tomar uma fatia cada vez maior de mercado, caracterizado pela competição entre diversos fabricantes e um potencial muito grande de base de instalações; eventualmente, o custo por porta da Gigabit Ethernet se tornará comparável ao custo de um nó convencional, de forma muito parecida com o que ocorreu com a Fast Ethernet há alguns anos atrás.

Myrinet

Myrinet é um tipo de rede baseada na tecnologia usada para comunicação e troca de pacotes entre processadores trabalhando em paralelo. Myrinet implementa auto-inicialização, baixa latência e switches "cut-through". As interfaces de host mapeiam redes, selecionam rotas, controlam o tráfico de pacotes e transformam endereços de rede em rotas. Seu software otimizado permite que seja feita uma comunicação direta entre os processos do usuário e a rede. Uma diferença em relação às LAN's está nas altíssimas taxas de transmissão e nas baixíssimas taxas de erro, além de controle de fluxo em todos os links.

Um link Myrinet é um par full-duplex de canais Myrinet opostos. Um canal Myrinet é unidirecional e ele é o meio de comunicação que carrega caracteres de informações. O fluxo do remetente pode ser bloqueado, temporariamente, pelo receptor a qualquer hora, durante ou entre pacotes, usando controle de fluxo.

Num ambiente de comunicação confiável pode-se empregar roteamento “cut-through”, no qual não existe a bufferização do pacote inteiro com checagem de erro no “checksum”.

No roteamento “store-and-forward”, se o canal de saída está ocupado ele fica enfileirado num circuito de roteamento ou nó, que supostamente tem memória suficiente para bufferizar o pacote. Já no roteamento “cut-through” os circuitos de roteamento podem bloquear com controle de fluxo se o canal de saída não estiver disponível. Desta forma o circuito de roteamento “cut-through” não requer bufferização, pois cada link pode prover seu próprio controle de fluxo. Para prover o controle de fluxo, as redes MPP reconhecem cada unidade de fluxo de controle, que é tipicamente um byte.

InfiniBand

InfiniBand é uma arquitetura que define um barramento de computador serial de alta velocidade, projetado tanto para conexões internas quanto externas. Ele é o resultado da combinação de duas tecnologias concorrentes, Future I/O, desenvolvida pela Compaq, IBM e Hewlett-Packard com a Next Generation I/O (ngio), desenvolvido por Intel, Microsoft, Dell, Hitachi, Siemens e Sun Microsystems. Em agosto de 1999, os sete líderes da indústria, Compaq, Dell, Hewlett-Packard, IBM, Intel, Microsoft e Sun Microsystems formaram a IBTA (InfiniBand Trade Association). A primeira especificação da arquitetura InfiniBand foi feita em junho de 2001.

A arquitetura InfiniBand surgiu devido à necessidade de se melhorar o desempenho dos dispositivos de E/S e das comunicações, que surgiu juntamente com o aumento da capacidade de processamento dos processadores.

InfiniBand é uma arquitetura ponto-a-ponto que se destina a fornecer aos centros de dados uma conectividade para entradas/saídas melhoradas e adaptadas a qualquer tipo de tráfego. Uma conexão InfiniBand substituirá os vários cabos atuais e servirá simultaneamente para a conectividade do cluster (proprietária), da rede (em vez do Gigabit Ethernet) e do armazenamento (em vez da atual Fibre Channel). É uma tecnologia comutada que utiliza três tipos de dispositivos, comutadores, interfaces HCA (Host Channel Adapter), que são os conectores usados

na comunicação interprocessadores do lado dos servidores e nas interfaces TCA (Target Channel Adapter), que são tipicamente usadas para conexão nos subsistemas de E/S.

A tecnologia InfiniBand utiliza uma estrutura hierárquica, com comunicação do tipo ponto-a-ponto. Nessa abordagem, todo nó pode ser o iniciador de um canal para qualquer outro. Ainda é possível que vários dispositivos de E/S peçam dados simultaneamente ao processador.

As duas principais vantagens do InfiniBand são a baixa latência e alta largura de banda. A baixa latência beneficia principalmente as aplicações sensíveis à latência, com comunicação entre processos (IPC) e sistemas gerenciadores de bancos de dados (DMBS). A alta largura de banda beneficia principalmente as aplicações que necessitam grande largura de banda, como armazenamento, web, computação de alto desempenho, e outras aplicações especializadas, como edição de vídeo.

Devido a suas características, InfiniBand é uma tecnologia adequada para aplicações de HPC (High Performance Computing). Enquanto InfiniBand provê muitas características avançadas que servem para um grande leque de aplicações, contudo esta tecnologia ainda é um padrão em evolução e que deve sofrer muitas melhorias. Algumas das melhorias planejadas para InfiniBand incluem especificações de maiores taxas de sinalização, controle de congestionamento e qualidade de serviço (QoS).

Gigabyte System Network

GSN é um padrão ANSI (American National Standards Institute) de tecnologia de rede que foi desenvolvida para redes de alta performance enquanto mantém compatibilidade com tecnologias de rede como HIPPI, Ethernet, e outros padrões de rede. GSN tem uma alta capacidade de banda (800MB por segundo) e baixa latência.

Características:

- Capacidade de Banda: acima de 800MB por segundo em full-duplex. Velo-

cidade comparável com Fibre Channel, ATM OC12, Gigabit Ethernet, and HIPPI;

- Latência: latência de 4 microseconds; latência do MPI é de 13 microseconds;
- Interoperabilidade: IP sob GSN, ST sob GSN, BDS sob GSN e ARP sob GSN;
- Biblioteca para diversos S.O.

Mais informações podem ser obtidas no endereço: <http://hsi.web.cern.ch/HSI/gsn/gsnhome.htm>

Quadrics Network, também conhecida como QSNET, consiste de dois blocos. Um chamado de "ELAN", que representa uma interface de rede programável, e outro chamado de "ELITE" que é caracterizado pelo switch de alto desempenho e baixa latência. Os dispositivos ELITE são interligados em forma de topologia "Flat-Tree", alcançando a possibilidade de interligação da ordem de milhares de dispositivos de comutação.

Scalable Coherent Interface (SCI)

SCI é um padrão recente de comunicação utilizado na interligação de componentes de um cluster. A abordagem cria um sistema de compartilhamento de memória global, através de um sistema de coerência de cache, baseado em diretórios distribuídos.

6.7 Protocolos de Comunicação

6.7.1 Frame Relay

O Frame Relay é uma eficiente tecnologia de comunicação de dados usada para transmitir de maneira rápida e barata a informação digital através de uma rede de dados, dividindo essas informações em frames (quadros) ou packets (pacotes) a um ou muitos destinos de um ou muitos end-points. Em 2006, a Internet baseada em ATM e IP nativo começaram, lentamente, a impelir o desuso do frame relay.

6.7.2 Asynchronous Transfer Mode

ATM é um protocolo de redes de computadores para comunicação de alto nível que encapsula os dados em pacotes de tamanho fixo (53 bytes; 48 bytes de dados e 5 de cabeçalho), em oposição aos de tamanho variável, comuns nas redes de comutação de pacotes (como os protocolos IP e Ethernet). No ATM, esses pacotes são denominados células. O protocolo VPI (Virtual Path Identifier) que é utilizado neste tipo de tecnologia de rede, possui 8 bits na interface UNI e 12 bits na interface NNI. A tecnologia ATM permite a transmissão de dados, voz e vídeo.

O ATM é uma tecnologia de comunicação ou mais especificamente, de comutação rápida de pacotes que suporta taxas de transferência de dados com variação de velocidades sub-T1 (menos de 1,544 Mbps) até 10 Gbps. Como outros serviços de comutação de pacotes (Frame Relay, SMDS), ATM atinge as suas altas velocidades, em parte, pela transmissão de dados em células de tamanho fixo, e dispensando protocolos de correção de erros.

6.7.3 FDDI

O padrão FDDI (Fiber Distributed Data Interface) foi estabelecido pelo ANSI (American National Standards Institute) em 1987. Este abrange o nível físico e de ligação de dados (as primeiras duas camadas do modelo OSI).

A expansão de redes de âmbito mais alargado, designadamente redes do tipo MAN (Metropolitan Area Network), são algumas das possibilidades do FDDI, tal como pode servir de base à interligação de redes locais, como nas redes de campus.

As redes FDDI adotam uma tecnologia de transmissão idêntica às das redes Token Ring, mas utilizando, comumente, cabos de fibra óptica, o que lhes concede capacidades de transmissão muito elevadas (na casa dos 100 Mbps ou mais) e a oportunidade de se alargarem a distâncias de até 100 Km.

Estas particularidades tornam esse padrão bastante indicado para a interligação de redes através de um backbone. Nesse caso, o backbone deste tipo de redes é justamente o cabo de fibra óptica duplo, com configuração em anel FDDI, ao qual

se ligam as sub-redes.

6.7.4 Modelo OSI

OSI (Open Systems Interconnection), ou Interconexão de Sistemas Abertos, é um conjunto de padrões ISO relativo à comunicação de dados. Um sistema aberto é um sistema que não depende de uma arquitetura específica.

O modelo tem como propósito facilitar o processo de padronização e obter interconectividade entre máquinas de diferentes sistemas operativos, a Organização Internacional de Padronização (ISO - International Organization for Standardization) aprovou, no início dos anos 80, um modelo de referência para permitir a comunicação entre máquinas heterogêneas, denominado OSI (Open Systems Interconnection). Esse modelo serve de base para qualquer tipo de rede, seja de curta, média ou longa distância.

6.7.5 Protocolo IP

IP é um acrônimo para a expressão inglesa "Internet Protocol"(ou Protocolo de Internet), que é um protocolo usado entre duas máquinas em rede para encaminhamento dos dados.

Os dados numa rede IP, são enviados em blocos referidos como pacotes ou datagramas (os termos são basicamente sinônimos no IP, sendo usados para os dados em diferentes locais nas camadas IP). Em particular, no IP nenhuma definição é necessária antes do host tentar enviar pacotes para um host com o qual não comunicou previamente.

O IP oferece um serviço de datagramas não confiável (também chamado de melhor esforço); ou seja, o pacote vem quase sem garantias. O pacote pode chegar desordenado (comparado com outros pacotes enviados entre os mesmos hosts), também podem chegar duplicados, ou podem ser perdidos por inteiro. Se a aplicação precisa de confiabilidade, esta é adicionada na camada de transporte.

O IP é o elemento comum encontrado na Internet dos dias de hoje. É descrito no RFC 791 da IETF, que foi pela primeira vez publicado em Setembro de 1981.

6.7.6 Transmission Control Protocol

O TCP (acrônimo para o inglês *Transmission Control Protocol*) é um dos protocolos sob os quais assenta o núcleo da Internet nos dias de hoje. A versatilidade e robustez deste protocolo tornou-o adequado para redes globais, já que este verifica se os dados são enviados pela rede de forma correta, na sequência apropriada e sem erros, pela rede.

O TCP é um protocolo do nível da camada de transporte (camada 4) do Modelo OSI e é sobre o qual assentam a maioria das aplicações cibernéticas, como o SSH, FTP, HTTP, portanto, a World Wide Web.

6.7.7 User Datagram Protocol

O UDP é um acrônimo do termo inglês *User Datagram Protocol* que significa protocolo de datagramas de utilizador (ou usuário). O UDP faz a entrega de mensagens independentes, designadas por datagramas, entre aplicações ou processos, em sistemas host. A entrega é “não confiável”, porque os datagramas podem ser entregues fora de ordem ou até perdidos. A integridade dos dados pode ser gerida por um “checksum”(um campo no cabeçalho de checagem por soma).

6.7.8 Real-time Transport Protocol

RTP (do inglês *Real Time Protocol*) é um protocolo de redes utilizado em aplicações de tempo real como, por exemplo, Voz sobre IP, que é a entrega de dados áudio ponto-a-ponto. Define como deve ser feita a fragmentação do fluxo de dados-áudio, adicionando a cada fragmento informação de sequência e de tempo de entrega. O controle é realizado pelo RTCP - *Real Time Control Protocol*. Ambos utilizam o UDP como protocolo de transporte, o qual não oferece qualquer

garantia que os pacotes serão entregues num determinado intervalo. Os protocolos RTP/RTCP são definidos pela RFC 3550 do IETF (Internet Engineering Task Force).

6.7.9 Virtual Router Redundancy Protocol - VRRP

O VRRP é designado para eliminar pontos de falhas criados por default-gateway de rede LAN (LOCAL AREA NETWORK).

VRRP é um protocolo especificado pela IETF² (RFC 3768) que permite dois ou mais roteadores atuarem como um roteador virtual. De acordo com essa especificação, os roteadores se apresentam para cliente com um endereço IP virtual (VIP - Virtual IP) correspondente a um MAC virtual (VMAC), mas cada qual com seu próprio IP e MAC reais. Se o roteador primário (master), que inicialmente possuía o dados virtuais, falhar então um roteador de backup (secundário) os assume.

As trocas de mensagem, para verificação de estado entre os servidores, acontecem através de IP multicast. Uma falha no recebimento dessas mensagens em um intervalo específico de tempo leva a um processo de eleição de um novo master. Em situação normal, apenas o master envia mensagens (IP multicast), apenas quando há escolha para novo master é que os servidores de backup enviam mensagens.

- . **Virtual Router** Roteador virtual, abstração formada por um ou mais roteadores rodando VRRP.
- . **VRRP Instance** Implementação em programa do protocolo VRRP rodando em um roteador.
- . **Virtual Router ID (VRID)** Identificação numérica para um Virtual Router em particular que deve ser único para cada segmento de rede.
- . **Virtual Router IP** Endereço IP associado ao um VRID que é usado por clientes para obter serviços dele. É gerenciado pela instância do VRRP que possui o VRID.

²Internet Engineering Task Force

- . **Virtual MAC address** Em casos em que endereço MAC é usado (Ethernet), este endereço MAC virtual é associado ao Endereço IP virtual.
- . **Priority** Valor (que varia de 1 a 254) associado a cada roteador rodando VRRP como maneira de determinar o master (quanto maior o número, maior prioridade).

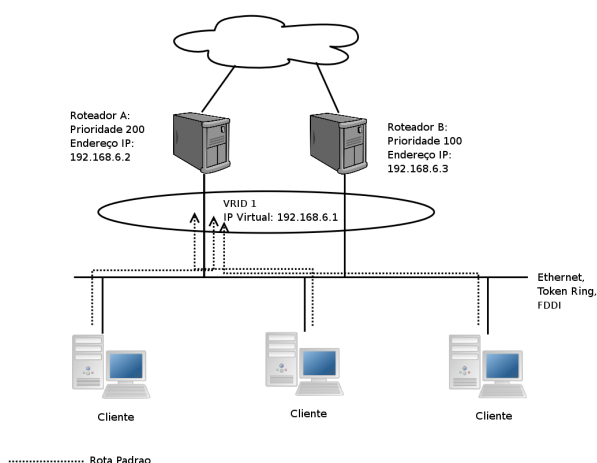


Figura 6.10: Esquema de funcionamento de um sistema VRRP

O servidor A envia pacotes multicast para outras instâncias do VRRP que rodem na rede (no caso, apenas o roteador B). Estes pacotes carregam informação para duas finalidades principais:

- . Forçar a eleição de outro master caso haja algum com maior prioridade.
- . Notificar instâncias VRRP de backup que há um master ativo, caso não haja comunicação em intervalo definido, haverá nova escolha de master.

Capítulo 7

Cluster de Armazenamento

7.1 Introdução

O aumento da capacidade de processamento e a maior utilização de sistemas informatizados para automatizar e auxiliar a execução dos mais variados processos e sistemas de informação, ocasionou um acúmulo de informações e de dados que necessitam ser armazenados e consolidados.

Conjuntamente com este aumento na demanda de armazenamento dos dados, a capacidade e as tecnologias de armazenamento evoluíram expressivamente nos últimos anos, chegando recentemente a alcançar PetaBytes¹.

No ambiente corporativo são utilizadas diversas mídias e tecnologias para armazenamento de dados, de uma maneira geral podem ser classificadas em dois grandes grupos:

- Tecnologias de Armazenamento Online - Encontram-se nesta categoria as tecnologias de armazenamento normalmente utilizadas por aplicações ou sistemas que demandam o acesso online aos dados. Alguns exemplos de tecnologias que encontram-se neste grupo são: Disco Rígido, Storage Devices, Sistemas de arquivos distribuídos, Sistemas de Arquivos Paralelos, Dispositivos Raid, Controladoras de Discos, entre outras.

¹ 1PetaByte = 1.073.741.824MegaByte

- Tecnologias de Armazenamento Offline - Encontram-se neste grupo as tecnologias de armazenamento normalmente utilizadas para armazenar dados de backup ou dados que não precisam ser acessados online. Alguns exemplos de tecnologias que encontram-se neste grupo são: Fitas, CD, DVD, dispositivos de fitas, bibliotecas de fitas.

Em sistemas críticos normalmente são utilizados dispositivos de armazenamento proprietários denominados "Storage Devices" e/ou "Bibliotecas de Fita" que possuem capacidade de armazenar Terabytes de informações, com funcionalidades que permitem consolidar e manter a integridade dos dados em um ambiente centralizado.

Existem alternativas tecnológicas de Cluster e Grid baseadas em padrões abertos de hardware e software para a implementação da camada de armazenamento e consolidação de dados em sistemas críticos.

Estas tecnologias em Cluster e Grid para armazenamento podem ser divididas em 3 categorias:

- Tecnologias baseadas em dispositivos de Blocos
- Sistemas de Arquivos Distribuídos
- Sistemas de Arquivos Paralelos

Sendo abordadas as principais tecnologias neste capítulo.

7.2 Block Devices

A definição básica de dispositivos de blocos é:

“Bloco especial de arquivo ou dispositivo de blocos são usados para corresponder a dispositivos pelos quais os dados são transmitidos na forma de blocos. Estes nós de dispositivo são freqüentemente usados para dispositivos de comunicações paralelos como discos rígidos e drives de CD-ROM.”[387]

“A diferença mais significativa entre dispositivos de blocos e dispositivos de caráter, é que os dispositivos de blocos tem rotinas de *buffer* para os controles de entrada e saída. O sistema operacional aloca um *buffer* de dados para prender cada bloco simples para a entrada e saída. Quando um programa envia um pedido de dados para ser lido , ou escrito, no dispositivo, cada caráter de dados é armazenado no *buffer* apropriado. Quando o *buffer* está cheio e um bloco completo é alcançado, a operação apropriada é executada e o *buffer* é limpo.”[387]

Os Dispositivos de Blocos são a parte de sustentação dos sistemas de arquivos dos sistemas operacionais. Sendo sua manipulação um processo básico para exploração de dispositivos de armazenamento.

Existem várias implementações de Dispositivos de Blocos com a intenção de serem utilizados em ambientes de Cluster. Neste capítulo serão apresentados os mais conhecidos.

7.2.1 Arranjo Redundante de Discos (RAID)

O Arranjo redundante de discos (**Redundant Array of Independent Disks - RAID**), é um sistema que usa múltiplos discos rígidos para compartilhar ou replicar dados entre esses discos. Dependendo da versão escolhida, o benefício do RAID é um ou mais vezes o incremento da integridade de dados, de tolerância à falhas, de desempenho ou de aumento de capacidade de armazenamento de dados, comparado com um simples disco.

Em suas implementações originais, sua vantagem chave era a habilidade de combinar múltiplos dispositivos de baixo custo usando uma tecnologia mais antiga em uma disposição que oferecesse uma grande capacidade de armazenamento, confiabilidade, velocidade, ou uma combinação destas. Num nível bem mais simples, RAID combina múltiplos discos rígidos em uma única unidade lógica. Assim, em vez do sistema operacional ver diversos discos rígidos diferentes, ele vê somente um disco rígido. O RAID é usado tipicamente em servidores, e geralmente, mas não necessariamente, é implementado com discos rígidos de tamanhos idênticos.

Com as diminuições dos preços de discos rígidos e com a disponibilidade em

larga escala de RAID em *chipsets* de placas-mãe, RAID vem se tornando uma opção para computadores de usuários mais avançados, assim como na criação de grandes sistemas de armazenamento de dados.

Usuários avançados vem usando RAID em suas estações de trabalho e *Workstations* para aplicações que necessitam de utilização intensiva de disco, seja de leitura/escrita ou mesmo capacidade de armazenamento, como no caso de aplicações de edição de vídeo e áudio.

7.2.2 RAID via Hardware e via Software

RAID pode ser implementado por hardware, na forma de controladoras especiais de disco, ou por software, como um módulo do kernel que fica dividido entre a controladora de disco de baixo nível e o sistema de arquivos acima dele.

RAID via hardware é sempre um controlador de disco, isto é, um dispositivo que pode através de um cabo conectar os discos. Geralmente ele vem na forma de uma placa adaptadora que pode ser “plugada” em um slot ISA/EISA/PCI/S-Bus/MicroChannel. Entretanto, algumas controladoras RAID vêm na forma de uma caixa que é conectada através de um cabo entre o sistema controlador de disco e os dispositivos de disco.

RAIDs pequenos podem ser ajustados nos espaços para disco do próprio computador; outros maiores podem ser colocados em um gabinete de armazenamento, com seu próprio espaço, para disco e suprimento de energia. O hardware mais novo de RAID usado com a mais recente e rápida CPU irá provavelmente fornecer o melhor desempenho total, porém com um preço expressivo. Isto porque a maioria das controladoras RAID vem com processadores especializados na placa e memória cache, que pode eliminar uma quantidade de processamento considerável da CPU. As controladoras RAID também podem fornecer altas taxas de transferência através do cache da controladora.

RAID via hardware geralmente não é compatível entre diferentes tipos, fabricantes e modelos: se uma controladora RAID falhar, é melhor que ela seja trocada por outra controladora do mesmo tipo. Para uma controladora de RAID via hardware poder ser usada no Linux ela precisa contar com utilitários de configuração e ge-

renciamento, feitos para este sistema operacional e fornecidos pelo fabricante da controladora.

RAID via software é uma configuração de módulos do kernel, juntamente com utilitários de administração que implementam RAID puramente por software, e não requer um hardware especializado. Pode ser utilizado o sistema de arquivos ext2, ext3, DOS-FAT, etc.

Este tipo de RAID é implementado através dos módulos MD do kernel do Linux e das ferramentas relacionadas.

RAID por software, por sua natureza, tende a ser muito mais flexível que uma solução por hardware. O problema é que ele em geral requer mais ciclos e capacidade de CPU para funcionar bem, quando comparado a um sistema de hardware, mas também oferece uma importante e distinta característica: opera sobre qualquer dispositivo do bloco podendo ser um disco inteiro (por exemplo, */dev/sda*), uma partição qualquer (por exemplo, */dev/hdb1*), um dispositivo de *loopback* (por exemplo, */dev/loop0*) ou qualquer outro dispositivo de bloco compatível, para criar um único dispositivo RAID. Isso diverge da maioria das soluções de RAID via *hardware*, onde cada grupo unifica unidades de disco inteiras em um arranjo.

Comparando as duas soluções, o RAID via *hardware* é transparente para o sistema operacional, e isto tende a simplificar o gerenciamento. Já o via *software*, tem mais opções e escolhas de configurações, fazendo sua manipulação mais complexa.

7.2.3 Distributed Replicated Block Device - DRBD

Projeto:DRBD

Sítio Oficial:<http://www.drbd.org>

Licença:GPL

Responsável: LinBit - <http://www.linbit.com>

DRBD é um acrônimo para o nome inglês *Distributed Replicated Block Device*. O DRBD consiste num módulo para o kernel de Linux, juntamente com alguns *scripts* e programas, que oferecem um dispositivo de blocos projetado para dispo-

nibilizar dispositivos de armazenamento distribuídos, geralmente utilizado em clusters de alta disponibilidade. Isto é feito espelhando conjuntos de blocos via rede (dedicada). O DRBD funciona, portanto, como um sistema RAID baseado em rede.

Como Funciona

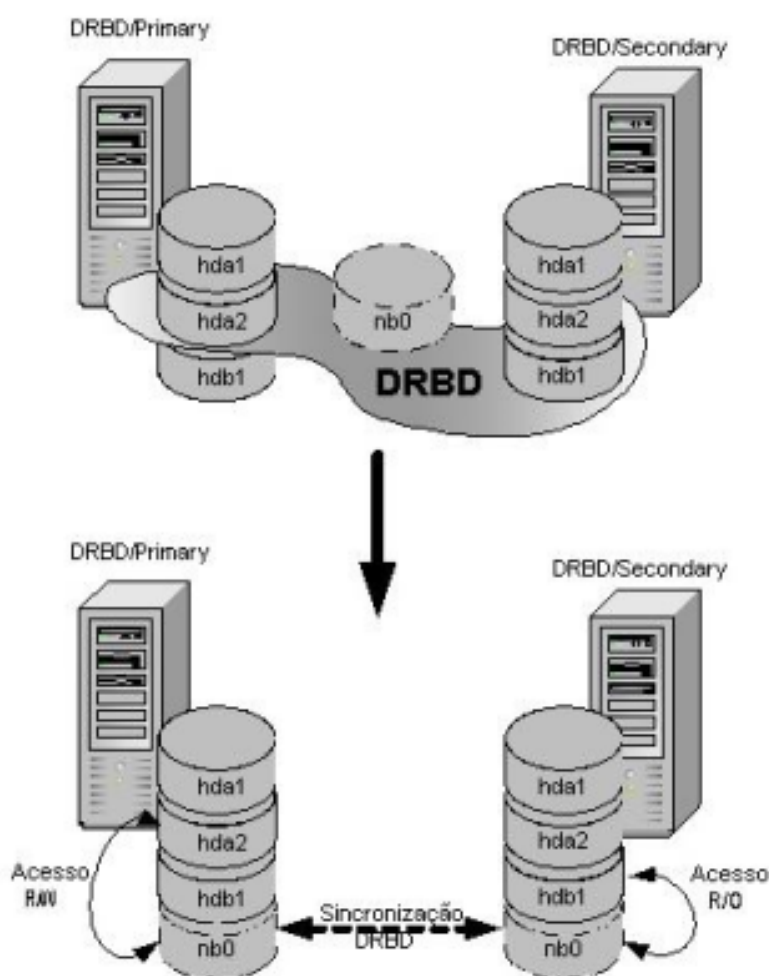


Figura 7.1: Visão do nível conceitual de funcionamento do DRBD - Trata-se de um *driver* intermediário entre o *block device* virtual (`/dev/drbd*`), o *block device* real local (`/dev/[sh]d*`) e os *block device's* remotos. Todas as transferências são efetuadas pelo protocolo TCP/IP, o que permite sua implementação até mesmo em máquinas geograficamente afastadas.

Cada dispositivo envolvido (tratados localmente como partições) tem um estado, que pode ser primário ou secundário. O DRBD cria, em todos os nós, um vínculo

entre um dispositivo virtual (`/dev/drbdX`) e uma partição local, inacessível diretamente. Toda a escrita é realizada no servidor primário, que irá transferir os dados para o "dispositivo de bloco do nível mais baixo" (a partição) e propagá-los para os restantes servidores, de estado "secundário". O secundário simplesmente escreve os dados no "dispositivo de bloco do nível mais baixo". As leituras são sempre realizadas localmente.

Se o servidor primário falhar, o DRBD mudará o dispositivo secundário para primário e as transferências passarão a ocorrer no sentido oposto. Note-se que o DRBD não trabalha ao nível do sistema de arquivos, mas sim ao nível de blocos do disco rígido. Nos sistemas de arquivos que não disponibilizam *journaling*, deverá ser realizada após a transição primário-secundário uma verificação da consistência do sistema de arquivos; em Linux, significa executar o *fsck*.

Para evitar a execução da verificação da consistência do sistema de arquivos, um processo altamente custoso, recomenda-se a utilização de sistemas de arquivos que possuam *journaling*.

Se o servidor que falhou retornar, o DRBD, mediante as configurações, devolverá - ou não - o estado de primário ao servidor original, após uma sincronização. Em caso negativo, o chamado modo *legacy*, o servidor que detém o estado de primário irá conservá-lo até o serviço ser encerrado nessa máquina.

Apesar do DRBD possuir o seu próprio modo de determinar qual dos servidores deverá ser primário, a sincronização com o sistema genérico não é trivial. Para reduzir estas dificuldades e a necessidade de interação do usuário, frequentemente é utilizado um sistema gerenciador de cluster, como o *heartbeat*, para tratar das transições de estado. Além desta transição de estados, o sistema será responsável por montar o sistema de arquivos na nova máquina que se tornou primária.

Características

Note-se, no entanto, que:

- Se as partições não forem do mesmo tamanho, o dispositivo DRBD irá automaticamente assumir-se como sendo do tamanho mínimo entre as partições

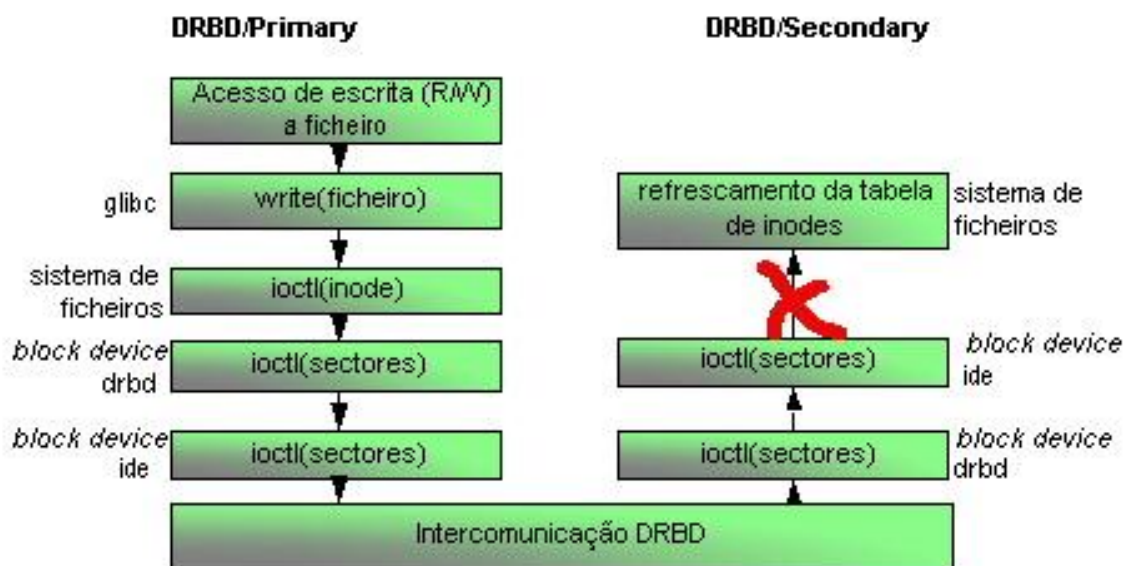


Figura 7.2: Fluxo de intercomunicação entre as camadas dos dispositivos Linux - repare que o DRBD não tem como notificar o módulo do sistema de arquivos - mas o oposto ocorre.

envolvidas - o que permite alguma flexibilidade relativamente aos discos disponíveis em cada nó.

- Apenas um dos nós pode ter acesso de leitura e escrita (ReadWrite, R/W)[KROVICH], e será designado como DRBD/Primary. O(s) restante(s) nó(s) será/serão designado(s) como DRBD/Secondary. Embora o nó DRBD/Secondary possa montar o dispositivo (apenas) em modo ReadOnly (R/O), é praticamente inútil, dado que as atualizações ocorrem apenas num sentido (do Primary para o Secondary) e a camada que gere *block device's* não dispõe de nenhuma forma de notificar alterações na camada que gere o sistema de arquivos embutido.

Situação do projeto

A maioria dos clusters HA (HP, Compaq, ...) atualmente utilizam dispositivos de armazenamento compartilhados; estes dispositivos, altamente dispendiosos, permitem que sejam conectados simultaneamente mais de um servidor, em geral através do barramento SCSI compartilhado ou Fiber Channel.

O DRBD usa a mesma semântica de um dispositivo compartilhado, mas não necessita de nenhum hardware específico. Ele trabalha no topo de redes IP, que são de implementação generalizada e de baixo custo, comparativamente aos sistemas

dedicados de armazenamento (como *Storage Area Networks* - SAN).

Atualmente o DRBD garante acesso de leitura-escrita apenas num servidor de cada vez, o que é suficiente para a implementação de sistemas *fail-over* típico de um cluster HA. Existe uma versão de desenvolvimento 0.8 que garante acesso de leitura-escrita para dois servidores, entretanto esta versão ainda não está estável suficiente para ser utilizada em ambientes de produção. As possibilidades de aplicação serão múltiplas, como para servidores web ou banco de dados de larga escala, onde operam sistemas de arquivos como o GFS, ou OCFS2, por exemplo.

Para se utilizar a versão de desenvolvimento do drbd(0.8), no modo Primário/Primário é necessário utilizar um sistema de arquivos compartilhado como por exemplo os citados acima. Somente um sistema de arquivos compartilhado é capaz de gerenciar o *lock* de acesso de maneira global ao sistema de arquivos, garantindo a integridade dos dados. Não se deve ser utilizado sistemas de arquivos comuns, tais como: xfs, ext3, reiserfs, neste modo de operação do drbd.

7.2.4 Global Network Block Device - GNBD

Projeto: Global Network Block Device

Sítio Oficial: <http://sources.redhat.com/cluster/gnbd/>

Licença: GPL

Responsável(eis): Red Hat

GNBD (*Global Network Block Devices*) [230] é um dispositivo que provê o acesso no nível de blocos a dispositivos de armazenamento remotos em uma rede TCP/IP. O GNBD é composto por um módulo no kernel e um conjunto de utilitários de sistema, possuindo uma parte servidora, que é responsável por exportar o disco via rede, e uma parte cliente, que é responsável por mapear localmente um disco remoto.

A parte servidora do GNBD é capaz de exportar qualquer dispositivo de blocos, alguns exemplos de dispositivos de blocos são: Disco Rígidos Ide ou SCSI, Pen-drive, Volumes lógicos, DRBD, dispositivos armazenados em *storage devices*.

Normalmente um servidor GNBD exporta um dispositivo de blocos local para um nó GFS[230]. Este dispositivo exportado em rede pode ser acessado localmente e remotamente por vários servidores simultaneamente, entretanto para manter a integridade dos dados é necessário utilizar um sistema de arquivos compartilhado, como por exemplo GFS ou OCFS2.

Também é possível agregar diversos dispositivos `gnbd` em um ou mais volumes lógicos LVM, utilizando a tecnologia de clusterização de volumes lógicos desenvolvida pela redhat CLVM. Através da utilização do GNBD e CLVM é possível criar uma estrutura de SAN² para armazenamento de arquivos em um cluster ou rede de servidores.

O gargalo de performance para configurações com um grande número de clientes, geralmente, encontra-se na conectividade de rede, pois o GNBD utiliza uma única *thread* para cada instância cliente-servidor-dispositivo, desta forma, quanto mais clientes melhor será a performance do servidor `gnbd` (em termos de *throughput* total). Obviamente, a performance por cliente irá diminuir, por conta da limitação da largura de banda. Outro fator de performance num ambiente que utilize `gnbd` é a performance do disco local no servidor, se esta performance for baixa, ela não será ampliada com a utilização do GNBD. Desta forma é recomendado sempre fazer uma análise da performance dos discos locais e da rede para manter um equilíbrio e tentar conseguir a maior performance possível.

É importante salientar que o GNBD não é um dispositivo de blocos distribuído ou replicado, de forma que os dados não são replicados ou distribuídos entre dois ou mais servidores.

A figura 7.3 apresenta um exemplo de cenário `gnbd` onde o dispositivo de blocos local do servidor `gnbd` é exportado via rede TCP/IP para 3 clientes `gnbd` que mapeiam este disco localmente.

7.2.5 Internet SCSI (iSCSI)

O *Internet SCSI* (iSCSI) é um protocolo de rede padrão, oficialmente ratificado em 11/02/2003 pelo IETF(*Internet Engineering Task Force*), que permite o uso do

²Storage Area Network

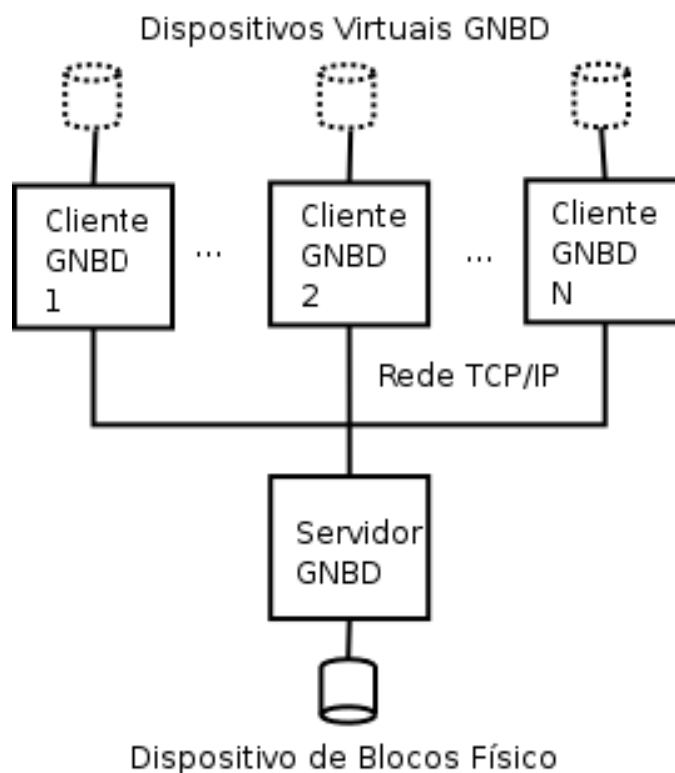


Figura 7.3: Exemplo de cenário GNBD

protocolo SCSI sobre redes TCP/IP. O iSCSI é um protocolo de camada de transporte nas especificações do *framework* SCSI-3. Outros protocolos de camada de transporte incluem a interface SCSI paralela e Fiber Channel.

A Aceitação do iSCSI em ambientes de produção corporativos foi acelerada pela grande utilização de gigabit ethernet nos dias de hoje. A construção de uma *Storage Area Network* (SAN) baseada em iSCSI se tornou mais barato, além de uma alternativa viável a utilização de SANs baseadas em Fiber Channel.

O protocolo iSCSI utiliza TCP/IP para sua transferência de dados. Diferente de outros protocolos de armazenamento em rede, como por exemplo Fiber Channel, para o seu funcionamento ele requer somente uma simples interface Ethernet ou qualquer outra interface de rede capaz de suportar TCP/IP. Este fato permite a centralização do armazenamento com baixo custo, sem o ônus e os problemas de incompatibilidade naturalmente associados a utilização de Fiber Channel em SANs.

Algumas pessoas críticas do protocolo iSCSI esperam uma performance pior que a existente no Fiber Channel, isto ocorre por conta do *overhead* adicionado pelo

protocolo TCP/IP na comunicação entre cliente e dispositivo de armazenamento. Entretanto, novas técnicas, como por exemplo *TCP Offload Engine* (TOE³) ajudam a reduzir este *overhead*. De fato, com a performance disponível nos servidores modernos, uma interface de rede padrão com um driver eficiente pode superar a performance de uma placa TOE porque menos interrupções e menos transferências de memória DMA são necessárias. As soluções iniciais de iSCSI são baseadas em uma camada de *software*. O Mercado iSCSI está crescendo rapidamente e deve melhorar em performance e usabilidade quanto mais organizações implementarem redes gigabit e 10 gigabit, e fabricantes integrarem suporte ao protocolo iSCSI nos seus sistemas operacionais, produtos SAN e subsistemas de armazenamento. iSCSI se torna cada vez mais interessante pois as redes ethernet estão começando a suportar velocidades maiores que o Fiber Channel.

Dispositivos de Armazenamento

No contexto do armazenamento em computadores, o iSCSI permite que um iSCSI initiator conecte a dispositivos iSCSI target remotos tais como discos e fita em uma rede do IP para I/O ao nível de bloco (*block level I/O*). Do ponto de vista dos drivers de sistema operacional e de aplicações de *software*, os dispositivos aparecem como dispositivos locais SCSI. Ambientes mais complexos, que consistem de múltiplos *Hosts* e/ou dispositivos, são chamados de Área de Armazenamento em Rede (Storage Area Networks - SAN).

Os dispositivos de iSCSI não devem ser confundidos com os dispositivos *Network-Attached Storage* (NAS) que incluem softwares de servidor para o controle de requisições de acessos simultâneos de vários *Hosts*. Permitir que múltiplos *Hosts* tenham acesso simultâneo a um simples dispositivo é uma tarefa comumente difícil para todos os dispositivos SCSI. Sem uma comunicação entre os hosts, cada um dos hosts não possui conhecimento do estado e intenção dos outros hosts. Esta condição ocasiona corrupção de dados e race conditions. Para realizar esta tarefa através da utilização do iSCSI é necessário utilizar um sistema de arquivos compartilhado como por exemplo o GFS e o OCFS2.

³http://en.wikipedia.org/wiki/TCP_Offload_Engine

iSCSI Conceitos e Visão Funcional

O protocolo iSCSI é responsável pela execução de comandos scsi entre dois dispositivos em uma rede TCP/IP. Comandos SCSI são executados através de chamadas iSCSI e os status SCSI são retornados como respostas.

Os termos *Initiator* e *Targets* referem-se à “iSCSI initiator node” e “iSCSI target node” respectivamente.

De acordo com protocolos similares, o *Initiator* e *Targets* dividem suas comunicações em mensagens. O termo *iSCSI protocol data unit* (iSCSI PDU) é usado para designar essas mensagens.

Por razões de performance, iSCSI permite *phase-collapse*. Através de um comando os seus dados associados, podem ser enviados em conjunto do *Initiator* para o *Targets* e os dados e respostas podem ser respondidos em conjunto pelos *Targets*.

O sentido de transferência do iSCSI é definido com respeito ao *Initiator*. Os limites de saída ou a saída de dados são transferidos do *Initiator* para o *Targets*, enquanto que o limite de entrada de dados ou os dados entrantes são transferências do *Targets* para o *Initiator*.

Implementações do *Initiator*, no Linux.

Linux *Initiators*:

- Core-iSCSI - Baseado em partes GPL do comercial PyX initiator <http://www.kernel.org/pub/linux/utils/storage/iscsi>.
- Intel-iSCSI (Intel) - Prova de conceito do iSCSI initiator e target da Intel para Linux <http://intel-iscsi.sf.net/>.
- Open-iSCSI - Nova implementação do initiator para Kernel 2.6.11 e superiores <http://www.open-iscsi.org/>.
- UNH-iSCSI - Initiator e target implementado pela University of New Hampshire.

Informações mais detalhadas sobre iSCSI podem ser obtidas nas RFCs: RFC-3720 <http://www.ietf.org/rfc/rfc3720.txt> e RFC-3783 <http://www.ietf.org/rfc/rfc3783.txt>

7.3 Sistemas de Arquivos Distribuídos

Os Sistemas de Arquivos Distribuídos (SADs) se destacam pelo acesso remoto aos dados de forma transparente para os usuários. Nas seções a seguir, realizamos uma resenha sobre alguns deles, começando com aqueles que deram início a toda pesquisa na área. É importante deixar claro que a maior parte dessa resenha foi baseada nos textos ([245], [246]).

7.3.1 Conceitos Básicos

Nesta sessão encontram-se alguns conceitos e serviços disponibilizados pelos sistemas de arquivos distribuídos e paralelos, assim como algumas de suas características, qualidades e soluções ([192], [349]) que os pesquisadores e desenvolvedores da área tentam prover.

Nomes e Localização

A maioria dos arquivos armazenados em um sistema de arquivos possui um nome e um caminho, que o identifica unicamente em tal sistema. Um caminho representa um nó de uma estrutura de diretórios, que pode ser representada como uma árvore (veja fig. 7.4).

Tal árvore possui somente uma raiz. Cada nó pode possuir árvores ou arquivos.

Dessa forma, para localizar um arquivo em uma árvore de diretórios (usados para agrupar arquivos) basta seguir o caminho do arquivo e, ao chegar no diretório final, procurar pelo nome de tal arquivo. A forma como esse nome e esse caminho são definidos depende muito do sistema operacional. Por exemplo, no Unix um

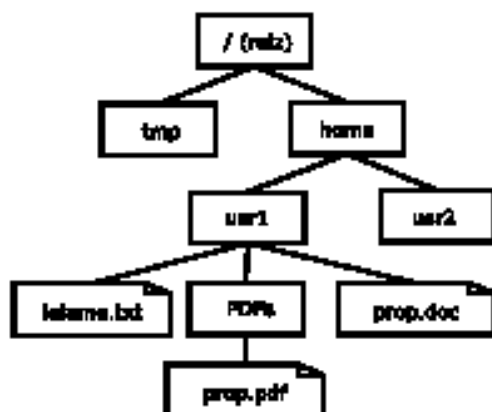


Figura 7.4: Exemplo de uma árvore de diretórios

caminho é definido como uma seqüência de nomes de diretórios, todos separados pelo caractere "/". O ultimo nome dessa seqüência pode ser o nome do arquivo ou de um diretório.

Em sistemas distribuídos, é possível encontrar o nome da máquina, em que o arquivo se encontra, dentro dessa definição de caminho. Porém procura-se deixar isso transparente para o usuário.

Cache

Para melhorar o desempenho no acesso aos arquivos de um sistema, procura-se guardar informações muito acessadas em memória, para evitar a sobrecarga de se ter que obtê-las novamente do meio físico onde estão armazenadas.

Isso ajuda muito na economia de tempo de processamento pois para acessar dados remotos, por exemplo, o sistema está limitado à velocidade da rede, que, mesmo rápida, estará limitada à velocidade do meio físico de armazenamento do servidor remoto, pois este ainda precisaria procurar os dados, carregá-los na memória e enviá-los para o cliente.

Mesmo no acesso a dados locais, a velocidade de acesso à memória é muito maior que a velocidade de acesso ao meio de armazenamento, por exemplo, um disco rígido, que precisaria mover o braço de leitura até a trilha em que se encontram

os dados e esperar até que a rotação do disco traga-os a cabeça de leitura.

Em sistemas de arquivos distribuídos, pode-se ter caches tanto no cliente como no servidor, evitando assim que o cliente acesse muito a rede para obter os dados do servidor, enquanto que o servidor diminui o acesso ao meio físico de armazenamento dos dados para enviá-los ao cliente.

O uso de cache é uma boa solução para o problema de desempenho no acesso aos arquivos, porém existem problemas como o de sincronização dos dados do cache com os dados do meio físico. Assim, se algum outro cliente alterar os dados no servidor, este precisa avisar a todos os clientes que seus caches podem estar com uma versão antiga dos dados.

Além disso, o tamanho do cache é reduzido, o que gera a necessidade de um algoritmo para saber quais dados devem permanecer no cache e quais podem ser removidos para dar lugar a novos dados. O ideal, segundo [349], é remover somente os dados que não serão mais acessados. Como não é possível prever isso, foram estudadas várias técnicas e algoritmos para que o resultado final chegue o mais próximo disso. O algoritmo LRU (*Least Recently Used*), segundo [349], é o que melhor se aproxima do ótimo e, talvez por isso, o mais usado nesse tipo de situação. Assim, sempre que um novo dado é acessado, este é incorporado ao cache. Se o cache estiver cheio, são removidos os dados que foram acessados há mais tempo, para dar lugar aos dados que estão vindo. Porém, se os dados retirados do cache tiverem sido alterados, estes devem ser enviados de volta ao servidor ou ao disco para serem gravados. Naturalmente, conforme o padrão de uso pode ser mais interessante usar outras políticas de substituição.

Transparências para o Usuário

Alguns sistemas de arquivos distribuídos (SADs) implementam características que o tornam transparentes para o usuário, que não precisa saber detalhes sobre o sistema de arquivos. Algumas dessas transparências são [192]:

- **Nome:** O nome do recurso a ser utilizado (como um arquivo ou diretório) não deve indicar ou conter indícios de onde este está localizado;

- **Localização:** O usuário não precisa fornecer a localização física do recurso para encontrá-lo;
- **Acesso:** O usuário não perceberá se o arquivo que está sendo usado é local ou remoto. Essa é a filosofia usada no sistema de arquivos virtual (VFS) do Solaris e do Linux;
- **Replicação:** Os arquivos do SAD podem ter cópias armazenadas em locais diferentes. O usuário não deve perceber que existem várias cópias do mesmo arquivo. Para ele só será apresentada uma, e quem a escolherá é o SAD;
- **Concorrência ou Paralelismo:** Vários usuários podem acessar o mesmo arquivo ao mesmo tempo, mas isso não deve ser perceptível para esses usuários;
- **Falha:** O SAD deve garantir que o acesso aos arquivos seja ininterrupto e sem falhas, sem que o usuário saiba como isso é tratado.

7.3.2 Serviços Oferecidos pelos SADs

Para proporcionar um ambiente simples e fácil de usar, escondendo toda a complexidade por trás dos engenhosos algoritmos e idéias desenvolvidas pelos pesquisadores em sistemas de arquivos, existem vários serviços oferecidos pelos SADs. Muitos deles acabaram se tornando essenciais e são adotados em vários sistemas. Além disso, por ser muito comum encontrá-los, vários possuem uma interface comum independente da implementação, para ajudar o desenvolvedor das aplicações que irão usar tais SADs.

Serviço de Nomes Distribuído

O serviço de nomes se preocupa em indicar a localização de um determinado arquivo, dado o seu nome ou caminho. Se a localização do arquivo estiver armazenada no nome dele, como por exemplo "jaca:/tmp/teste", então esse serviço de nomes não provê transparência de localização. Para prover essa transparência, o nome ou caminho de um arquivo não deve ter indícios de sua localização física.

Caso esse arquivo mude de lugar, ou tenha várias cópias, o seu nome ou caminho não precisará ser alterado para ser encontrado. Para isso, o serviço precisa oferecer ou **resolução por nomes**, ou **resolução por localização**, ou ambos.

Resolução por nomes mapeia nomes de arquivos legíveis por humanos para nomes de arquivos compreensíveis por computadores, que normalmente são números, facilmente manipuláveis pelas máquinas. Por exemplo, o endereço `www.example.com` é mapeado para o IP `192.0.34.166`. Através desse conjunto de números é possível encontrar uma máquina na rede Internet, utilizando-se de tabelas de rotas de endereços mascarados que indicam como chegar a posição desejada.

Resolução por localização mapeia nomes globais para uma determinada localização. Por exemplo, números de telefone possuem código do país, da localidade, etc. Se transparência por nome e por localização estiverem sendo utilizadas simultaneamente, pode ser muito difícil realizar um roteamento para determinar a localização de um determinado nome. Soluções com servidores centralizados ou distribuídos são opções, porém os centralizados podem se tornar um gargalo, enquanto os distribuídos precisam usar alguma técnica de descentralização, como por exemplo cada servidor é responsável por um determinado subconjunto de arquivos, ou cada um resolveria a localização de determinados tipos de arquivos.

Serviço de Arquivos Distribuído

O serviço de arquivos é responsável por fornecer operações sobre os arquivos que compõem o sistema, que podem ser armazenados de diferentes formas, dependendo do seu tipo e uso. Por exemplo, arquivos que compõem um banco de dados podem ser armazenados em formato de registros, arquivos que são usados por aplicações multimídia costumam ser armazenados em formato contínuo no disco, para agilizar sua leitura, etc.

Esse serviço também cuida das propriedades dos arquivos, como data de criação, data de alteração, tamanho, dono do arquivo, permissões de leitura, escrita e execução, além de qualquer outra informação relevante. Tais informações são chamadas também de meta-dados.

Também é responsabilidade desse serviço manter a integridade das operações realizadas nos arquivos. Por exemplo, quando uma aplicação altera algum arquivo, todas as demais aplicações que estiverem acessando-o devem perceber essa alteração o mais rápido possível.

Existem dois tipos de implementação de serviço de arquivos: acesso remoto e cópia remota, que podem ser com ou sem estado. No caso do acesso remoto, o cliente não possui um espaço para guardar os arquivos que estiver usando e toda e qualquer operação realizada com os arquivos será sempre através da rede. Isso pode deixar o sistema muito lento, já que depende da velocidade da rede.

Já no caso da cópia remota, o cliente recebe uma cópia do arquivo para trabalhar e, quando tiver terminado, devolve as alterações para o servidor. Isso só funciona se o cliente tiver espaço suficiente para armazenar o arquivo. A velocidade da rede só influenciará durante as transmissões do arquivo de um local a outro e a implementação só será considerada muito eficiente caso o arquivo seja totalmente alterado. Porém, se o cliente só se interessar por um determinado trecho do arquivo, recursos de transmissão estarão sendo gastos sem necessidade. Daí existe uma variante dessa implementação onde somente os blocos que se quer trabalhar são enviados para o cliente, chamada de cache de bloco.

É possível também que esse serviço lide com replicação de arquivos, que ajuda no acesso concorrente, dando maior velocidade para as aplicações dos clientes, ajudando-o também a deixar os arquivos sempre disponíveis, caso algum servidor fique fora do ar.

Serviço de Diretórios Distribuído

Esse serviço é responsável por manter a organização dos arquivos armazenados no sistema. Ele fornece uma interface para que os usuários possam arranjar seus arquivos de forma hierárquica, que é estruturada em diretórios e subdiretórios. Na maioria dos casos, um subdiretório só tem um único pai.

O serviço precisa manter uma lista de todos os diretórios ativos, junto com seus respectivos arquivos. Ele precisa ter a capacidade de identificar e remover arquivos que não estejam em diretório algum, como por exemplo quando um diretório

é removido. No caso em que são permitidos múltiplos pais, essa tarefa não é tão fácil, pois os arquivos ou subdiretórios ainda podem estar sendo referenciados por outros diretórios ou recursos. Para resolver esse problema, é colocado um contador de referências para arquivos e diretórios, que se chegar a zero significa que o arquivo ou diretório não possui nenhum outro recurso (como *hard links*, subdiretórios, etc.) apontando para ele, podendo então ser removido. Note que, geralmente, os links simbólicos não influenciam nesse contador. Exemplificando, a figura 7.5 mostra uma estrutura de diretórios distribuída em dois servidores. Se for requisitada a remoção da ligação $A - E$ (um *hard link*), ou da ligação $B - E$, ou da ligação $C - E$ (um link simbólico), somente a ligação pedida será removida. Porém, somente as duas primeiras alteram o contador do diretório E , indicando quantas referências apontam para ele. Assim, se forem removidas as ligações $A - E$ e $B - E$ esse contador chegará a zero e os nós E , F e G serão removidos. No caso, o diretório F está em um servidor diferente do diretório raiz a ser removido, mas o serviço de diretórios deve cuidar disto.

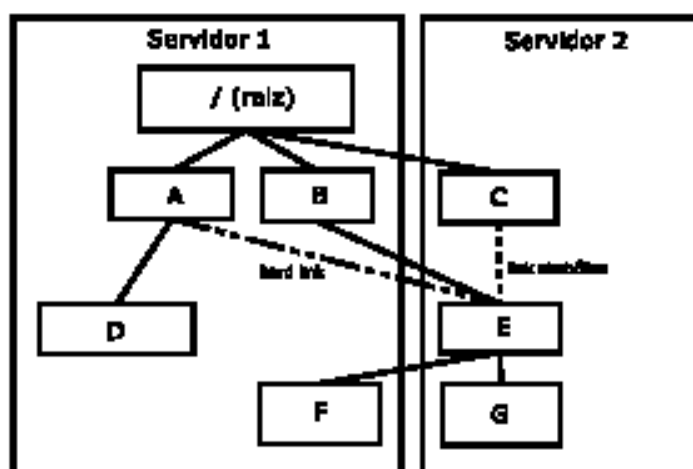


Figura 7.5: Estrutura de diretórios distribuída

Algumas das operações sobre diretórios oferecidas pelos serviços de diretórios são criação, remoção, alteração, listagem, alteração de permissões. Além delas, o serviço também influencia no gerenciamento de arquivos, como nas operações de criação, remoção, mudança de nome, busca, duplicação, entre outras.

7.3.3 Algumas Características Desejadas em SADS

Qual sistema de arquivos usar em um sistema distribuído? Para resolver essa dúvida, primeiramente analisa-se qual o tipo de aplicação que será utilizada e, a partir disso, tenta-se descobrir o que é mais importante para ela, como tolerância a falhas, acesso concorrente, ou alguma outra característica. Algumas dessas características ([192], [246]) são descritas nessa sessão.

Disponibilidade

Depender de um serviço da rede que saiu do ar por questões de falta de energia elétrica, manutenção ou falha do hardware é algo inconveniente, ainda mais quando ocorre perda dos dados.

Dessa forma, existem vários estudos para evitar que os serviços deixem de ser oferecidos, seja qual for o motivo.

Se um servidor cair, ficar fora do ar ou da rede, o sistema de arquivos não pode perder informações e nem ficar inacessível total ou parcialmente. Além disso, o usuário não precisa saber como isso foi implementado. Ele simplesmente requisita um arquivo e o sistema de arquivos deve entregá-lo, mesmo que algum dos servidores esteja fora do ar. O arquivo não pode ficar indisponível e isso deve ser totalmente transparente ao usuário. Isso se chama transparência a falhas.

É muito comum sistemas ficarem indisponíveis não por falha do próprio servidor, mas por falha na rede de comunicações. Caso um cliente deseje acessar um arquivo que está em um servidor inacessível naquele momento, o sistema operacional do cliente costuma travar o processo que o pediu até que o servidor volte a ficar acessível.

Uma das soluções para se resolver esse problema é a replicação dos dados, ou seja, o mesmo arquivo, ou parte dele, deve estar em diferentes servidores. Assim, caso algum servidor fique fora da rede, outro fornecerá os mesmos arquivos. Alguns sistemas de arquivos distribuídos foram criados levando esse conceito às últimas consequências, como é o caso do CODA, detalhado na sessão 7.3.6.

Escalabilidade

Os sistemas distribuídos são, em geral, projetados e configurados pensando-se na configuração da rede naquele momento. Porém essa rede pode aumentar, ou seja, dezenas ou centenas de novos nós podem ser adquiridos e conectados nesse sistema. A menos que se tenha considerado essa situação no momento do projeto da rede, dificilmente um sistema de arquivos distribuído apresentará bom desempenho para servir a todos os clientes após esse crescimento [246].

Vários problemas podem ocorrer, dentre eles, a inutilização da vantagem de se usar cache quando o servidor responde a vários pedidos de vários clientes. O servidor mantém os dados enviados em cache, para permitir uma rápida resposta caso esse mesmo dado seja requisitado novamente. No caso de se ter muitos clientes, têm-se muitos pedidos diferentes, fazendo com que as tabelas do cache sejam atualizadas com frequência, sem a reutilização dos dados lá contidos.

Caso se tenha cache do lado dos clientes, ao se alterar um arquivo que está sendo usado por muitas outras máquinas, o servidor terá que avisá-las que o cache local das mesmas está inválido e todas deverão se atualizar com a versão do servidor, causando sobrecarga.

Por outro lado, caso se tenha estimado que a rede seria muito grande e se tenha distribuído sistema de arquivos em muitos servidores, fica difícil descobrir onde um arquivo está armazenado fisicamente. Por exemplo, se para abrir um arquivo um cliente tiver que perguntar para cada servidor se ele é o responsável por aquele arquivo, certamente haverá um congestionamento na rede. Caso se tente resolver isso colocando um servidor central para resolver todos os caminhos para os arquivos, indicando a localização do mesmo, tal servidor sofrerá sobrecarga.

Um sistema escalável é um sistema que leva em conta esses problemas e tenta evitar sua ocorrência quando o número de clientes aumenta muito. O ANDREW é um sistema de arquivos que conseguiu adotar uma solução satisfatória [245] para esses problemas, através da descentralização das informações e da hierarquização da rede. Esse SAD é descrito na sessão 7.3.5.

Segurança

Compartilhar arquivos entre vários ambientes e usuários é uma das vantagens que os sistemas de arquivos distribuídos trazem. Porém, deixar que outras pessoas possam acessar arquivos confidenciais é um grande problema. Dessa forma, torna-se necessário adotar mecanismos de segurança, para evitar que pessoas desautorizadas tenham acesso aos arquivos do sistema.

Sistemas Unix adotam um método baseado em **permissões** para controlar o acesso aos seus arquivos [246]. Cada arquivo possui informações sobre quais usuários podem acessá-lo e de que maneira.

Nos sistemas distribuídos que executam sob o Unix, quando um servidor recebe um pedido para enviar dados de um determinado arquivo, ele também recebe informações sobre qual usuário está tentando realizar tal acesso. Com isso, verifica se tal usuário tem permissão suficiente para realizar essa solicitação, fazendo uma comparação com as informações de permissões do arquivo.

Outra forma de implementar esse controle de segurança é um sistema baseado em **capacidades** [349], que consiste em enviar ao servidor uma prova de que ele possui a capacidade de acessar um determinado arquivo. Na primeira vez que o usuário acessa tal arquivo, é enviado ao servidor sua identificação e o servidor, por sua vez, retorna um código que é a sua prova de capacidade para acessar aquele arquivo. Nas próximas requisições, o cliente não precisa se identificar novamente, bastando apenas enviar a prova de sua capacidade. Deve-se tomar cuidado para não criar provas de capacidade que sejam fáceis de ser forjadas.

E possível, também, implementar o controle de segurança através de **listas de controle de acesso** [349], onde cada elemento da lista possui as permissões que cada usuário tem para acessar determinado arquivo. Isso evita que se crie, por exemplo, uma matriz de arquivos por usuários, onde cada elemento representa o tipo de acesso (o que utilizaria muita memória, dado que muitos desses elementos seriam iguais). O sistema de arquivos distribuído ANDREW utiliza esse mecanismo de listas de controle de acesso.

O controle no acesso aos arquivos é uma das medidas de segurança para protegê-los. Porém, caso haja outras máquinas no caminho de duas máquinas confiáveis,

existe o risco de se ter dados interceptados ou, até mesmo, adulterados. Uma forma de se resolver esse problema é criptografar as informações antes de enviá-las.

O sistema de arquivos SWALLOW [246] é um sistema de arquivos distribuído que transmite os arquivos criptografados. Ele funciona em um sistema baseado em capacidades, onde a prova da capacidade é a chave criptográfica. A vantagem é que o servidor não precisa verificar se a prova da capacidade é autêntica: se ela não for, o cliente não conseguirá decodificar os dados.

Meios mais modernos e eficazes para o controle da segurança no acesso e manipulação dos dados armazenados podem ser encontrados em [192].

Tolerância a Falhas

Durante a transmissão dos dados entre servidores e clientes, falhas podem ocorrer, seja por excesso de tráfego de pacotes pela rede, seja por algum dos servidores estar sobrecarregado. Além disso, podem ocorrer falhas de hardware, especialmente dos mecanismos de armazenamento, de transmissão, etc. Esses problemas acontecem em grande parte porque os sistemas distribuídos são implementados sobre redes de computadores que não são totalmente confiáveis.

Dessa forma, um sistema distribuído precisa usar um protocolo de comunicação com capacidade para detecção de erros de transmissão. Assim, caso uma mensagem chegue alterada no seu destino, o protocolo precisa perceber isso e retransmiti-la. Isso deve ocorrer também para mensagens que se perderam no caminho. Um outro problema que a rede pode ter é o seu particionamento por tempo indeterminado.

Além disso, o hardware dentro das máquinas também pode apresentar falhas. Por exemplo, um disco rígido pode deixar de funcionar de um momento para outro. Nesse caso, soluções como redundância física do equipamento (realizada através de hardware) ou redundância controlada pelo próprio sistema distribuído, que cuidaria de replicar os dados, já evitaria a perda das informações armazenadas.

Seja qual for o problema, o sistema deve evitar que o cliente fique aguardando uma resposta por muito tempo, ou que seus dados sejam danificados ou até mesmo perdidos. Isso significa que o serviço precisa ter **disponibilidade e confiabilidade**.

Porém, essas características podem ser conflitantes se não forem bem aplicadas. Por exemplo, para garantir a confiabilidade é necessário implementar redundância dos dados, porém a complexidade que isso gera pode aumentar demais a carga do servidor, comprometendo a disponibilidade, pois as respostas aos clientes seriam mais lentas.

Outro mecanismo que auxilia a confiabilidade é a **transação**. Ela evita que o conteúdo de algum arquivo fique em um estado inconsistente caso haja uma queda do servidor ou cliente durante a execução de alguma operação sobre o mesmo. Maiores detalhes sobre transações são descritas na próxima sessão.

Operações Atômicas

Uma operação em um arquivo é dita **atômica** quando as etapas da mesma não podem ser percebidas por outros processos exteriores a esta operação [349]. Assim, antes dessa operação o arquivo apresenta um estado e após outro, sem que apresente nenhum outro estado intermediário durante a operação. Caso alguma etapa falhe durante a operação, o arquivo volta ao estado inicial.

Dessa forma, ou todas as etapas são realizadas com sucesso, ou nenhuma será realizada.

Operações de leitura, escrita, criação ou remoção de um arquivo são implementadas de forma atômica pela maioria dos sistemas de arquivos.

Transações são mecanismos que permitem realizar uma seqüência de operações de forma atômica. Tais mecanismos disponibilizam determinados comandos para os usuários para que possam escolher quais operações serão executadas dentro de transações. Para montar uma transação, existem os comandos início e fim. O comando de início avisa ao sistema que todas as operações a partir daquele ponto estarão dentro da transação e o comando de finalização indica que

não virá mais nenhuma operação para aquela transação.

Assim, caso alguma dessas operações falhe, o sistema ou desfaz, ou aborta todas as alterações que as operações antes daquela realizaram. Isso é chamado de *rollback* ou *abort*. Caso todas as operações sejam executadas sem problemas ou erros, ao chegar no fim da transação é realizado um *commit*, ou seja, todas as alterações que foram executadas são efetivadas e persistidas, de tal forma que outros processo possam percebê-las.

Com isso as transações implementam a semântica do tudo ou nada, ou seja, ou todas as operações são executadas com sucesso, ou nenhuma será executada. Isso faz das transações um importante mecanismo de tolerância a falhas, pois elas evitam que pequenas falhas prejudiquem a integridade de todo o sistema.

Embora as transações sejam implementadas de forma praticamente obrigatória em sistemas de bancos de dados, elas não costumam ser implementadas em sistemas de arquivos. Os sistemas LOCUS e o QuickSilver [246] são algumas exceções à regra, pois implementam transações em sistemas de arquivos distribuídos.

Acesso Concorrente

Vários usuários podem acessar vários arquivos, ou os mesmos arquivos, sem sofrer danos, perda de desempenho ou quaisquer outras restrições. Isso tudo deve ocorrer sem que o usuário precise saber como o acesso é realizado pelos servidores. Assim, é necessário haver transparência de concorrência e de paralelismo.

O maior problema encontrado nas implementações desse tipo de solução é quanto à sincronização dos arquivos, o que inclui leitura e escrita concorrente. A leitura concorrente pode ser implementada facilmente se não houver escrita concorrente, pois quando um arquivo estiver sendo lido, certamente ninguém poderá escrever nele. Caso também se queira escrita concorrente, deve-se levar em conta que quando um cliente escreve em um arquivo, todos os leitores devem ser avisados que o arquivo foi alterado e todos escritores precisam tomar cuidado para não escrever sobre as alterações que foram feitas por outros. Assim, ou vale a última alteração, ou os escritores discutem entre si para tentar fazer uma fusão das alterações.

Para se ter uma idéia da complexidade desse problema, imagine duas operações bancárias simultâneas na mesma conta. Uma delas é um saque de R\$100,00 e outra é um depósito de R\$1000,00. Antes dessas operações, suponha que essa conta possua R\$100,00 de saldo e também suponha que esse valor esteja armazenado em um arquivo de um sistema de arquivos distribuído. Quando o cliente da conta for realizar o saque, a aplicação irá armazenar em memória o valor atual do saldo, assim como acontecerá com a aplicação do outro caixa que estará recebendo o depósito. Esta aplicação, então, irá adicionar ao saldo o valor do depósito e gravará no arquivo o novo saldo, que será de R\$1100,00. Porém, a primeira aplicação irá subtrair do valor armazenado em memória (que para seu contexto é de R\$100,00) o valor do saque e gravará o resultado (R\$0,00) no mesmo arquivo, sobrescrevendo o valor lá existente. Dessa forma, o cliente perderia seu depósito.

Para evitar esse tipo de problema, as aplicações que operam dessa forma podem agrupar um conjunto de operações no sistema de arquivos como sendo uma única transação, deixando a cargo do sistema operacional gerenciar a melhor forma de executar isso. Existem alguns mecanismos para o controle dessa concorrência, como por exemplo o uso de bloqueios, o controle de otimista e o de controle por data e hora, que são encontrados em ([349], [192]).

O mecanismo de bloqueios destaca-se por ser amplamente utilizado, e baseia-se no bloqueio do arquivo que se quer acessar antes de acessá-lo, através de uma chamada ao sistema operacional ou ao sistema de arquivos. Caso um segundo processo queira usar o mesmo arquivo, tentará realizar o bloqueio, usando o mesmo comando que o primeiro processo. O sistema operacional ou o sistema de arquivos o avisará caso esse arquivo esteja bloqueado. Se estiver, cabe ao processo decidir se espera na fila pelo desbloqueio ou se continua seu processamento sem o acesso ao arquivo. Esse desbloqueio é realizado pelo processo detentor do arquivo, através de um comando, similar ao usado para o bloqueio.

Através desses bloqueios, é tornar as transações serializáveis, isto é, o resultado da operação de várias transações simultâneas é o mesmo obtido se elas fossem realizadas uma após a outra [246]. Um protocolo para a realização dessa serialização é o protocolo de bloqueio de duas fases, onde na primeira fase ocorre o bloqueio de todos os arquivos a serem usados nessa transação e, na segunda fase, a liberação conjunta de todos os arquivos, após a realização das operações dentro dessas fases.

Porém esse protocolo pode gerar um travamento (*deadlock*), onde um processo esperaria a liberação de um arquivo que foi bloqueado por outro processo, que também estaria esperando a liberação de um arquivo que foi bloqueado por aquele primeiro processo, por exemplo. Para evitar travamentos em sistemas distribuídos, existem técnicas e algoritmos que fogem do escopo deste trabalho, devido à complexidade, mas que são descritos em [192], [349].

Replicação de Arquivos

Em um ambiente distribuído, pode-se também distribuir a carga causada pelo acesso aos arquivos nos vários servidores que compõe o sistema. Pode-se replicar os arquivos em mais de um servidor, ou então replicar somente os arquivos mais acessados, ou ainda replicar somente os pedaços dos arquivos que costumam ter um alto nível de acesso. Note que o uso de cache em um sistema de arquivos pode ser encarado como uma replicação de arquivos, embora seu objetivo seja principalmente desempenho.

Além disso, se um sistema de arquivos oferece essa funcionalidade, a eficiência e a confiança do serviço de arquivos é generosamente aumentada. Eficiência em termos de tempo de resposta, carga do servidor e tráfego de rede. Confiança caso um determinado servidor caia ou fique indisponível, pois o serviço de arquivos ainda pode realizar suas obrigações por possuir cópias dos dados em outro ponto da rede.

Dessa forma, replicação de arquivos provê tolerância a falhas, já que o usuário pode não perceber que o servidor que ele estava usando caiu e que outro entrou no lugar para prover o recurso que ele estava usando. Daí o sistema também deve oferecer transparência de replicação, pois o usuário não precisa saber como o sistema cuida da replicação desse arquivo.

O maior problema nessa característica do SAD é que a implementação pode ser muito complicada, pois é necessário manter os dados sincronizados e coerentes ao mesmo tempo.

Existem soluções centralizadas e distribuídas [192] para esse tipo de problema.

A centralizada consiste de um único servidor que cuida dos pedidos dos clientes através de *handles*. Com esses *handles*, os clientes acessam diretamente os arquivos através dos servidores e secundários. Porém, caso o servidor primário caia, nenhum outro cliente conseguirá abrir nenhum outro *handle*, somente os que já estavam abertos continuam acessando os arquivos.

No caso da solução distribuída, existem dois tipos de implementações: a primeira utiliza comunicação em grupo, que consiste em quando ocorrer uma alteração por algum dos servidores, este manda um *broadcast* para os outros servidores dizendo que o arquivo foi alterado. Estes, por sua vez, podem alterar esse arquivo imediatamente ou somente quando forem utilizá-lo; a segunda utiliza votação e números de versão. Isso significa que quando um cliente pedir permissão para alterar um arquivo, os servidores votarão entre eles pra saber quem possui a versão mais recente.

Esse servidor será o servidor padrão daquele arquivo e seu número de versão será incrementado.

Todas essas idéias, além de serem complicadas de implementar, geram alguns problemas. Manter a sincronização entre os servidores, para o caso de alterações no sistema de arquivos, é uma delas.

Tal tarefa não pode interferir no desempenho (por causa da disponibilidade) e nem no uso do sistema pelos usuários (devido à transparência).

Os Primeiros Sistemas de arquivos Distribuídos

O primeiro SAD que se tem notícia, segundo [246], usava a ARPANET, rede construída pelo Departamento de Defesa dos Estados Unidos em 1969 que entrou em funcionamento em 1973. Ele disponibilizava um repositório de dados para computadores que não possuíam capacidade de armazenamento adequada. Era chamado de Datacomputer e usava um serviço parecido com o FTP atual.

Depois dele, veio o *Interim File Server* (IFS), criado por pesquisadores do *Xerox Palo Alto Research Center* (PARC). Ele já organizava os arquivos privados e compartilhados em uma árvore de diretórios. Um sistema subsequente foi o *Woods-*

tock File Server (WFS), criado também pelo PARC, que permitia enviar aos clientes somente páginas dos arquivos, ao invés de enviar o arquivo completo, possibilitando trabalhar assim com máquinas sem discos locais.

Em 1977, o PARC criou o *Xerox Distributed File System*, destinado a oferecer uma base para a implementação de sistemas gerenciadores de banco de dados. Ele já implementava transações atômicas envolvendo vários arquivos e servidores, usando um protocolo de duas fases e o acesso a pequenos trechos de arquivos.

Depois dele vieram o LOCUS (1980) que já implementava transparência de localização, replicação e transações atômicas aninhadas; o SWALLOW (início dos anos 80) do MIT, que usava uma técnica de controle de acesso concorrente baseado em *timestamps*; o *Acorn File Server* (início dos anos 80), desenvolvido para implantação de uma rede de microcomputadores em escolas a um custo muito baixo; e o VICE (1984), ancestral do AFS e do CODA.

7.3.4 Network File System - NFS

Projeto:Network Filesystem

Sítio Oficial:<http://nfs.sourceforge.net>

Licença:GPL

Responsável:

Network File System [339], [245], [246], [269], sistema de arquivos distribuído desenvolvido inicialmente pela Sun, é o SAD mais utilizado em sistemas Unix. Em 1985 a Sun tornou público seu protocolo, o que permitiu que outras empresas e desenvolvedores pudessem criar clientes e servidores compatíveis.

Hoje em dia, já é possível encontrar implementações do NFS (tanto cliente como servidor) para quase todos os sistemas operacionais existentes, inclusive sistemas não-UNIX, como o Windows. Isso também foi facilitado pelo fato do NFS definir uma interface RPC [231] que utiliza uma representação de dados independente de máquina chamada *External Data Representation* (XDR).

As versões mais usadas do NFS são as 2 e 3, porém já existe a RFC3530 [328]

que descreve o NFSv4. Assim como as versões antigas são incompatíveis entre si, essa nova versão também será. As diferenças e características de cada uma dessas versões, levando em conta funcionamento, desempenho e segurança, estão detalhadas na seção a seguir.

Características do NFSv2 e NFSv3

Os servidores NFSv2 e NFSv3 não guardam o estado das transações realizadas. Isso faz com que não percam nenhuma informação enviada em caso de falha na transmissão ou nos serviços, agilizando sua recuperação. Os clientes também não precisam se preocupar com essa falha, pois basta pedir os dados novamente para o servidor, até que ele responda.

Por outro lado, servidores que não guardam o estado das transações realizadas não conseguem gerenciar *locks* e nem realizar transações atômicas. Existem soluções disponibilizadas à parte para resolver alguns desses problemas, como um servidor de *locks*, chamado de *Network Lock Manager* [227], para auxiliar as políticas de acesso a arquivos de forma concorrente. Também pelo fato do NFS não manter estado, ele não pode controlar o acesso concorrente aos seus arquivos e nem garantir a sua consistência.

No NFSv3 o mecanismo de cache do servidor foi alterado para possuir tamanho variável (antes era constante) e sua política de escrita foi alterada do *write-on-close* (após se fechar o arquivo, este é gravado em disco) para o *delayed-write* (o arquivo é gravado em disco após ficar algum tempo no cliente sem ser alterado). Assim, caso um arquivo seja usado constantemente e depois apagado, nada será gravado.

Outra vantagem do protocolo NFSv3 em relação ao NFSv2 é o fato de que este ultimo limitava o tráfego de dados de arquivos em blocos de 8KB, enquanto que aquele permitiu enviar dados entre servidor e cliente em blocos de até 56Kbytes via UDP. Além disso, no NFSv2 o servidor só retorna o resultado da gravação desses 8Kbytes após eles estarem gravados fisicamente. Isso consumia muito tempo pois só se gravava em blocos de 8KB. No NFSv3 o disco pode gravar uma quantidade de dados maior simultaneamente, pois o servidor retorna uma resposta do pedido de escrita ao cliente antes de realmente gravar os dados no disco, acumulando-os para escrever de uma só vez.

Segurança

No NFSv2, o cliente era responsável pelo controle de acesso aos arquivos (sem nenhuma validação por parte do servidor). Isso mudou na versão 3, onde o servidor passou a tomar tal decisão, usando o mesmo esquema de segurança dos sistemas de arquivos locais Unix. Quando um cliente faz um pedido, ele envia o uid e o gid do usuário solicitante, e, através de uma consulta às permissões do arquivo local em questão, o servidor decide se libera o acesso ao cliente ou não.

Porém, isso necessita de sincronização de uid e gid entre as máquinas da rede. Para resolver isso foi criado o *Network Information Service* (NIS), um serviço de informações distribuído que é usado para fornecer tais informações a todos os nós da rede.

Percebe-se facilmente a fragilidade disso, dado que se um cliente não for confiável ele pode fornecer uids e gids falsos, podendo, assim, acessar e alterar arquivos de outros usuários. Para resolver esse problema, o NFS possui a possibilidade de autenticação mútua entre cliente e servidor, baseada no método DES de criptografia, onde as chaves são fornecidas pelo NIS. Porém, a informação trafegada não é criptografada, o que possibilita que intrusos possam obter pedaços de arquivos que trafeguem pela rede.

O Novo Protocolo NFSv4

Alguns anos após o lançamento da especificação do protocolo NFSv3, foi criada uma nova versão que revê vários conceitos que não estavam presentes nos protocolos anteriores, que causam mudanças drásticas [269] no que se conhecia até então sobre o NFS. Essa nova versão está disponível para o Linux a partir da versão 2.6 do seu kernel.

Nela, o servidor mantém o estado dos arquivos em conjunto com os clientes, diferentemente das versões anteriores. Assim, é possível que um determinado cliente pergunte ao servidor o que outros clientes estão fazendo com determinado arquivo. Isso pode indicar ao cliente se vale a pena ou não realizar um cache dos dados de forma mais agressiva.

É possível também bloquear e compartilhar partes de arquivos através do próprio protocolo NFS, sem a necessidade de servidores externos (como o NLM). O

mecanismo para isso é baseado em *leases*, ou seja, um cliente NFS pede ao servidor um contrato de bloqueio temporário (*lease*) e deve manter contato com o mesmo para continuar prolongando esse contrato conforme a necessidade.

Além disso, foi introduzido um esquema de delegação de arquivos, onde o cliente NFS pode acessar e modificar o arquivo dentro do seu cache local sem a necessidade de mandá-lo para servidor, até que o servidor contate o cliente avisando que outro cliente gostaria de acessar o arquivo, quando então este é atualizado no servidor. Isto reduz o tráfego de rede consideravelmente nos casos em que os clientes não desejam acessar um conjunto de arquivos concorrentemente.

Quanto à comunicação entre cliente e servidor, o NFSv4 usa chamadas RPC compostas, ou seja, uma mesma chamada RPC pode conter uma operação complexa envolvendo bloqueio, abertura, leitura, etc. Essas chamadas são realizadas através de conexão TCP, ao contrário das versões mais antigas, que usam UDP.

Em relação à segurança, o NFSv4 possui mecanismos sofisticados, e todas as implementações de clientes obrigatoriamente devem tê-los. Dentre esses mecanismos estão inclusos *Kerberos 5* e *SPKM3*, juntamente com o tradicional *AUTH SYS* [160]. Além disso, uma nova API foi criada para permitir estender esse mecanismo no futuro.

No NFSv4 a interpretação das listas de controle de acesso (ACLs) foi padronizada tanto para o ambiente Posix quanto para o Windows. Os nomes de usuário e grupo são armazenados em forma de texto e não mais como valores. Além desses, existe a possibilidade de se ter outros atributos, conforme a necessidade. Todos eles são armazenados na codificação UTF-8.

Por fim, todos os protocolos NFS existentes (tais como stat, NLM, mount, ACL e NFS) convergem para uma única especificação, proporcionando uma melhor compatibilidade com os firewalls de rede, além de introduzir no protocolo suporte a migração e replicação de arquivos.

Análise Crítica

O NFSv4 tornou sua manutenção e uso muito mais simples, por possuir, agora, controle de bloqueios encapsulado no mesmo protocolo e não mais através de sistemas de terceiros, além de permitir o controle da consistência dos arquivos

que estão nos caches dos seus clientes. O controle da segurança aos arquivos era muito simplificado e frágil, permitindo que clientes não confiáveis pudessem acessar arquivos de maneira desonesta. Isso foi resolvido na versão 4 do protocolo, onde mecanismos avançados de segurança e autenticação foram incorporados.

Outro problema era o grande consumo de recursos da rede nas operações entre cliente e servidor (devido à interface RPC/XDR). Associado à política de cache, o NFSv3 não é muito recomendado para aplicações que necessitam de acesso contínuo aos arquivos. O NFSv4 resolve esse problema pois é possível enviar múltiplos pedidos ao servidor através da mesma chamada RPC, além do uso do cache ter melhorado por conta do controle de estado no acesso aos arquivos.

Uma excelente característica é a transparência que o sistema de arquivos fornece ao usuário final, que nem sequer percebe estar lidando com arquivos remotos. Na versão 4, onde os controles de bloqueios e estado são nativos do protocolo, isso é ainda mais evidente, dando a impressão de que se está usando o sistema de arquivos local.

Além disso, o fato de ter sua especificação aberta para que qualquer um possa implementar seu servidor ou cliente permitiu que ele se tornasse o sistema de arquivos distribuído mais utilizado no mundo.

7.3.5 Andrew File System - AFS

Projeto:Open Andrew Filesystem

Sítio Oficial:<http://www.openafs.org>

Licença: IBM Public License Version 1.0

Responsável:

O projeto ANDREW ([245], [246]) começou na Universidade Carnegie-Mellon em 1983, com apoio da IBM. Seu objetivo era projetar e implementar um sistema distribuído para o ambiente acadêmico de ensino e pesquisa, que ofereceria a cada professor e aluno uma estação de trabalho com um sistema operacional compatível com o UNIX BSD. Além disso, a partir de qualquer um desses computadores,

o usuário deveria ter a mesma visão do sistema. Essa transparência de localização deveria ser altamente escalável, podendo aceitar de 5 a 10 mil estações de trabalho nessa rede.

Ao lado da escalabilidade, um outro problema importante que os desenvolvedores iriam enfrentar era a segurança. Com tantos clientes e usuários, era praticamente impossível confiar em todos sem provocar uma fragilidade na segurança de todo o sistema. O ANDREW sofreu modificações gradualmente durante sua existência, que foi dividida em três fases:

- **AFS-1:** Durante o ano de 1985, o AFS-1 operou 100 estações de trabalho e 6 servidores. O desempenho do sistema era razoável tendo até 20 clientes conectados por servidor, porém um trabalho pesado que algum deles realizasse poderia degradar o funcionamento do sistema como um todo de forma intolerável. Além disso, a administração do sistema era complicada, dado que os administradores não dispunham de ferramentas adequadas.
- **AFS-2:** A partir de toda a experiência adquirida com o AFS-1 e com todos os seus testes de desempenho, foi possível criar uma nova versão muito mais eficiente. Eficiência ganha com o uso de algoritmos melhores para manutenção da consistência do cache, além de uma melhoria na implementação da resolução de nomes, comunicação e estrutura dos servidores. Funcionou desde o final de 1985 até meados de 1989.

O AFS-2 [245] trouxe o conceito de *callback*, que permite ao cliente abrir e fechar um arquivo várias vezes sem precisar acessar o servidor. Quando um cliente recebe um arquivo do servidor, ele também recebe um *callback*, que é uma promessa de que ele está com a versão mais recente do arquivo, que pode ser quebrado ou quando um cliente atualiza um arquivo, ou quando o servidor recebe uma nova versão desse arquivo de um outro cliente. A cópia local pode ser utilizada quantas vezes se desejar, contanto que o cliente possua um *callback* válido.

O problema de escalabilidade foi amenizado ao se passar grande parte do trabalho dos servidores para os clientes: todas as operações de leitura e escrita são realizadas na cópia local do arquivo. Somente quando o arquivo alterado é fechado ele é então transferido de volta para o servidor. Uma consequência desta técnica é que o AFS utiliza semântica de sessão (e não a semântica UNIX de acesso concorrente a arquivos). Assim, um cliente só

perceberá a alteração de um arquivo, feita por um outro cliente, quando ele abrir o arquivo depois que o outro já o tiver fechado.

Como vários usuários passaram a depender do sistema, percebeu-se a importância da disponibilidade dos dados, já que a queda de um servidor provocava interrupção dos trabalhos por vários minutos. Assim, em 1987 iniciou-se o desenvolvimento de um sistema de arquivos de alta disponibilidade, baseado na escalabilidade e segurança do AFS, denominado CODA (mais detalhes na seção 7.3.6).

- **AFS-3:** O AFS-3, ou simplesmente AFS, foi uma iniciativa de tornar o ANDREW um sistema comercial no início de 1990. Para tanto, era necessário adotar alguns padrões, como por exemplo o *Virtual File System* (VFS) da SUN, possibilitando integrá-lo a outros sistemas de arquivos.

Foi implementado o protocolo Kerberos para autenticação mútua entre clientes e servidores, resolvendo assim o problema de segurança no acesso aos dados. A proteção dos arquivos é baseada em listas de controle de acesso, que especificam quais usuários, ou grupos, têm que tipo de acesso sobre eles.

Além disso, a partir dessa implementação os arquivos deixaram de ser cacheados em sua totalidade e passaram a ser transferidos, conforme a necessidade, em blocos de 64 Kbytes, reduzindo assim a latência da abertura e tornando possível o acesso a arquivos grandes que não cabem no disco local.

Princípios do AFS

A fim de atingir seus objetivos, foram adotadas algumas regras para o desenvolvimento do ANDREW e, conseqüentemente, do AFS:

1. Sempre que for possível deve-se realizar as operações no cliente e não no servidor, distribuindo, assim, as tarefas entre as máquinas disponíveis, evitando sobrecarregar o servidor;
2. Sempre que possível usar o cache para diminuir o tráfego dos dados e a carga dos servidores;

3. Explorar os tipos de acesso aos arquivos. Por exemplo, manter arquivos temporários na máquina local, replicar em diversos servidores arquivos executáveis que são muito usados e raramente alterados, etc;
4. Replicar serviços e informações sempre que possível, evitando limitar a escalabilidade de todo o sistema à capacidade dessa máquina central;
5. Confiar no menor número possível de entidades (segurança);
6. Agrupar o trabalho sempre que possível. Por exemplo, realizar uma leitura de 50 KB é muito mais eficiente que realizar 50 leituras de 1 KB.

Características do AFS

O espaço de nomes do AFS é dividido em duas partes: os locais, que consistem dos arquivos cacheados, temporários e daqueles necessários para a inicialização da máquina; os remotos, que são aqueles que podem ser encontrados a partir de qualquer cliente conectado na mesma rede.

Ao contrário do NFS, no AFS toda informação sobre os nomes dos arquivos e diretórios é armazenada nos servidores. Deste modo, a manutenção dos clientes é trivial e a uniformidade do espaço de nomes é uma consequência natural da configuração dos servidores. Quando um cliente precisa acessar um arquivo remoto, ele pergunta a todos os servidores por sua localização, que é então guardada em cache local para futuras consultas.

O acesso concorrente a arquivos pode ser controlado a partir de chamadas UNIX para *flock*, que administra bloqueios ao arquivo, de forma emulada. O responsável por esse bloqueio é o servidor que detém tal arquivo. Caso esse bloqueio dure mais de 30 minutos, o servidor automaticamente o libera, para evitar que a queda de um cliente impossibilite o acesso aos arquivos que ele bloqueou.

Em 1998 havia mais de 100 células AFS por todo o mundo dando a seus usuários a possibilidade de compartilhar seus arquivos através de diferentes continentes usando uma interface de sistema de arquivos parecida com a do UNIX. O AFS começou a ser comercializado pela Transarc Corporation, que foi comprada pela IBM. No momento em que esse texto foi escrito, o AFS estava na versão 3.6, sendo distribuído de forma independente do ANDREW. Para maiores informações visite: <http://www-3.ibm.com/software/stormgmt/afs/library/>.

Análise Crítica

O AFS é um sistema de arquivos distribuídos que evoluiu muito desde sua primeira versão. Pensando-se sempre em escalabilidade, transparência de localização e segurança, ele foi implementado usando conceitos simples, mas que são de extrema importância para se atingir tais objetivos.

Ele oferece um serviço altamente escalável e seguro, através da adoção de semântica de sessão no acesso concorrente a arquivos, na utilização de grandes caches no disco local do cliente e no uso de listas de controle de acesso, juntamente com o protocolo de autenticação mútua Kerberos.

Por causa do cache e da iniciativa de não se compartilhar arquivos temporários, os clientes necessitam obrigatoriamente de disco local. O espaço de nomes é dividido entre local e remoto, sendo que este último é mantido e organizado pelos servidores através de um banco de dados de localização.

7.3.6 Constant Data Availability - CODA

Projeto:CODA Filesystem

Sítio Oficial:<http://www.coda.cs.cmu.edu/doc/html/index.html>

Licença: GPL

Responsável: Carnegie Mellon University

O CODA ⁴ (*Constant Data Availability*) ([339], [245], [56], [246]) começou a ser desenvolvido em 1987 pela Universidade de Carnegie Mellon, EUA, tendo sua origem a partir do AFS-2. Seu principal objetivo é fornecer operações desconectadas ao sistema de arquivos para computadores portáteis, que costumam ficar grande parte do tempo fora da rede. Isso provê uma máxima disponibilidade dos arquivos aos seus usuários.

Para que isso seja possível, o CODA implementa alguns mecanismos de replicação não presentes no AFS, dado que ele foi criado para lidar com estações de trabalho portáteis ou que permanecem conectadas aos servidores por curtos pe-

⁴<http://www.coda.cs.cmu.edu/doc/html/index.html>

ríodos de tempo.

Replicação dos Dados.

Pelo CODA, cada volume (um conjunto de diretórios do sistema de arquivos) é associado a um *volume storage group* (VSG), que consiste de um conjunto de servidores que replicam o volume. O conjunto de servidores acessíveis de um certo grupo em um certo momento é chamado de AVSG (accessible VSG). Essa organização é melhor visualizada na figura 7.6. A coerência entre as várias cópias de um arquivo é mantida por um sistema parecido com o de *callbacks* do AFS.

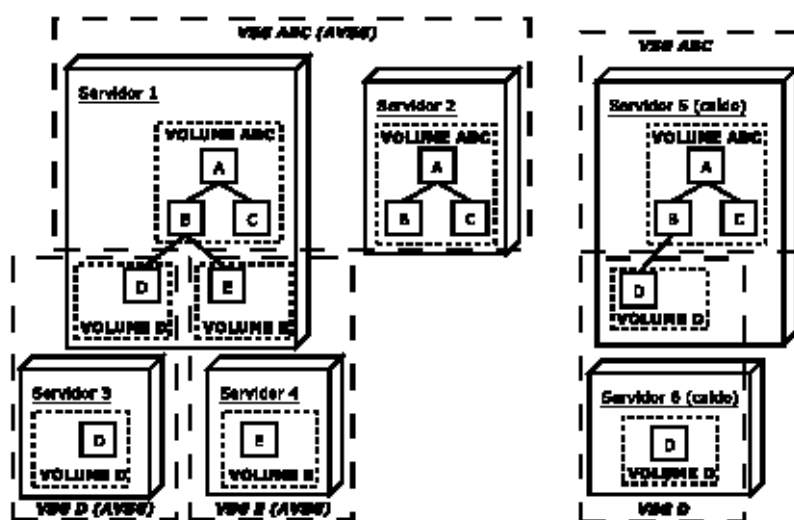


Figura 7.6: Volumes, VSGs e AVSGs

Quando um cliente envia uma atualização de um arquivo para o servidor, a atualização é enviada para todos os servidores AVSG usando um mecanismo denominado *multiRPC*. Além disso, são enviadas mensagens aos clientes quebrando o *callback* que eles possuem para aquele arquivo, invalidando o cache do mesmo.

Se um servidor que estava caído volta à rede, nada é feito inicialmente para atualizar seus arquivos. Porém, sempre que um cliente envia uma requisição para abrir um arquivo para o seu servidor preferido, ele também pede a todos os servidores AVSG que enviem a versão daquele arquivo que eles detêm. Assim, o cliente pode descobrir se existe algum servidor com uma cópia desatualizada, avisando-o para atualizar esse arquivo. Dessa forma, quem toma as iniciativas

para atualização dos arquivos que possuem réplicas inconsistentes são os próprios clientes.

Essa replicação aumenta a disponibilidade dos arquivos, o que aumenta a segurança para os clientes encontrarem o que procuram e guardarem os dados que possuem. Por exemplo, se um computador portátil perder todos seus dados, a chance de recuperá-los com a replicação é maior. Além disso, o espaço em disco nos servidores tende a ser maior que nos clientes, facilitando ainda mais o uso dessa característica.

Controle de Consistência

O CODA tenta prover ao máximo as operações desconectadas. Para isso, ele permite que os clientes possam ler e escrever seus arquivos de forma indiscriminada, a partir de qualquer servidor da rede que possua os dados que ele precise, mesmo que a rede seja particionada devido à queda de algum servidor ou conexão entre eles. Isso pode gerar perda de informação e acesso a dados inconsistentes quando, por exemplo, dois usuários alteram o mesmo arquivo em partições diferentes.

Operações Off-Line

A parte mais interessante do CODA é a possibilidade de acessar um sistema de arquivos distribuído estando completamente desconectado da rede. Se um arquivo está armazenado localmente na máquina, o usuário pode ler e escrever no arquivo sem a prévia permissão do servidor.

Isso só é possível graças a um software chamado venus⁵, que é o responsável pelo sistema de arquivos do lado do cliente. Ele possui três estados de funcionamento:

- **Cacheando:** Esse é seu estado normal de funcionamento. Aqui a comunicação com os servidores é possível sempre que necessário, mas o cliente

⁵<http://www.coda.cs.cmu.edu/doc/html/kernel-venus-protocol.html>

procura estar preparado para o caso de uma desconexão da rede, seja voluntária ou não;

- **Emulação:** Esse estado é atingido quando o cliente perde a conexão com os servidores. Nesse caso, o venus tenta fazer o papel dos servidores, disponibilizando as réplicas dos arquivos gravadas localmente, como se ainda estivessem sendo acessados através dos servidores;
- **Reintegração:** Assim que o computador é conectado à rede, entra-se no modo de reintegração, onde ele passa a fornecer aos servidores responsáveis, os arquivos em seu cache que sofreram alterações. Após o final dessa operação, volta-se ao primeiro estado.

Desempenho

Alguns testes [325] realizados em situações normais de uso mostraram que o tamanho do cache local necessário para uma semana desconectado e o tempo de reintegração dos dados após esse mesmo período não são muito grandes.

Além disso, concluiu-se que os problemas de acesso concorrente, que poderiam causar conflitos na reintegração, são raros, dado que 99% das alterações dos arquivos são realizadas pelo mesmo usuário que já o alterou anteriormente.

O desempenho com 4 servidores replicados do CODA foi no máximo 5% pior que o do AFS, este sem replicação. Porém, o CODA se mostrou menos escalável que o AFS nesses testes [325].

Análise Crítica

O CODA apresenta inovações que auxiliam usuários que necessitam de um sistema de arquivos distribuído de alta disponibilidade. Por exemplo, ele permite que um usuário defina os arquivos que devem estar acessíveis a todo momento, dando assim a facilidade de se conectar à rede por alguns instantes, atualizar seus arquivos e os da rede e voltar a se desconectar, para ir trabalhar em casa como se estivesse conectado.

A replicação dos dados permite aumentar ainda mais essa disponibilidade e a segurança dos dados, já que não só os servidores possuem os arquivos, mas também os clientes. O problema é que isso diminui as garantias de consistência dos arquivos em caso de acesso concorrente.

O CODA não respeita a semântica de sessão (ao contrário do AFS), dado que alterações realizadas por clientes desconectados são aceitas pelo sistema, mas não são informadas a outros usuários. Isso é tolerável, considerando o ganho extra de disponibilidade no sistema de arquivos.

7.3.7 Lustre

Projeto: Lustre

Sítio Oficial: <http://www.lustre.org/>

Licença: GPL

Responsável(eis): Cluster File Systems, Inc

Lustre é um sistema de arquivos distribuídos de código aberto, largamente utilizado em cluster de grande porte. O projeto tenta prover um sistemas de arquivos para um cluster de dezenas de milhares de nós e pentabytes de capacidade de armazenamento, sem comprometer a estabilidade e a segurança.

Cada arquivo armazenado em um sistema de arquivos Lustre é considerado um objeto. Lustre apresenta a todos os clientes uma semântica POSIX padrão e acesso de leitura e escrita concorrente aos objetos compartilhados. Um sistema de arquivos Lustre tem quatro unidades funcionais: um "Servidor de Meta dados" (MDS) para armazenar os meta dados; um Armazenador de Alvos de Objeto (OST) para armazenar os dados atuais; um Servidor de Objetos Armazenados (OSS) para administrar o OSTs e cliente(s) para acessar e o usar os dados. OSTs são baseados em dispositivos de blocos. Um MDS, OSS, e um OST podem estar no mesmo nó ou em nós diferentes. Lustre não fala diretamente e não administra OSTs, ele apenas delega esta responsabilidade a OSSs para assegurar escalabilidade a grandes clusters e supercomputadores.

7.4 Sistemas de Arquivos Paralelos

Sistemas de arquivos paralelos são SADs projetados para proporcionar alto desempenho sob grande demanda e concorrência de acesso. Como essa não é uma tarefa fácil, os projetistas acabam não se preocupando tanto com a transparência no acesso, a segurança ou mesmo a qualidade do serviço. Porém, isso vem mudando.

A seguir, realizamos uma resenha de alguns SADs que têm foco em desempenho, deixando um pouco de lado praticidade ou segurança, sendo muito usados por aplicações paralelas.

7.4.1 Parallel Virtual Filesystem Version 1 - PVFS

Projeto: Parallel Virtual Filesystem Version 1

Sítio Oficial: <http://www.parl.clemson.edu/pvfs/>

Licença: GPL/LGPL

Responsável(eis): Argonne National Laboratory e Clemson University

Atualmente os aglomerados de PCs têm se tornado cada vez mais populares para aplicações paralelas. Com isso, a demanda por software para esse tipo de plataforma tem crescido muito. Hoje em dia encontra-se todo tipo de software para o ambiente de computação paralela, como sistemas operacionais confiáveis, sistemas de armazenamento de dados local e sistemas de envio de mensagens.

O *Parallel Virtual File System* ([105], [209]) se encaixa na área de sistemas de arquivos paralelo, pois é um sistema de arquivos distribuído desenvolvido para prover alto desempenho e escalabilidade paralela para aglomerados de PCs linux. Em geral, o PVFS promete 4 características:

- Um espaço de nomes consistente para todo o aglomerado;
- Acesso transparente para programas e aplicações já existentes, sem a necessidade de recompilá-los;

- Distribuição física de dados em múltiplos discos e múltiplos nós;
- Alto desempenho no acesso em modo usuário.

Para que um sistema de arquivos paralelo possa ser usado de maneira fácil, ele deve prover um espaço de nomes único em todo o aglomerado e deve ser possível acessá-lo através de utilitários comuns.

Para prover acesso de alto desempenho para os clientes do aglomerado, os dados armazenados no PVFS estão distribuídos entre os vários nós que compõe o aglomerado, assim como o **BRIDGE** faz, porém usando algoritmos de distribuição diferentes. Cada um desses nós é chamado de **I/O node**.

Dessa forma, para se obter os dados de um determinado arquivo, é necessário acessar várias máquinas, utilizando-se, assim, de vários caminhos pela rede para chegar aos respectivos discos em que estão armazenados. Isso elimina o gargalo da transferência de dados que se tem quando toda a informação está em uma só máquina, distribuindo a carga e aumentando o potencial total da banda para múltiplos clientes.

Usar mecanismos tradicionais de chamadas de sistema para acesso a arquivos pode ser conveniente, mas pode causar uma sobrecarga muito grande para o sistema como um todo, especialmente o *kernel*. Assim, é possível acessar os arquivos do PVFS usando uma API (*Application Programming Interface*) disponibilizada como biblioteca, que contém operações comuns, além de outras específicas do PVFS, que contactam diretamente os servidores, evitando acessos ao *kernel* local. Essas bibliotecas, como a *ROMIO MPI-IO* [358], podem ser usadas pelas aplicações ou por outras bibliotecas para acesso de alta velocidade ao PVFS.

Os componentes do PVFS

O **servidor de meta-dados(MGR** na figura 7.7) é um programa que gerencia todos os dados que constituem informações sobre o arquivo (exceto seu conteúdo), como seu nome, sua localização na hierarquia de diretórios, seu dono, seus atributos e como seus dados estão distribuídos entre os vários nós de dados do sistema. Esse programa realiza todas as operações sobre os meta-dados dos arqui-

vos atomicamente, evitando assim ter que implementar esquemas complexos de concorrência, *locks*, consistência, etc, para múltiplos acessos simultâneos.

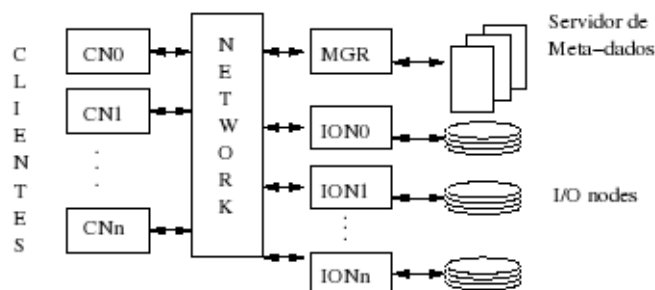


Figura 7.7: Visão Geral do PVFS

O **servidor de dados (ION na figura 7.7)** gerencia o armazenamento do conteúdo dos arquivos, bem como a recuperação dos mesmos, nos discos locais conectados nos nós. Esse servidor grava os dados dos arquivos do PVFS em um sistema de arquivos local, através de chamadas das funções tradicionais, como *read()*, *write()* e *mmap()*, para acessá-los.

Isso significa que pode-se usar qualquer tipo de sistema de arquivos local, como Ext2, Ext3 ou Reiser [371], por exemplo. Adicionalmente é possível usar suporte a RAID para que cada nó possua tolerância a falhas de disco de forma transparente e confiável para todo o sistema.

servidores do PVFS não possuem estado, da mesma forma que o NFS, o que simplifica sua implementação, que não considera casos como quando um cliente se desconecta da rede sem aviso prévio. Isso pode gerar problemas de consistência, pois o servidor pode não conter a versão mais recente do arquivo (caso o cliente possuísse um cache sujo), ou algum arquivo pode ficar bloqueado para escrita.

A **API nativa do PVFS** possibilita acesso em modo usuário aos servidores do PVFS. Esta biblioteca, chamada de *libpvfs*, cuida das operações necessárias para mover dados entre os clientes e servidores, mantendo-as transparentes para o usuário. Para operações que necessitam de meta-dados, a biblioteca se comunica com o servidor de meta-dados, conforme figura 7.8(a). Para acesso aos dados dos arquivos, o servidor de meta-dados é deixado de lado e os servidores de dados são acessados diretamente, conforme figura 7.8(b). Essa é a chave para se obter

um alto desempenho agregado no acesso aos dados.

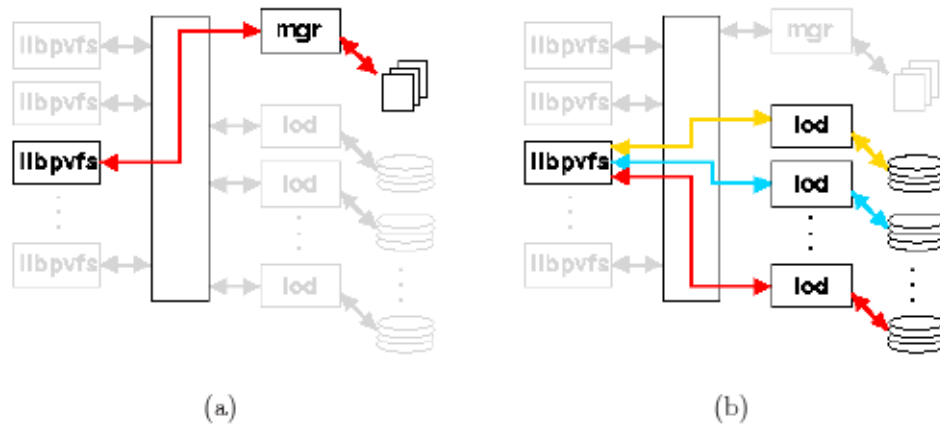


Figura 7.8: Clientes acessando o PVFS

O suporte no kernel do linux para o PVFS provê as funcionalidades necessárias para se usar comando *mount* nos clientes. Isso permite acesso aos arquivos do PVFS sem necessidade de alteração das aplicações ou programas já existentes. Esse suporte não é necessário para se usar o PVFS, mas ele traz uma enorme conveniência para a interatividade com o sistema. Para isso, é necessário instalar um módulo no *kernel* do linux (existe um patch para carregá-lo diretamente no *kernel*) e um Figura 7.9: Fluxo de dados pelo *kernel* programa chamado (*pvfsd*) que se encarrega de buscar os dados para as aplicações. Ele se utiliza da biblioteca *libpvfs* para realizar essas operações. A figura 7.9 mostra como o fluxo de dados passa por esses componentes.

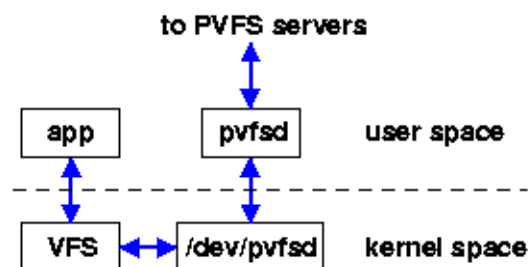


Figura 7.9: Fluxo de dados pelo kernel

Além disso, existe a implementação da interface *ROMIO MPI-IO* para o PVFS, possibilitando que aplicações paralelas se utilizem do PVFS sem precisar passar pelo *kernel*, além de poderem usar outras funções específicas desse sistema que possibilitam um ajuste fino no acesso e armazenamento dos arquivos.

Análise Crítica

O PVFS é um sistema de arquivos distribuído e paralelo que se preocupa em diminuir o gargalo provocado pelo tráfego de dados, seja pela rede, seja pela velocidade do armazenamento físico, usando a mesma técnica de distribuição de dados encontrada no BRIDGE.

Alguns problemas existentes são quanto à segurança no acesso dos dados, já que se o cliente souber onde os dados estão, basta pedi-los para os nós de dados que eles responderão, sem se preocupar com nenhum tipo de validação de permissão de acesso. Quem cuida da permissão é o servidor de meta-dados, sendo que esse mecanismo não é muito sofisticado. Outro problema existente é quando o servidor de meta-dados fica indisponível. Somente os arquivos já abertos continuarão sendo acessados, até serem fechados. Todos os outros arquivos do sistema não poderão ser acessados. Além disso, o servidor de meta-dados pode representar um gargalo no sistema, já que ele é único.

É um sistema de arquivos paralelo que já apresenta bons resultados, mesmo tendo problemas visíveis. Para aplicações paralelas e confiáveis, em uma rede privativa e fechada, ele pode ser usado sem grandes problemas de segurança

7.4.2 Parallel Virtual Filesystem Version 2 - PVFS2

Projeto: Parallel Virtual Filesystem Version 2

Sítio Oficial: <http://www.pvfs.org>

Licença: GPL/LGPL

Responsável(eis): Argonne National Laboratory e Clemson University

PVFS2 é uma reimplementação das melhores características da primeira versão

do PVFS, usando uma nova arquitetura para torná-lo mais flexível. Isso possibilitou a implementação de novas características, técnicas e inovações que foram sendo discutidas e requisitadas durante as correções de defeitos da primeira versão.

As novidades ([313], [352]) implementadas (ou ainda a serem implementadas) no PVFS2 e sua nova arquitetura estão detalhadas nas próximas seções. No capítulo 5 há mais detalhes sobre o desempenho do PVFS2.

Novas características

Suporte modular para múltiplos protocolos de rede e armazenamento

O PVFS1 foi desenvolvido com a idéia de que seus dados seriam acessados via soquete e armazenados em sistemas de arquivos locais. Analisando os aglomerados de computadores existentes hoje, nota-se que existem muitas tecnologias diferentes em cada um deles, sendo que algumas são mais populares que outras. O mesmo ocorre com os sistemas de armazenamento de dados locais. Dessa forma, o PVFS2 foi projetado usando o BMI [214] (*Buffered Messaging Interface*) como interface de acesso à rede e o Trove como interface de acesso ao sistema de armazenamento físico. O objetivo é abstrair do projeto os detalhes do mecanismo de transmissões e armazenamento. Isso permite que um desenvolvedor personalize módulos específicos para seu ambiente, sem ter que alterar o núcleo do PVFS2.

Acesso a dados estruturados não-contínuos

Muitas aplicações científicas possuem estruturas de dados complexas que nem sempre podem ser armazenadas de forma contínua, pois isto certamente impacta o desempenho da aplicação como um todo.

O Trove é uma interface desenvolvida pela equipe do PVFS2, sendo que até agosto de 2005 não havia um documento público descrevendo-a, a não ser pelo seu próprio código-fonte.

Assim como o PVFS1, o PVFS2 dá suporte ao *ROMIO MPI-IO*, que permite ao

desenvolvedor da aplicação descrever seus dados usando tipos MPI, melhorando o desempenho na leitura dos dados persistidos.

Distribuição de dados flexível

No PVFS1 os dados são distribuídos entre os servidores de dados usando o algoritmo round-robin, isto é, um arquivo é dividido em blocos de igual tamanho e cada bloco subsequente é armazenado no próximo servidor, sendo que ao chegar no ultimo servidor volta-se para o primeiro, até que todos os blocos estejam armazenados. Isso é muito eficiente como uma técnica genérica para quando não se conhece o padrão de acesso ao arquivo. Porém, em geral sabe-se qual é o padrão de acesso de um arquivo e isso poderia ser usado para otimizar o acesso a ele. O PVFS2 permite que se informe esse padrão de acesso e decide qual a melhor forma de armazenar os dados para máxima eficiência, podendo até mesmo utilizar-se de redundância.

Servidores de meta-dados distribuídos

No PVFS1 o servidor de meta-dados (que armazena informações sobre estrutura de diretórios, data de criação de arquivos, etc) é centralizado, podendo representar um gargalo maior conforme o número de clientes aumenta.

O PVFS2 permite ter mais de um servidor de meta-dados, que pode ou não ser um subconjunto dos servidores de dados.

Suporte explícito à concorrência

Um sistema de arquivos paralelo deve ser extremamente eficiente quanto a prover dados para vários clientes simultaneamente.

O projeto do servidor e cliente PVFS2 foi baseado em uma máquina de estados que está intimamente ligada a um componente de monitoramento da finalização das operações em todos os sistemas envolvidos. Isto é, permite-se que se realize acesso sem bloqueios a todos os tipos de dispositivos. Isso dá suporte a operações assíncronas nativamente, facilitando a implementação do *ROMIO MPI-IO*.

Semânticas de consistência ajustáveis

Muitos sistemas de arquivos distribuídos implementam as semânticas POSIX, que são muito estritas. O NFS, por exemplo, não implementa essas semânticas, pois não garante que o cache de seus clientes estejam coerentes o tempo todo.

Por existirem vantagens e desvantagens em cada tipo de semântica, o PVFS2 permite que o usuário opte por uma semântica mais estrita, para permitir a implementação do *ROMIO MPI-IO*, ou mais relaxada, permitindo um uso mais amplo.

Mapeamento flexível de referências de arquivos para servidores

É possível reconfigurar os servidores de meta-dados para escolher onde armazenar um determinado arquivo. Isso é muito útil na administração do sistema de arquivos, para, por exemplo, remover um servidor ou adicionar outro. Isso também pode ser feito sem a necessidade de se desativar o sistema de arquivos.

Redundância de dados e meta-dados

O PVFS1 possui um grande problema com relação à tolerância a falhas: caso um servidor saia da rede, perde-se o acesso aos seus dados. Pode-se utilizar um sistema RAID de disco para evitar a perda dos dados, mas isto não garante tolerância à falhas.

Está sendo estudado para versões futuras do PVFS2 um sistema de redundância relaxada dos dados. A idéia é realizar uma cópia dos dados e meta-dados de um servidor em outro, utilizando-se de uma operação explícita ao cliente. Isto significa que o cliente PVFS2 teria que realizar essa cópia.

A desvantagem nisso está em realizar operações de forma atômica e em encontrar formas de se evitar uma grande perda de desempenho. A vantagem é que a operação seria otimizada, ao criar as informações redundantes em paralelo.

Arquitetura do PVFS2

Servidores

No PVFS1 cada servidor tem papel distinto: servir meta-dados ou somente da-

dos. Além disso, o servidor de meta-dados é único. No PVFS2, cada servidor pode atuar tanto como servidor de meta-dados como também de e dados. A definição do papel que cada um vai representar está no arquivo de configurações, lido durante a inicialização. Além disso, pode-se ter múltiplos servidores de meta-dados.

Redes

Como já mencionado, utilizando-se do BMI é possível que o PVFS2 se comunique por TCP/IP, InfiniBand6.6.4, Myricom6.6.4 ou qualquer outro protocolo de rede que venha a ser implementado.

Interfaces

Os clientes podem acessar o PVFS2 através de duas interfaces: UNIX nativo, representado pelo cliente do sistema operacional, ou *ROMIO MPI-IO*. Ambas as formas seguem o mesmo perfil que foi desenvolvido para o PVFS1.

Interações cliente-servidor

Durante o primeiro acesso ao PVFS2, os clientes acessam algum dos servidores para obter informações sobre a configuração do sistema de arquivos. Esse processo ocorre de forma similar ao NFS: para abrir um arquivo, o cliente pede ao servidor um *handle*. Tendo um *handle*, o cliente pode acessar qualquer trecho do arquivo, desde que tenha permissão de acesso. Quando esse *handle* expirar, o servidor avisará o cliente no momento do acesso.

Esse tipo de estratégia permite que um processo possa passar seu *handle* a outro processo, que evita uma nova busca pelo arquivo junto ao servidor. Como os clientes e servidores não possuem estado, uma desvantagem é que se um arquivo é removido, o cliente que tiver o *handle* ainda poderá acessá-lo por um tempo, até expirar. Esse tipo de problema também ocorre em sistemas de arquivos locais.

Consistências do ponto de vista do cliente

O PVFS2 permite que vários clientes realizem escritas simultâneas em regiões não-coincidentes dos arquivos, até mesmo em regiões não-contínuas, de forma atômica. Isso possibilita paralelizar a escrita sem correr o risco de se gerar incon-

sistências entre servidor e clientes.

Quanto à consistência do cache, o PVFS2 permite colocar no cache do cliente a estrutura de diretórios do servidor de meta-dados. Isso pode gerar inconsistências temporárias, pois caso haja alguma mudança em tal estrutura, o cliente ficará desatualizado por um certo tempo (configurável).

Consistência do sistema de arquivos

Ao realizar alterações na estrutura de diretórios do PVFS2, o sistema de arquivos é bloqueado enquanto essa tarefa é realizada. Foi notado que esse tipo de tarefa não representa um gargalo na maioria das aplicações, mesmo em larga escala. Porém, esses bloqueios não ocorrem em todas as operações. Por exemplo, para criar um arquivo deve-se:

1. criar uma entrada no diretório;
2. criar um objeto de meta-dados;
3. apontar a entrada no diretório para o objeto de meta-dados;
4. criar um conjunto de objetos de dados para o novo arquivo e apontá-los aos objeto de meta-dados.

Cada uma dessas operações é realizada atomicamente, mas o conjunto delas não. Isso é um problema para o PVFS2, caso a execução dessas tarefas seja interrompida.

Análise Crítica

O PVFS2 realmente evoluiu muito em comparação ao PVFS original. As novas características que estão sendo adotadas permitem que ele seja cada vez mais utilizado, o que ajuda os desenvolvedores a entender a real necessidade que os pesquisadores têm de um sistema de arquivos paralelo para suas aplicações.

A mudança na forma como o código foi implementado facilita sua evolução, atraindo desenvolvedores de plataformas específicas a criar módulos robustos

para o PVFS2, que permitem usar esse SAP em cada vez mais aglomerados de computadores.

Capítulo 8

Cluster de Aplicação

Um cluster de aplicação é um conjunto de servidores que respondem coletivamente por um serviço. Esse conjunto de servidores pode dividir entre si a carga do sistema, ou simplesmente aguardar um dos servidores falhar para assumir o serviço. Esta tecnologia tem sido muito utilizada na Internet, como em sites de portais, e-commerce, e-business, abrangendo desde situações onde é necessário tratar milhares de requisições simultâneas, até a demanda do serviço que exige 99.99% do tempo on-line por exemplo.

Clusters de aplicação, em geral, são tecnologias maduras e estáveis para serem utilizadas. E se subdividem basicamente em 2 grupos:

- Alta disponibilidade;
- Balanceamento de carga;

Neste capítulo serão descritas tecnologias que executam ou prestam suporte para a obtenção destas características.

8.1 Linux Virtual Server

Linux Virtual Server (LVS) é uma tecnologia que permite a construção de sistemas com alta disponibilidade e altamente escaláveis, a partir do uso conjunto de vários servidores. A arquitetura é totalmente transparente para o usuário final, ou seja, um grupo de máquinas servidoras com um direcionador que transparece como apenas um único ponto de acesso. O LVS pode oferecer serviços de maior capacidade/desempenho, ou serviços redundantes (quando servidores individuais tiverem que sair do ar para manutenção) em relação aos serviços disponíveis em servidores únicos (LVS-HOWTO [8]).

O LVS é como um roteador da camada 4 do modelo OSI 6.7.4. A semântica padrão do cliente-servidor continua preservada, onde cada cliente pensa que está diretamente conectado a um servidor real e este pensa que está conectado diretamente a um cliente. Nem o cliente nem o servidor sabem que as conexões sofrem a intervenção do direcionador. Um servidor real do LVS não coopera e nem sabe da existência de outros servidores reais no LVS, um LVS não é um beowulf (vide 10.1) e também não é um Cluster, mas se comporta como um agregador de serviços que possibilita o atendimento de uma demanda grande em um serviço (LVS-HOWTO [8]).

O código IPVS, que possibilita a construção do sistema LVS, é uma coleção de modificações para kernel Linux, que combinadas com as capacidades de roteamento e filtragem de pacotes de uma máquina Linux, transformam-se em um roteador com características especiais, capaz de balancear sessões TCP e UDP entre várias máquinas. Este roteador especial, chamado Linux Director (ou ainda load balancer ou simplesmente Director) distribui a carga de requisições de serviços entre as máquinas que os provêem. Com isso, o sistema constituído pela máquina Linux com o código IPVS e as outras máquinas que hospedam os serviços é chamado Linux Virtual Server (LVS), vide figura 8.1.

O sistema LVS não necessita de nenhuma modificação nos Servidores Reais (que podem estar interconectados na mesma LAN ou geograficamente dispersos em uma WAN) ou nos clientes, este por sua vez, enviam suas requisições apenas para uma máquina (Director) não importando quantos Servidores Reais estejam provendo os serviços, que podem ser os comuns HTTP e HTTPS, servidores de email, bancos de dados, CVS, SSH (em geral todas as aplicações TCP/IP usu-

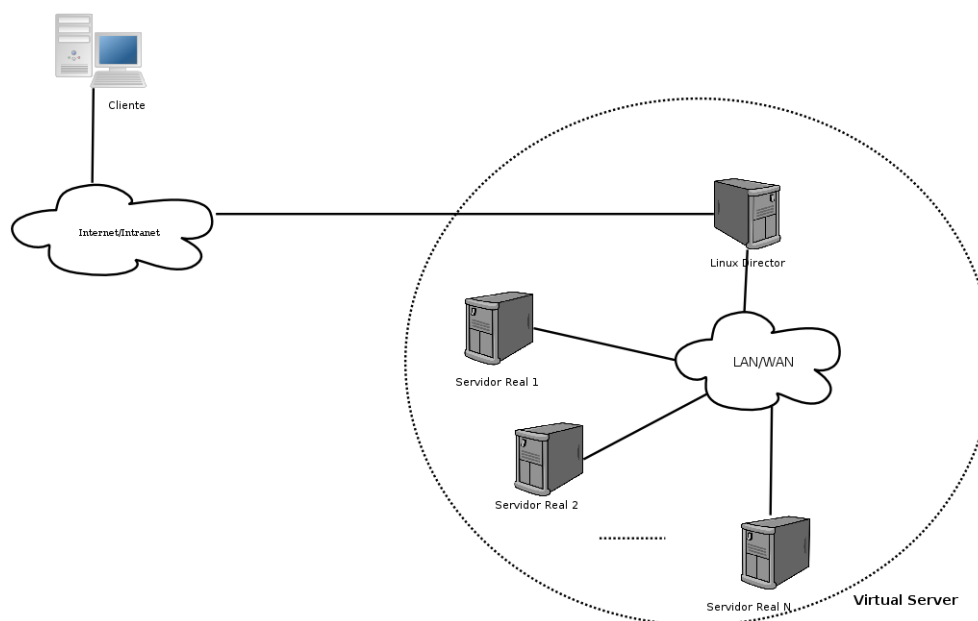


Figura 8.1: Esquema geral de um Linux Virtual Server

fruem desse sistema).

O LVS provê, de maneira transparente e simples, um ambiente altamente escalável de alta disponibilidade. Seu ponto único de falha (ponto crítico) é o Director, mas mesmo assim, em conjunção com outras ferramentas (como Heartbeat e LDirector) pode-se construir um sistema de alta-disponibilidade para este item, aumentando ainda mais sua confiabilidade e eliminando seu ponto crítico de falha.

Mais informações e documentações detalhadas sobre o LVS podem ser obtidas nos seguintes endereços:

<http://www.linuxvirtualserver.org>

<http://www.ultramonkey.org>

<http://www.austintek.com/LVS/LVS-HOWTO/>

8.1.1 Nomenclatura e abreviações

Em textos sobre LVS, tornou-se comum o uso de termos para designar os componentes do sistema:

- . **LVS, Linux Virtual Server** - designa a combinação Director+Servidores Reais, que juntos compõem o Servidor Virtual, sendo visto pelos clientes como uma única máquina.
- . **Director** - a máquina que roda o código **ipvs**. É um roteador com regras especiais que recebe requisições de serviços de clientes e as repassa para máquinas que disponibilizam os serviços.
- . **Servidor Real** - a máquina que hospeda os serviços, é quem de fato trata requisições de clientes.
- . **Métodos de Redirecionamento (LVS-Nat,LVS-DR,LVS-Tun)** - Sendo o Director um roteador com regras específicas de redirecionamento, estes métodos determinam como os pacotes dos clientes são redirecionados para os Servidores Reais.
- . **Métodos de escalonamento (scheduling)** - algoritmos usados pelo Director para selecionar o Servidor Real que vai tratar uma nova conexão estabelecida por um cliente.
- . **VIP (Virtual IP)** - endereço IP usado pelo Director para fornecer os serviços aos clientes.
- . **RIP (Real IP)** - designa os endereços IP usados pelos Servidores Reais.
- . **DIP (Director's IP)** - endereço IP usado pelo Director para se comunicar com os Servidores Reais.

8.1.2 Tipos de LVS Cluster

Os sistemas montados com o uso de LVS são, normalmente, descritos pelo tipo de método de redirecionamento das requisições para os nós do cluster. Há três métodos disponíveis:

- . LVS-NAT (Network address translation)
- . LVS-DR (Direct Routing)
- . LVS-TUN (IP tunneling)

Mais de um método pode ser usado em um único Director, tendo por base as características dos nós do cluster. O método mais simples de se implementar é o LVS-NAT.

Network Address Translation (LVS-NAT)

Em uma configuração LVS-NAT, o Director usa a habilidade do kernel Linux de mudar o endereço IP e porta dos pacotes que passam por ele. Neste método, o Director recebe uma requisição de um cliente e a repassa para um Servidor Real, que a processa, enviando o resultado de volta para o Director, que então faz as mudanças necessárias para converter o IP dos pacotes no endereço de servidor virtual, dando resposta ao cliente, e passando a impressão que está tratando apenas com uma máquina, conforme figura 8.2.

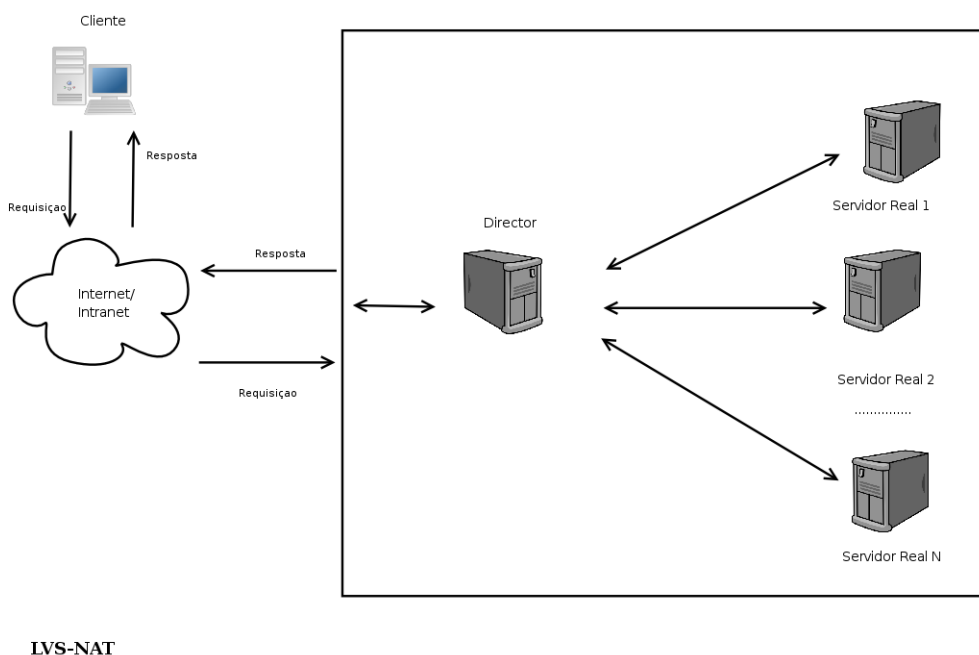


Figura 8.2: LVS-NAT

Propriedades de LVS-NAT

- . Os Servidores Reais devem estar na mesma subrede do Director,

- . Os endereços dos nós (Servidores Reais) normalmente estão em conformidade com RFC 1918,
- . As conexões (de entrada e saída) passam todas pelo Director,
- . O Director deve ser o gateway padrão dos Servidores Reais,
- . O Director pode remapear números de portas, isto é, uma requisição recebida em uma porta dele e pode ser redirecionada para uma porta diferente de um Servidor Real,
- . Qualquer sistema operacional pode ser usado nos Servidores Reais,
- . O gargalo do ambiente pode ser um único Director configurado para atender a demanda, embora uma rede saturada normalmente seja o problema mais comum.

Direct Routing (LVS-DR)

Neste modelo, baseado no NetDispatcher¹, o Director repassa as conexões para os Servidores Reais e estes respondem diretamente para os clientes que fizeram as requisições. Para isto os Servidores Reais deverão estar no mesmo segmento de rede que o Director, assim como todas as máquinas (Directors e Servidores Reais) que usam o mesmo VIP.

Todos os Servidores Reais possuem uma interface virtual de loopback(lo:0) configurada com o VIP que não responde a requisições ARP, redirecionando as conexões que receber para uma porta local e respondendo diretamente ao cliente, o que implica que o Director não necessita estar no caminho de volta.

Os Servidores Reais podem ser acessados de fora da rede, caso haja falha no balanceador de carga. No caso de ser usado apenas um Director e não houver outro que atue como backup, os servidores reais podem ser acessados de fora da rede diretamente.

¹Software de roteamento da IBM usado para balancear carga entre servidores TCP, mais informações podem ser obtidas em: <http://www.cs.princeton.edu/courses/archive/fall03/cs518/papers/networkdispatch.pdf>

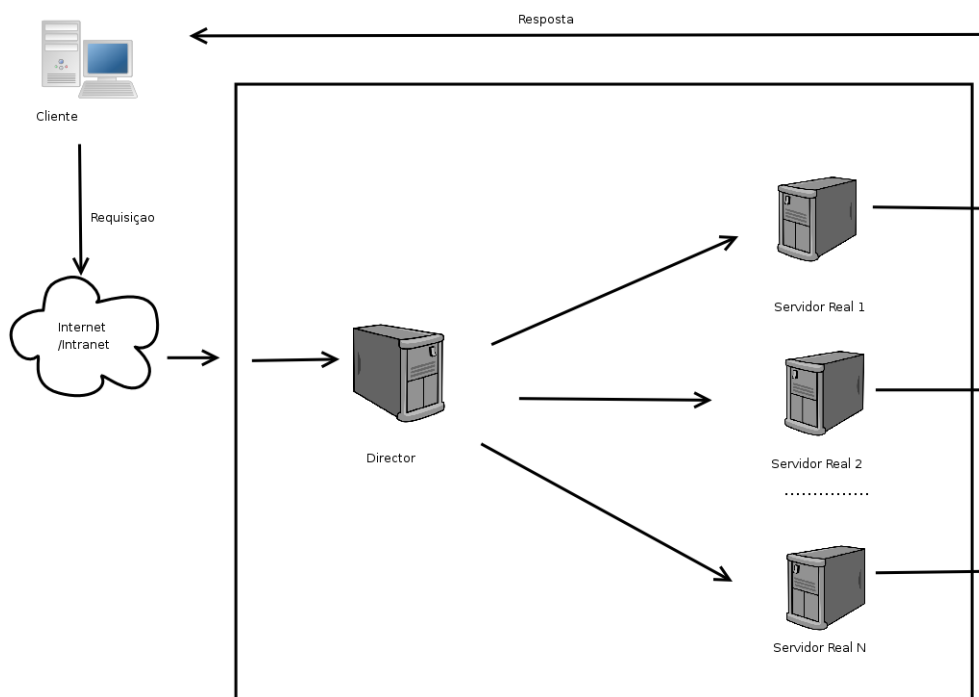


Figura 8.3: LVS-DR

Características do LVS-DR

- . Os Servidores Reais devem estar na mesma rede que o Director,
- . Os RIP não necessitam estar em conformidade com a RFC 1918,
- . Somente as requisições passam pelo Director, as respostas são enviadas diretamente aos clientes pelos Servidores Reais,
- . As portas não podem ser remapeadas no Director,
- . LVS-DR permite mais Servidores Reais que LVS-NAT,
- . Não há sobrecarga no Director como no LVS-NAT.

Encapsulação IP-IP(Tunneling)(LVS-Tun)

IP tunneling (RFC 2003) é uma técnica que permite que pacotes IP sejam colocados dentro de outros pacotes IP, permitindo que pacotes destinados a um determinado endereço IP sejam redirecionados para outro endereço IP. Neste método de configuração de LVS o Director e os Servidores Reais não necessitam estar no

mesmo segmento de rede, mesmo estando geograficamente distantes (como no caso de mirrors de ftp). Dessa forma, os Servidores Reais podem usar qualquer endereço de rede e não apenas endereços privado.

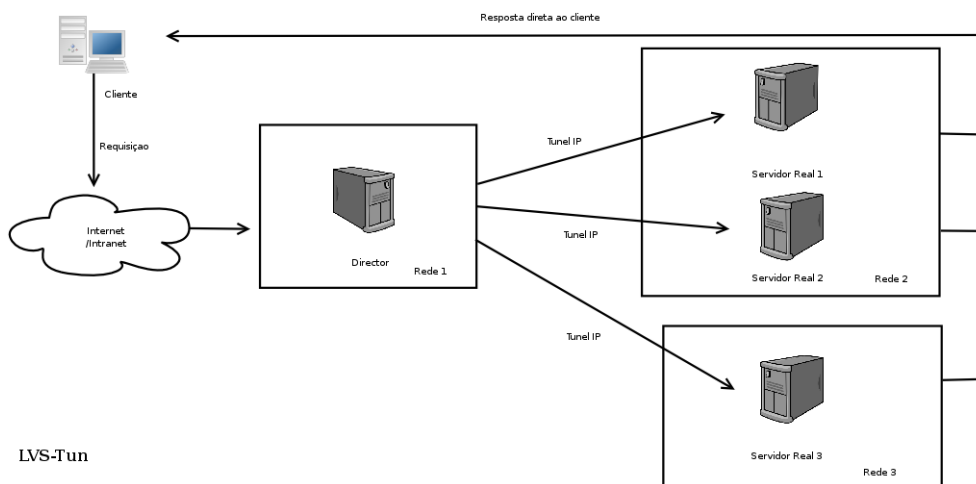


Figura 8.4: LVS-Tun

Características do LVS-Tun

- . Os Servidores Reais não necessitam estar no mesmo segmento de rede que o Director,
- . Os RIP não necessitam estar de acordo com a RFC 1918,
- . O Director apenas recebe requisição dos clientes, as respostas são enviadas diretamente dos Servidores Reais,
- . O Director não pode remapear portas,
- . Os sistemas operacionais dos Servidores Reais precisam suportar IP tunneling.

8.1.3 Algoritmos de escalonamento

Existem vários algoritmos utilizados para a implementação do LVS e seus métodos de escalonamento para o balanceamento de carga. Os métodos de escalonamento regulam a maneira como a carga é distribuída entre os nós que compõem

o sistema. Quando o Director recebe uma requisição de um cliente, é através dos algoritmos de escalonamento que ele decide qual nó deverá tratá-la.

Existem métodos de escalonamento dinâmico que dão maior controle sobre a carga de chegada, com pouco ou nenhum custo adicional em processamento. O Director mantém uma lista do número de conexões ativas e inativas para cada nó do cluster e usa esta informação para determinar qual nó irá enviar uma nova conexão.

Os métodos mais utilizados são discutidos a seguir.

Round-Robin (RR)

O Director mantém uma lista com os endereços de cada Servidor Real, assim que recebe uma conexão, ele a redireciona para um servidor dessa lista, onde uma próxima conexão será enviada para o servidor seguinte e assim continua percorrendo essa lista de forma circular atendendo todas as requisições. Todos os servidores da lista são tratados de igual maneira, não importando quantas conexões estão sendo manipuladas por um servidor específico, nem seu tempo de resposta e/ou capacidade de processamento.

Round-Robin com pesos (WRR)

Cada nó do sistema LVS possui um peso (inteiro associado à sua capacidade de processamento e atribuído pelo administrador do ambiente), baseado na quantidade de carga que ele pode manipular (capacidade de processamento). O peso é então usado, em conjunção com o método round robin, para selecionar o próximo nó que será usado quando uma nova conexão chegar, sem levar em consideração o número de conexões que ainda estão ativas. Servidores Reais com pesos maiores terão prioridade no recebimento e quantidade de requisições, se comparados com Servidores Reais com pesos menores.

Destination hash (DH)

Neste método, o Director sempre envia requisições de um mesmo endereço IP de origem para o mesmo Servidor Real no sistema LVS, usando uma lista estática de endereços de destino. O método é útil quando o Servidor Real é um servidor proxy ou cache.

Least-Connection (LC)

Com este método, quando uma nova conexão chega, o Director verifica o número de conexões ativas e inativas para determinar para qual nó irá enviar a requisição.

Para realizar esta escolha, o Director multiplica o número de conexões ativas do nó por 256 (atribuição interna do algoritmo em sua implementação) e adiciona ao resultado o número de conexões inativas resultando num overhead² para cada nó. O nó com menor overhead receberá a conexão. Caso haja nós com mesmo overhead, o primeiro nó encontrado na tabela do IPVS será selecionado.

Weighted Least-Connection (WLC)

Este método combina o Least-Connection com um peso para cada nó (este é o método padrão se nenhum for selecionado), útil para ser usado com nós de diferentes capacidades de processamento.

O Director determina para qual nó uma requisição será enviada, calculando o overhead, como no método anterior, dividindo este número pelo peso associado ao nó. O nó com menor valor associado após esta operação receberá a conexão. Caso haja nós com mesmo valor associado, o primeiro da tabela do IPVS será selecionado.

²Métrica definida para o algoritmo utilizada para realizar a distribuição da carga.

Shortest Expected Delay (SED)

Este método pode oferecer uma sensível melhoria em relação ao método WLC em serviços que usam TCP e mantêm a conexão ativa enquanto o nó processa a requisição.

O cálculo do valor do overhead para o SED é feito adicionando-se 1 ao número de conexões ativas, dividido pelo peso associado a cada nó. O nó com menor overhead recebe a requisição.

Deve-se notar o seguinte neste algoritmo:

- . Ele não usa o número de conexões inativas enquanto determina o overhead para cada nó.
- . Adiciona 1 ao número de conexões ativas para antecipar como o overhead irá parecer quando uma nova conexão for permitida.

O algoritmo SED tenta minimizar o tempo de espera para cada trabalho até sua finalização. O tempo de espera é $(C_i + 1)/U_i$, sendo C_i o número de conexões do servidor e U_i o peso fixado para este servidor.

A diferença entre SED e WLC é que SED inclui a conexão que chega na função de custo, incrementado 1. Ele apresenta melhor qualidade em grandes sistemas heterogêneos, cujos pesos variam muito.

Never Queue (NQ)

Este método apresenta uma melhoria em relação ao SED, pois caso um nó não possua conexões ativas, ele receberá uma nova requisição de serviço, apesar do resultado apresentado no cálculo do SED, já que podem ocorrer situações que uma máquina que não possua nenhuma conexão ativa apresente um overhead maior que outra que possua.

Locality-Based Least-Connection (LBLC)

Directors também podem direcionar o tráfego de saída para o conjunto de servidores proxy transparentes. Nesta configuração, os nós do cluster são proxy transparentes ou servidores de web cache que estão entre os clientes e a internet.

Quando o LBCL é usado, o Director tenta enviar todas as conexões de um endereço IP particular para o mesmo servidor proxy transparente (nó do cluster). Ou seja, a primeira vez que uma requisição chegar, o Director irá escolher um Servidor Real para atendê-la usando um versão um pouco modificada do método WLC e todas as requisições subsequentes deste cliente continuarão a ser enviadas para o servidor escolhido.

Locality-Based Least-Connection with Replication Scheduling (LBLCR)

É semelhante ao método anterior com uma melhoria, o Director mantém um mapeamento de um cliente para um conjunto de servidores que podem atendê-lo. Se todos os nós do conjunto estiverem sobrecarregados, o Director apanha um servidor com menos conexões e o adiciona ao conjunto. Caso este conjunto não se modificar em um intervalo de tempo específico (seis minutos, intervalo padrão como definido do código do IPVS), o servidor com maior carga será excluído.

8.1.4 Casos de uso de LVS

Muitas empresas utilizam o LVS para suprir a demanda por uma grande capacidade de processamento de requisições e para poder dividir/balancear a carga de seus sistemas por outras localidades (máquinas remotas), melhorando assim o atendimento das demandas de acesso a seus sistemas e sítios WEB.

Alguns exemplos de sítios e empresas que utilizam a tecnologia são listados abaixo. Mais casos de uso podem ser encontrados em <http://www.linuxvirtualserver.org/deployment.html>.

Sítio	Forma de utilização
http://linux.com	Balanceamento de carga HTTP
http://sourceforge.net	Balanceamento de carga HTTP, HTTPS, FTP, SSH, CVS
http://themes.org	Balanceamento de carga HTTP
http://wwwcache.ja.net	40 servidores Squid em 3 clusters LVS
http://www.zope.org	Balanceamento de carga HTTP
www.songn.com	Um Director rodando em modo VS/DR com mais de 6 nós de servidores Windows 2000

Tabela 8.1: Exemplos de Sítios que utilizam LVS

8.2 Cluster Tomcat

O Tomcat [188] é um servidor de aplicações Java, Java Servlet e JavaServer Pages (JSP). O objetivo de um servidor de aplicações é disponibilizar uma plataforma abstraindo do desenvolvedor de software algumas das complexidades de um sistema computacional. O servidor de aplicações responde a questões comuns à todas as aplicações, como segurança, alta disponibilidade, balanceamento de carga e tolerância à falhas.

Ele é distribuído como software livre e desenvolvido dentro do projeto Apache Jakarta, que é oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias Java Servlet e JavaServer Pages (JSP). O Tomcat é, suficiente, robusto e eficiente o para ser utilizado em ambientes de produção.

Tecnicamente o Tomcat é um container Web, cobrindo parte da especificação J2EE e servindo de container para tecnologias como Servlet e JSP, e tecnologias de apoio como JNDI Resources e JDBC DataSources. O Tomcat tem a capacidade de atuar também como servidor web/HTTP, ou pode funcionar integrado a um servidor Web dedicado, como o Apache httpd ou o Microsoft IIS.

A partir da versão 5, Tomcat passou a dispor de escalabilidade horizontal (capacidade de atender ao aumento de requisições de usuários através do aumento do número de servidores físicos) e alta disponibilidade (capacidade de suportar

falhas de hardware ou software e manter o sistema a maior tempo possível ativo) por meio de suporte nativo a clustering.

O uso da metodologia de cluster permite que várias instâncias de Tomcat estejam disponíveis para o usuário como um servidor único, permitindo que a carga das requisições sejam distribuídas entre essas várias instâncias (balanceamento de carga), sem o uso de recursos computacionais especializados, fazendo que o sistema possa manipular uma quantidade muito maior de requisições antes de uma possível sobrecarga. O sistema construído também se vale de alta disponibilidade, caso um dos nós que compõem o sistema caia, as requisições de usuários continuam a ser tratadas de maneira transparente.

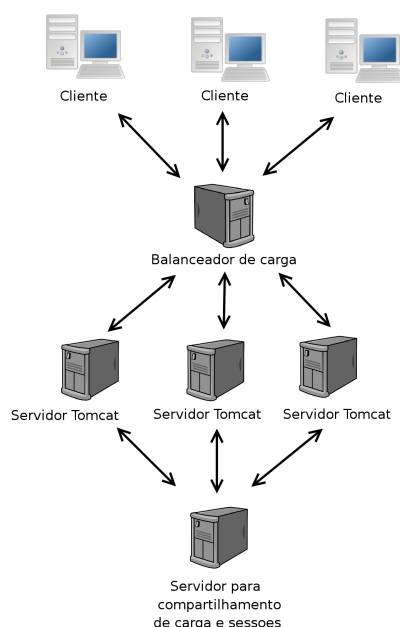


Figura 8.5: Visão geral de um cluster Tomcat

O modelo básico para clusterização em Tomcat envolve o uso de duas camadas adicionais^{8.5}, uma camada responsável pelo balanceamento de carga e outra camada que cuidará do compartilhamento de informações, como sessões e outras variáveis de estado, entre os servidores Tomcat

8.2.1 Balanceamento de carga

Há várias opções que podem ser usadas na camada de balanceamento de carga em um cluster Tomcat, todas elas visando distribuir as requisições de clientes entre os servidores para melhorar a performance do sistema. Entre essas opções serão aqui destacadas:

- . DNS Round-robin.
- . Hardware especializado.
- . Apache mod_jk/mod_jk2.
- . Balanceamento de carga via software, como LVS (switch de camada 4).

DNS Round-robin

DNS Round-robin é a solução mais simples de ser implementada, usando uma lista de IP's dos servidores Tomcat, percorrendo-a de maneira circular enviando cada nova requisição para um IP Tomcat diferente. Muito embora seja uma solução prática de ser implementada, ela não leva em consideração a carga da máquina para a qual uma requisição será enviada, não apresenta vantagens em relação a tolerância a falhas, já que não toma conhecimento de quais máquinas estão sendo ativas, podendo enviar conexões para máquinas inativas, entre outros reveses.

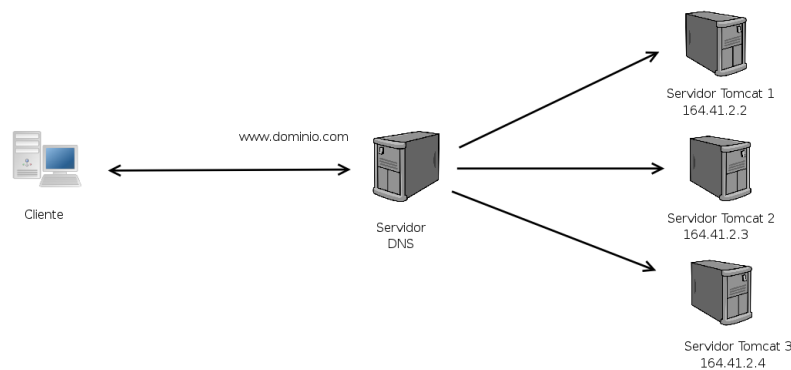


Figura 8.6: Balanceamento de carga via DNS Round-Robin

Em servidores DNS configurados para prestar este tipo de serviço, um endereço (como `www.seudominio.com`) é resolvido para os IP's dos servidores que hospedam as instâncias de Tomcat. Quando um cliente requisita uma requisição, o servidor DNS escolhe um dos IP's e o passa para o cliente.

Hardware especializado

Geralmente, hardware especializado para balanceamento de carga, também conhecido como roteador de camada 4/7, é um dispositivo físico que redireciona conexões para um conjunto de máquinas em uma rede interna. A decisão para o balanceamento é baseada na análise de uma série de fatores como carga do processador, conexões ativas no servidor, entre outros. Isso minimiza a sobrecarga dos servidores além disponibilizar os serviços hospedados nestas máquinas através de um único IP, mapeando as conexões para os IP's internos dos servidores.

Entre as vantagens do uso deste tipo de solução para balanceamento de carga em clusters Tomcat em relação ao uso de DNS Round-robin simples são:

- . Balanceamento de carga mais otimizado, já que fatores como carga de processador e número de conexões ativas são levadas em consideração.
- . Conexões dos clientes não serão enviadas para máquinas que não possam atendê-las.

As principais desvantagens são o custo destes dispositivos, a relativa complexidade de setup e o fato de constituírem um ponto único de falha.

mod_jk

O uso de Apache e seu módulo **mod_jk2** podem ser usados para:

- . Distribuir conexões entre várias instâncias de Tomcat.
- . Detectar falha em instâncias, evitando o envio de requisições a servidores Tomcat que não estejam respondendo.

- . Caso uma instância deixe de responder, mod_jk2 verifica periodicamente se ela voltou, caso a instância volte a responder ela é posta junto com as outras instâncias em funcionamento, voltando a receber requisições.

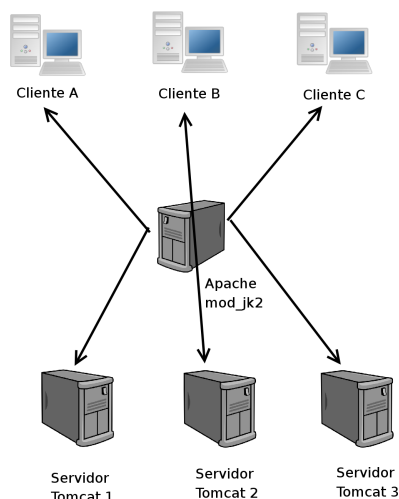


Figura 8.7: Balanceamento de carga via Apache mod_jk

As requisições são distribuídas com mod_jk2 através de um algoritmo de Round-robin podendo ser levado em conta um fator de carga (peso associado a cada instância que regula a prioridade com a qual recebem conexões).

mod_jk2 trabalha também com sessões afins (sticky sessions) que assegura que todas as requisições com mesma sessão serão tratadas pelo mesmo nó Tomcat. A desvantagem desse método é que caso uma instância deixe de funcionar, a sessão associada a ela será perdida.

Balanceamento de carga via software

Entre as soluções usadas para balanceamento de carga via software, uma das mais conhecidas é o LVS. Classificado como um roteador de camada 4, trata-se de modificações incluídas no kernel Linux usadas para redirecionar conexões TCP de maneira transparente para o usuário.

De maneira geral, funciona como o balanceamento feito com hardware especializado. Uma máquina com o sistema operacional Linux (conhecida no jargão LVS como Director) possui o IP que será acessado pelos clientes. O Director, usando

seus algoritmos de escalonamento, decide para qual máquina a conexão será redirecionada. Qualquer serviço TCP pode ser redirecionado, incluindo requisições máquinas que componham um cluster Tomcat.

O LVS mantém algumas informações sobre os servidores (como número de conexões ativas) para o processo de decisão do balanceamento. Também pode não enviar conexões a um servidor que não esteja ativo.

8.2.2 Compartilhamento de sessões

As soluções para balanceamento de carga resolvem o problema da distribuição das requisições dos clientes entre os nós que compõem o sistema. A outra camada mostrada na figura tal, serve a outro propósito, assegurar que sessões e outras informações não sejam perdidas caso o servidor Tomcat, que as manipulavam, caia.

Na camada de compartilhamento, mostrada na figura, podem ser usado alguns tipos de back-ends, cada qual com suas funcionalidades. O compartilhamento de informações nesta camada podem assegurar que a perda de conectividade de um dos servidores Tomcat que compõem o cluster seja manipulada de forma a não gerar transtorno para o usuário.

Sticky sessions em compartilhamento de sessões

Neste tipo de configuração, o balanceador de carga mod_jk2 assegura que as requisições de uma mesma sessão serão sempre tratadas pela mesma instância Tomcat. Este tipo de configuração é conveniente a muitos cenários de produção, apresentando as seguintes características:

- . Escalabilidade para a aplicação provida por algoritmo Round-robin, que cria novas sessões no próximo nó livre na fila round-robin.
- . Simplicidade de implantação e manutenção.
- . Nenhum tipo de configuração adicional ou sobrecarga de recurso.

Apesar das vantagens, as sessões são perdidas se o servidor que as manipula cai.

Sticky sessions com gerenciamento de sessões persistentes e armazenamento compartilhado

A partir da versão 5, Tomcat possui um sistema de gerenciamento de sessões persistentes. O propósito deste mecanismo é manter as sessões ativas caso haja desligamento e reinício de servidor. Para tanto, as sessões são gravadas em disco ou em SGBD, o que garante a manutenção da informação mesmo que o servidor seja desligado. Esse mecanismo, a princípio, não foi desenvolvido para atender demanda de clusterização, entretanto sistemas de arquivos compartilhados ou SGBD essas informações estarão disponíveis para todas as instâncias de Tomcat que compõem o sistema.

A figura ilustra o funcionamento deste mecanismo. Um diretório para armazenamento das sessões é acessível a todos os servidores Tomcat através de mecanismos como SMB, NFS, OCFS2, assim as sessões podem ser criadas ou modificadas por todas as instâncias. Isto também garante uma menor perda de informação em caso de problemas com o sistema e procedimentos de recuperação.

8.3 Heartbeat

O *High Availability Linux Project*³ (Projeto Alta Disponibilidade Linux) “tem por objetivo desenvolver soluções para GNU/Linux que promovam confiabilidade, disponibilidade e resistência(serviceability) através do esforço de uma comunidade de desenvolvimento” .

Heartbeat, fruto deste projeto, é um software que gerencia falhas de recursos entre dois computadores, procurando eliminar pontos únicos de falhas aumentando a disponibilidade do sistema formado por estes computadores. O princípio de funcionamento é o de heartbeat (o conceito não se resume ao software), onde um componente de um sistema de alta disponibilidade responsável por monitorar

³sítio do projeto: <http://www.linux-ha.org/>

os serviços do sistema trocando mensagens entre os servidores para verificar se ainda estão ativos.

Normalmente o Heartbeat trabalha com os conceitos de servidor primário e secundário, ou seja, um servidor que de fato atende a demandas (primário) e outro que fica em espera para assumir serviços caso algo de errado aconteça ao primário. Neste ambiente, um intervalo de tempo para troca de mensagens entre os servidores é especificado, caso não haja troca de mensagens, o Heartbeat entende que o primário está fora do ar, assumindo os serviços por este disponibilizados.

Para verificação do estado de cada nó, o Heartbeat pode usar uma ou mais conexões físicas, podendo ser a conexão ethernet normal para comunicações de rede, interfaces dedicadas ligadas por cabo crossover ou através de cabo serial. A conexão normal de rede não é recomendada por conta da carga normal que ela suporta, sendo ideal o uso de interfaces dedicadas ligadas por crossover ou mesmo cabo serial, que é uma boa opção pela simplicidade e segurança e por, estar sendo usado apenas para esse fim, entretanto é inconveniente a pequena extensão que esses cabos apresentam.

8.4 Zope Cluster

Há uma relação não linear entre o aumento de acesso a um servidor web e seu tempo de resposta às requisições. O uso de uma máquina mais poderosa geralmente não é a resposta para o problema. Uma solução é usar mais de um servidor para realizar o trabalho e distribuir as requisições de serviços entre eles.

Normalmente, um sistema que produz páginas dinâmicas é chamado servidor de aplicação, que é composto, tipicamente, por três partes distintas: um servidor HTTP, um banco de dados e alguma aplicação (middleware) que serve de interface aos outros dois componentes. Estas três partes podem estar todas em uma mesma máquina para o caso de sistemas que não se espera muita carga. Para o caso de sistemas com alta carga, os diferentes requisitos de cada componente sugerem separação para que hardware adequadamente ajustado para atender às suas necessidades.

Zope é uma solução que integra um servidor Web (*ZServer*), *middleware* e um servidor de dados (ZODB) em um único pacote. Como parte desta solução, Zope pode emular a separação entre o servidor Web e o servidor de dados através de ZEO (*Zope Enterprise Objects*).

ZEO é uma parte sistema Zope que permite que um Zope Object Database seja compartilhado entre mais de um processo Zope. Com o uso de ZEO, pode-se rodar múltiplas instâncias de Zope em um único computador ou em vários computadores, acrescentando escalabilidade ao sistema, já que para atender ao possível (e muito provável) aumento de demanda mais máquinas podem ser acrescentadas ao sistema, além do aumento de confiabilidade, caso uma máquina apresente problemas as outras ativas poderão atender a requisições até que a falha seja resolvida.

Os servidores Zoe(instâncias do Zope) que servem a aplicação aos clientes (da Internet ou Intranet) são chamados de clientes nesta arquitetura já que acessam o servidor de aplicação.

Os clientes e servidores ZEO se comunicam através de TCP/IP, o que permite que eles sejam distribuídos, inclusive, geograficamente, sendo capaz de gerenciar uma grande quantidade de requisições simultâneas, a partir de hardware de baixo custo. A única ressalva em relação a esta arquitetura é que não há mecanismos de redundância nativa do ZODB (servidor de armazenamento). Isso pode ser resolvido com o uso de *hardware* especializado (storage externo) ou com dispositivo de bloco como DRBD que pode ser usado para a replicação do banco. Combinado com alguma ferramenta de monitoramento (Heartbeat ou Keepalived), pode-se conseguir redundância para o servidor de armazenamento com o uso de *hardware* não especializado.

Nativamente não há suporte a balancamento de carga no Zope, sendo necessário o uso de ferramentas externas. Vários métodos podem ser utilizados para distribuir as requisições dos clientes entre os servidores ZOE, como DNS *round-robin*, o módulo `mod_proxy` do servidor http Apache ou *switch* de camada 4, sendo o LVS o mais conhecido deles.

Uma solução, para o caso de servidores de páginas estáticas, é usar DNS *round-robin* para distribuir as requisições recebidas por uma URL entre vários IP's de uma rede interna, sendo cada nova requisição enviada para um servidor dife-

rente da anterior. Sendo idêntico o conteúdo de todos os servidores, esse processo é transparente para o usuário. Contudo, esta é uma solução atraente por sua simplicidade entretanto apresenta seus reveses. Por exemplo, arquivos grandes carregaram mais um servidor que esteja atendendo à sua requisição, gerando eventualmente mais carga para alguns servidores que para outros. Outro problema é que o servidor DNS do cliente pode fazer cache do endereço IP acessado e usará este mesmo dado para uma consulta futura.

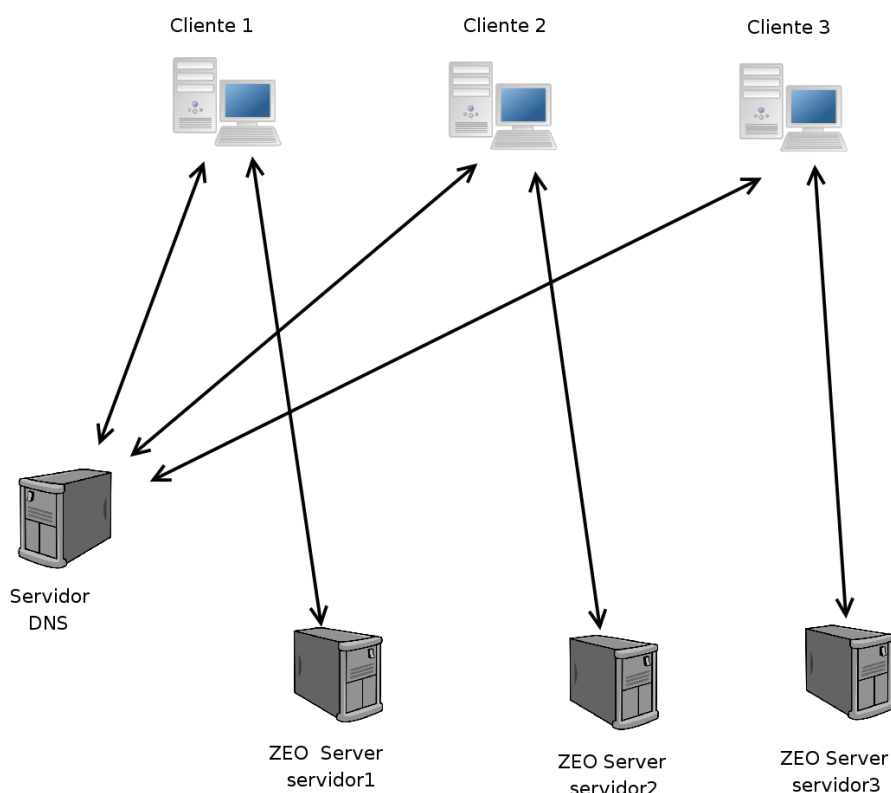


Figura 8.8: DNS round-robin

O DNS *round-robin* é uma maneira de se resolver um nome para vários endereços IP fazendo uso do servidor de DNS para realizar a função de balanceamento de carga, sem o uso de uma máquina dedicada para isso. O servidor DNS resolve `www.dominiozope.com`, por exemplo, para os endereços IP de ZEO Server1, ZEO server2 e ZEO server3 para os clientes 1, 2 e 3, respectivamente.

Outra solução é o uso de balanceamento de carga dinâmico, também tratado como roteador de camada 4. Neste caso um endereço específico é resolvido para apenas um IP que pertence ao roteador que por sua vez, transparentemente, redi-

reciona as conexões para um grupo de servidores em cluster. Preferencialmente, estes servidores possuem a capacidade de informar sobre sua carga de trabalho ao roteador, que depois de obter essa informação decide a qual servidor enviar uma nova requisição. Uma solução em software muito utilizada para isso é o LVS, parte integrante do kernel Linux que o transforma em um switch de camada 4.

O outro problema da arquitetura de cluster Zope é a falta de redundância nativa do servidor de armazenamento. Uma maneira de se retirar esse ponto de falha, além do uso de *hardware* especializado, é o uso conjugado de DRBD, OCFS2, Heartbeat ou Keepalived e LVS. O uso de DRBD (versão 0.8 ou superior) pode ser usado com OCFS2 para se criar um dispositivo que possa ser acessado por duas máquinas com instâncias ZODB lendo e escrevendo ao mesmo tempo. Heartbeat ou Keepalived verifica o estado de sanidade dessas máquinas, tomando providencias necessárias (reinício, notificação administrativa) caso haja algum problema. O LVS, que pode ser usado como balanceador de carga de requisições clientes, pode também balancear a carga dos ZEO clientes quando acessarem os servidores ZODB.

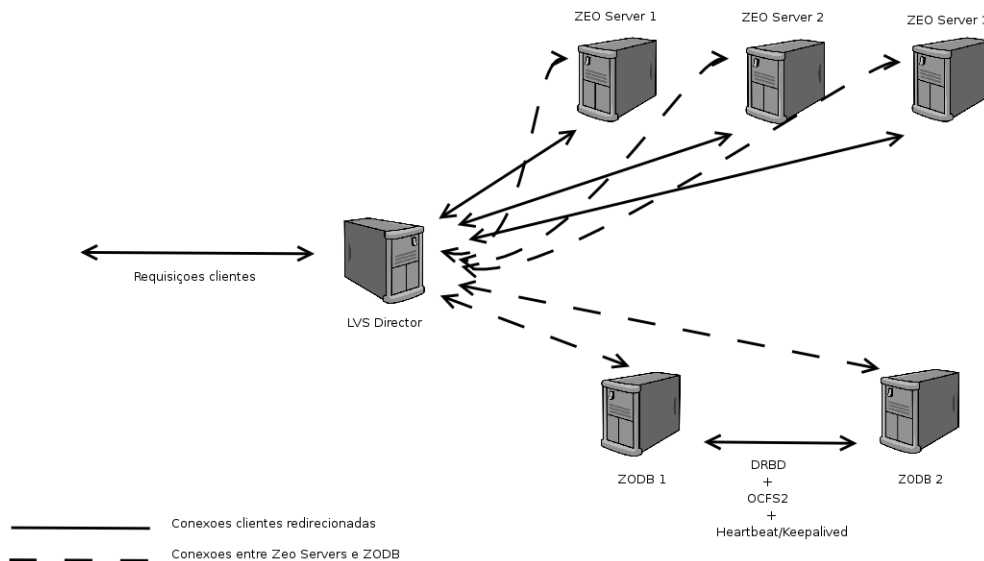


Figura 8.9: ZEO/ZODB + LVS+OCFS2

Capítulo 9

Cluster de Banco de Dados

Na maioria das aplicações que se encontram em produção os dados da aplicação são armazenados em um servidor de banco de dados. Dessa forma o banco de dados se torna um componente crítico na solução da aplicação, uma interrupção no serviço de banco de dados, ou uma pequena corrupção dos dados pode afetar totalmente a integridade da aplicação.

Existem muitas formas de se trabalhar com banco de dados de forma a se obter maior performance e/ou para obter outras características desejáveis que não estão disponíveis facilmente nos SGDBs mais conhecidos. Algumas áreas de desenvolvimento de bancos de dados mais avançados estão dispostos a seguir.

Design de bancos de dados distribuídos.

O design de banco de dados distribuídos responsivo é uma preocupação básica para os sistemas de informação. Em redes com grande largura da banda, latência e processamento local são os fatores mais significativos para consultas e atualização de dados no que se refere a tempo de resposta do sistema. Processamento paralelo pode ser usado para minimizar os efeitos, particularmente se for considerado no momento do design do sistema de banco de dados. A replicação e a localização dos dados na rede que faz o paralelismo ser efetivamente usado. O design de banco de dados distribuído pode ser visto assim como um problema de otimização que requer soluções a vários problemas relacionados: fragmentação

de dados, alocação de dados, e otimização local.

Processamento de consultas distribuído.

Em sistemas distribuídos de grande escala, é freqüentemente difícil achar um plano ótimo para consultas distribuídas: sistemas distribuídos podem ficar muito grandes, envolvendo milhares de sites heterogêneos. Como novos bancos de dados podem ser adicionados/removidos do sistema, fica mais difícil para o “processador de consulta” manter estatísticas precisas das relações participantes armazenadas nos diferentes sites e das operações de consulta relacionadas. Também, como a carga de trabalho dos vários servidores de processamento e da velocidade de transmissão dos links entre eles flutuam durante o tempo de processamento dos trabalhos, há a necessidade de mecanismos de consulta distribuídos, que dinamicamente se adaptem a grandes ambientes distribuídos.

Controle de Concorrência.

O Controle de concorrência (CC) permite os usuários acessar um banco de dados distribuído mantendo a impressão que está executando o sistema em um sistema dedicado.

Para isto, são requeridos mecanismos de CC que intercala a execução de um conjunto de transações debaixo de certas regras de consistência enquanto maximiza a capacidade de execução concorrente do sistema. As duas principais categorias de mecanismos de CC são: Concorrência Otimizada - Retardo da sincronização para as transações até que as operações sejam executadas de fato. Conflitos são menos prováveis mas não serão conhecidos até que eles aconteçam, enquanto tornando operações de rollback mais caras. Pessimista - As execuções potencialmente concorrentes de transações são sincronizadas no início do seus ciclos execução. Bloquear assim é mais fácil, mas terá de se conhecer mais cedo os problemas para diminuir os custos do rollbacks.

Processamento de Transações.

Transações distribuídas provêm unidades de execução segura que permitem que

várias operações sejam executadas em sites diferentes e provêem a preservação da consistência dos dados de um estado de execução para outro. Um protocolo comum por assegurar cumprimento correto de uma transação distribuída é o de “execução em duas fases” (two-phase commit - 2PC). Enquanto 2PC é normalmente aplicados para transações que são executadas em um curto período de tempo, ela se torna impraticável para transações distribuídas de grande escala por causa de travas de recursos disponíveis/ utilizados concorrentemente. Para isto, existem diferentes propostas, como o de dividir a execução de processos que ocupam muito tempo em sucessões de menores tarefas atômicas e a definição de mecanismos de compensação.

Replicação de Dados.

O desafio fundamental na replicação de dados é manter um baixo custo nas atualizações enquanto se assegura a consistência dos sites do cluster. A dificuldade do problema aumenta significativamente em ambientes de larga escala devido a latência alta e a probabilidade de quedas da rede. As duas principais categorias de técnicas de replicação de banco de dados são: replicação síncrona - implica em um protocolo de “commit” atômico ao longo do qual assegura consistência alta às custas de transação mais lenta e a presença de “deadlocks”. Replicação assíncrona - as atualizações são processadas periodicamente por todos os nós do cluster, permitindo um processo de transação mais eficiente, ao custo de uma baixa garantia da consistência global do dados. Também existe a proposta de replicação baseado em grupo de comunicações, para usufruir da vantagem da rica semântica das primitivas dos grupos de comunicação enquanto se utiliza de garantias de relaxadas de isolamento, para reduzir a possibilidade de “deadlocks” e congestionamento de mensagens, assim, aumentando o desempenho do sistema.

Integração de Dados.

Conflitos diferentes surgem da representação descoordenada de conceitos, quando ao se integrar fontes de dados autônomas distribuídos. Exemplos destes conflitos são: tipo de dados, estrutura, conflito de nomes, atributos perdidos, e conflitos de generalização. Todos estes conflitos podem ser estaticamente endereçados entre eles, durante integração dos esquemas de dados ou dinamicamente na geração de visão/consultas. De acordo com a arquitetura de integração, e a

estratégia de manipulação de consultas, sistemas de integração de banco de dados podem ser: Sistema de Multi-banco de dados - coleção de bancos de dados integrados nos quais cada DBMS mantém controle em cima de seus bancos de dados locais, mas coopera com a federação aceitando operações globais. Sistema de mediador - bancos de dados são integrados, através de um componente de mediação que executa tradução de dados em um modelo de dados canônico comum. Sistemas de Metadados - Consultas são formuladas dinamicamente em cada banco de dados pela interação com um dicionário de metadados global.

Existem vários tipos de “clusters” de banco de dados:

- Banco de dados distribuídos: Nesse tipo de banco de dados os dados são distribuídos em um conjunto de servidores. e o Acesso é ao banco de dados é direcionado ao servidor onde se encontra o dado.
- Banco de dados em alta disponibilidade: Dois ou mais servidores em cluster de HA de MASTER e SLAVE, onde um servidor MASTER é responsável pelo serviço e os servidores SLAVE ficam aguardando a falha do MASTER para assumirem o serviço.
- Banco de dados em alta disponibilidade e distribuídos: É um cluster de banco de dados onde as duas tecnologias anteriores estão presentes, criando um banco de dados escalável e tolerante a falhas.

Possíveis tecnologias de cluster de banco de dados:

- Gerenciamento do cluster na aplicação: Nessa alternativa o gerenciamento do cluster é realizado na aplicação que acessa o banco de dados. A aplicação que controla a distribuição e replicação dos dados. Vantagem: Pode ser Independente de sistema de banco de dados. Desvantagem: É dependente da aplicação que está acessando o banco de dados. Exemplo de solução: CJDBC - Cluster Java DataBase Conector, compatível e possui a mesma sintaxe do JDBC e para ser utilizado em uma aplicação que é escrita em java é necessário poucos ajustes na aplicação. Capaz de “clusterizar” QUALQUER banco de dados ODBC.
- Gerenciamento do Cluster no próprio banco de dados: Nesta alternativa o gerenciamento do cluster é implementado através de uma ferramenta no

próprio sistema de banco de dados. Vantagem: Possui maior integração com o sistema de banco de dados, sistema mais robusto de integridade de dados, maior integração com o sistema de banco de dados. Desvantagem: É dependente do sistema de banco de dados. Exemplos: Solução Proprietária: Oracle Real Application Cluster (RAC), Soluções Livres: Mysql Cluster, pgCluster.

- Criação de um Proxy de banco de dados: Semelhante ao gerenciamento na aplicação, só que neste caso é criado um serviço “falso” (honeypot) onde são feitas as conexões e o gerenciamento da distribuição das requisições para os servidores de banco de dados reais.
- LVS + Filesystem distribuído e compartilhado: No fim das contas banco de dados é arquivo armazenado em disco, essa idéia consiste em ter um sistema de arquivos único entre os servidores, que suporte acesso de escrita simultâneo (implementação via software das funcionalidades de uma SAN - Storage Area Network) e um sistema que realize o balanceamento de conexões TCP/ip entre os servidores. Funcionamento: Quando um usuário tenta realizar uma operação de escrita no banco de dados, ele é direcionado através do LVS para um dos servidores de dados, onde é processada a requisição como se o banco não estivesse em cluster. Após a escrita ter sido realizada em disco todos os outros servidores serão capazes de reconhecer transparentemente as alterações realizadas. Um problema nesse tipo de solução é o cache do servidor de banco de dados que tem q ser reduzido para o mínimo possível (Atomic Operations, Atomic Transactions).

A área de banco de dados é bastante sensível e as tecnologias estão começando a se consolidar, é necessário realizar muitos testes para se definir qual a melhor tecnologia a ser adotada para cada situação.

9.1 Banco de Dados Distribuídos

Definições

1. Segundo Date [138],

Um sistema de banco de dados distribuídos consiste em uma coleção de locais, conectados por alguma rede de comunicação e que:

- a. Cada um dos locais é um sistema de banco de dados completo com seus próprios direitos, mas
 - b. Os bancos de dados locais trabalham em conjunto para que os usuários que acessam os dados de qualquer outro local da rede possa acessar os dados de forma transparente.
2. Um banco de dados distribuído é um banco de dados que está sob o controle de um sistema de administração de banco de dados central, no qual dispositivos de armazenamento (storage) são anexados a um computador. O armazenamento pode ser em vários computadores localizados no mesmo local físico, ou pode ser disperso em uma rede de computadores interconectados.

Os dados, o banco de dados, pode ser distribuído fisicamente através de múltiplos locais. Um banco de dados distribuído é dividido em várias partes ou fragmentos. Essas partes ou fragmentos do banco de dados distribuído pode ser replicado, para por exemplo criar ambientes de redundância, RAID, ou mesmo cópias para Data Warehouse.

Além de replicação e fragmentação em banco de dados distribuídos, existem várias outras tecnologias para design de banco de dados distribuídos. Por exemplo, autonomia local, tecnologias de bancos de dados distribuídos síncronos e assíncronos. A implementação destas tecnologias podem e definitivamente dependem das necessidades das áreas de negócios e de a sensibilidade e confiabilidade dos dados a serem armazenados no banco de dados.

9.2 Replicação de Banco de Dados

Um banco de dados replicado é um sistema de bancos de dados com cópias distribuídas em uma ou mais máquinas. Este tipo de sistema oferece alta disponibilidade e tolerância a falhas, já que não há apenas uma única cópia dos dados e melhoria de performance, posto que as requisições não onerarão apenas uma fonte de dados.

Para se implementar a replicação em bancos de dados podem ser usados

dois métodos levando-se em consideração a maneira como é feita a propagação de uma atualização.

Uma primeira aproximação é chamada replicação síncrona, ou seja uma atualização (modificação fruto de uma operação de UPDATE, INSERT ou DELETE, por exemplo) só é consumada se for realizada em todos os nós que compõem o sistema. Isto significa que o cliente terá de esperar até que todas as instâncias do banco de dados sejam modificadas para receber uma confirmação. Por outro lado, isto garante a integridade da informação entre os nós.

A outra aproximação é realizar a atualização de maneira assíncrona, ou seja as modificações são difundidas entre os nós após ser consumada e uma resposta ser enviada ao cliente. O tempo de resposta, se comparado ao método anterior é menor, porém isto pode gerar inconsistências entre as réplicas.

Uma maneira de se contornar estes problemas é restringir as atualizações a um único nó, chamado cópia primária ou master, o que impede que atualizações em um mesmo objeto sejam feitas em duas máquinas diferentes. Todas as operações de modificação no banco serão enviadas para esta máquina que cuidará de propagar as modificações. A contrapartida para este modelo é permitir atualizações em qualquer banco que componha o sistema, não introduzindo uma réplica privilegiada mas requerendo um sistema que resolva conflitos de possíveis inconsistências.

O uso de middleware (software interface entre os clientes e o sistemas de bancos de dados) se tornou um atrativo, já que permite a construção de sistemas replicados sem a necessidade de modificação do sistema de gerenciamento de banco de dados nem no banco de dados em si. Em sistemas desta natureza as requisições são enviadas ao middleware que se encarrega de propagá-las às réplicas de maneira a prover controle de replicação e balanceamento de carga.

Dentre os bancos de dados livres dois vem se sobressaindo, o Mysql[13] e o Postgresql[19], dos quais veremos alguns detalhes sobre a clusterização e replicação de dados.

9.3 PostgreSQL

O PostgreSQL é um SGBD (Sistema Gerenciador de Banco de Dados) objeto-relacional de código aberto, com mais de 15 anos de desenvolvimento. É extremamente robusto e confiável, além de ser extremamente flexível e rico em recursos. Ele é considerado objeto-relacional por implementar, além das características de um SGBD relacional, algumas características de orientação a objetos, como herança e tipos de dados personalizados. A equipe de desenvolvimento do PostgreSQL sempre teve uma grande preocupação em manter a compatibilidade com os padrões SQL92/SQL99/SQL2003 (Postgresql-BR [19]).

As capacidades de Replicação e Clusterização são feitas através de middlewares externos próprios para o PostgreSQL, como o pgpool e o PGcluster, que são detalhados a seguir.

9.3.1 pgpool

pgpool[18] é um middleware para PostgreSQL, distribuído sob licença BSD, que se situa entre os clientes e os servidores de banco de dados provendo alta disponibilidade, replicação e balanceamento de carga. Além destas características em comum com outros sistemas similares, pgpool adicionalmente salva conexões com os servidores PostgreSQL por ele coordenados (pgpool atualmente trabalha apenas com dois servidores PostgreSQL), reutilizando-as quando uma nova conexão com mesmas propriedades (nome de usuário, banco de dados, protocolo) chega, reduzindo sobrecarga de conexão e aumentando a taxa de transferência de todo o sistema.

Como sistema de replicação de dados, pgpool permite backup em tempo real de bancos de dados, enviando as mesmas declarações SQL para ambos os servidores, podendo ser considerado um sistema de replicação síncrona. Apesar disto algumas instruções SQL são dependentes do servidor no qual são executadas como funções aleatórias, OID, XID e timestamp, não sendo replicadas com o mesmo valor para ambos servidores.

Se algum problema torna um dos servidores PostgreSQL indisponível, o pgpool tenta continuar o serviço com o servidor ainda ativo, este modo é chamado “ modo degenerado”. Inconvenientemente, pgpool não oferece

nenhum método para voltar um servidor com problemas de novo no nó, sendo necessário que os bancos sejam sincronizados novamente. A melhor maneira é desativar o servidor ativo, sincronizar os arquivos do postgresql via rsync, por exemplo, reiniciar os bancos e o pgpool.

pgpool envia uma query para o "master" que envia para o "slave" antes do "master" completar a query. Isto pode aumentar a performance mas acrescenta o risco de "deadlock". Para balancear performance e risco, pgpool pode operar de duas formas:

- 1) modo "restrict": Neste modo, pgpool espera a conclusão da query no nó master para depois envia-la para o secundário. Este é o modo de operação padrão e mais seguro do pgpool.
- 2) palavra chave `/*STRICT*/`: Visando performance, o modo restrict pode ser desabilitado através do ajuste da diretiva "pgpool_restrict" na configuração do pgpool. Para inibir deadlocks, deve-se inserir a `/*STRICT*/` no início de cada query passível de produzir deadlock, como por exemplo:

```
/*STRICT*/ LOCK TABLE t1;
```

Caso algum deadlock ocorra, não sendo detectado pelo próprio PostgreSQL, pgpool abortará a sessão se um dos nós não responderem por um certo intervalo de tempo configurável.

Para propósitos de balanceamento de carga (configurável pelo parâmetro "load_balance_mode" no arquivo de configuração), as consultas "SELECT" são distribuídas entre o nó master e o slave de maneira aleatória, para aumentar performance.

É importante notar que mesmo instruções "SELECT" podem introduzir alterações em bancos chamando funções que podem modifica-los. Em casos como este não se deve usar o balanceamento de carga, sendo necessário se usar espaços em branco antes da query para que ela não seja distribuída.

Eis uma lista de vantagens no uso do pgpool:

- . Não é necessário modificações na aplicação.
- . Qualquer linguagem pode ser usada.
- . O número de conexões com o PostgreSQL pode ser limitado.

- . Tolerância a falhas. Caso ocorra falha em um servidor PostgreSQL, o outro assume automaticamente.
- . Replicação.
- . Balanceamento de carga, consultas somente leitura podem ser distribuídas entre os servidores.

Desvantagens:

- . Sobrecarga. Todos os acesso ao PostgreSQL passam pelo pgpool o que pode reduzir um pouco a performance (de 1 a 15% de acordo com os desenvolvedor em testes feitos com pgbench).
- . Nem todos protocolos da libpq são suportados:
 - 1 Nenhum método de autenticação exceto " trust" e " clear text password" em modo de replicação.
 - 2 Em modo não replicado só são aceitos " trust", " clear text password", " crypt" e " md5".
 - 3 Controle de acesso via pg_hba.conf não é suportado.
- . Sem controle de acesso, qualquer um pode acessar pgpool, o que pode ser impedido via iptables, por exemplo.

pgpool-II

pgpool-II é um projeto que herdou as características do pgpool, mas que suporta múltiplas instâncias do PostgreSQL (128 nós, expansível via recompilação do código-fonte) e processamento paralelo nos múltiplos nós, o que aumenta muito a performance.

A arquitetura do pgpool-II consiste nele próprio, "System DB" que processa informações administrativas e operações agregadas, e múltiplos nós de dados onde são armazenados os dados. Novos dados serão incluídos/alterados no nó DB baseado em regras de particionamento pre-definido. Uma regra de particionamento é definida por funções SQL e são armazenadas no "System DB", que é outro servidor PostgreSQL. A arquitetura do pgpool-II é ilustrada na figura 9.1 que se segue.

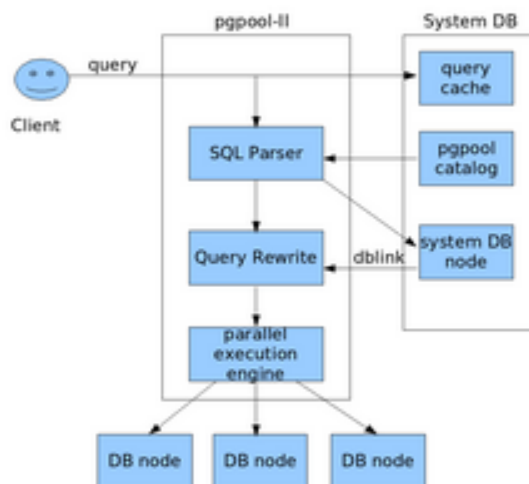


Figura 9.1: Sistema de balanceamento de carga

9.3.2 PGcluster

PGCluster[17] é um conjunto de modificações para o código fonte do PostgreSQL que permite a montagem de um sistema de replicação síncrono multi-master para PostgreSQL, garantindo replicação consistente e balanceamento de carga para bancos de dados baseados em PostgreSQL. A replicação síncrona garante que dados sejam replicados sem que haja atraso e a característica de ser multi-master permite que dois ou mais nós de armazenamento possam receber requisições de usuários ao mesmo tempo.

O sistema é composto por três tipos de máquinas:

- . **servidor de balanceamento (Load Balancer):** recebe consultas e as encaminha para os nós de armazenamento. As consultas são distribuídas de acordo com a carga de cada nó. Pode haver mais de um balanceador de carga.
- . **servidor de armazenamento (Cluster DB):** máquina que recebe e armazena as consultas em bancos de dados.
- . **servidor de replicação (Replicator):** cuida de manter os dados sincronizados entre os servidores. Mais de um servidor pode ser usado, neste caso, outro servidor só assume se o servidor de replicação principal falhar.

O sistema cumpre as seguintes funções:

- . **Balanceamento de carga:** Pela combinação de servidores de armazenamento e servidor de replicação, pode-se criar um sistema onde o PGCluster verificará qual máquina está com menor carga redirecionando uma possível consulta para ela.
- . **Alta disponibilidade:** Com a adição de um balanceador de carga, PGCluster configura um sistema de alta disponibilidade. O balanceador de carga e o servidor de replicação separam um nó que ocasionalmente falhe e continuam a servir com o restante do sistema. Assim que a máquina que falhou for reestabelecida ao sistema os dados são copiados para ela automaticamente, o mesmo acontece com um novo nó que venha a se integrar ao sistema.

Sistema de Balanceamento de Carga Combinando os servidores de armazenamento e servidor de replicação, PGCluster pode distribuir carga de acesso ao banco de dados, verificando qual máquina está com menor carga, redirecionando consultas para ela. A figura 9.2 ilustra a estrutura de balanceamento de carga.

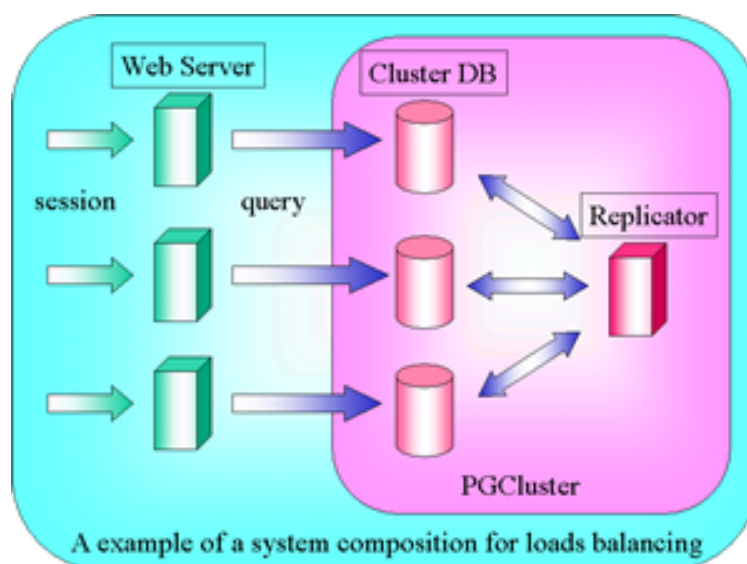


Figura 9.2: Sistema de balanceamento de carga

Sistema de Alta disponibilidade Adicionalmente, PGCluster pode prover um sistema de Alta Disponibilidade pela adição de um balanceador de

carga. Quando uma falha ocorre em um servidor de armazenamento, os servidores de balanceamento e de replicação separam o componente falho e continuam servindo com o restante do sistema. Isto é feito sem que haja interrupção do mesmo. A figura 9.3 ilustra a estrutura de Alta Disponibilidade para o PGcluster.

Quando uma máquina que falhou é recolocada no sistema, os dados são copiados para ela automaticamente, o mesmo acontecendo quando um servidor de armazenamento novo é adicionado.

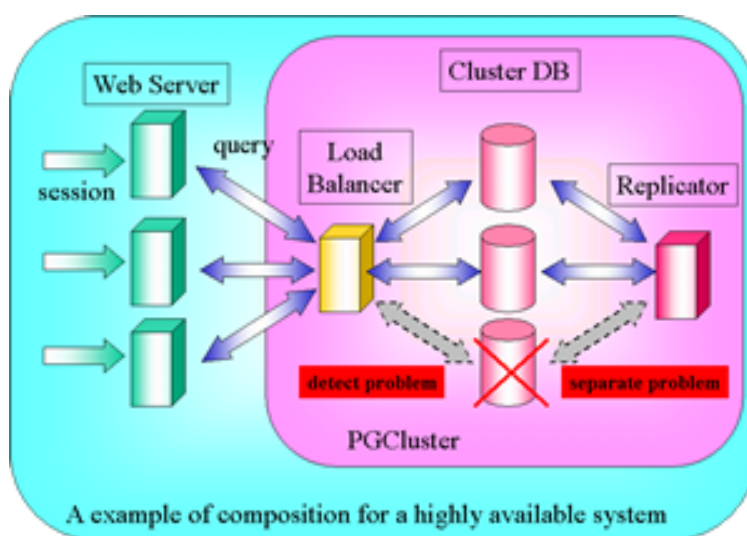


Figura 9.3: Sistema de alta disponibilidade

9.3.3 Slony

Slony[7] é um sistema de replicação “master” para “muitos slaves” que suporta cascadeamento e promoção de “slaves” para “master”, que inclui capacidade para replicar grandes bancos de dados em até doze servidores slave. O Slony foi criado para atender a data-centers e sites de backup onde todos os nós devem estar disponíveis a qualquer momento.

Há alguns modelos distintos de replicação de bancos de dados, é difícil que um modelo se adapte à todas as necessidades. O modelo implementado no Slony-I é chamado de replicação assíncrona usando triggers para coletar as atualizações, onde uma origem comum é replicada em múltiplas cópias incluindo cópias cascadeadas.

Em algumas situações, Slony não é aconselhável:

- . Sistemas com conectividade flakey(?)
- . Replicação em nós com certa imprevisibilidade na conexão.
- . Sistemas cuja configuração mude de maneira aleatória.
- . Sistemas onde schemas de bancos de dados podem ser mudados arbitrariamente.

Slony é um sistema de replicação criado para ser independente de versão do PostgreSQL, permitindo ser iniciado ou parado sem a necessidade de ciclos de dump/reload.

Dentre as coisas que slony não é:

- . Não é um sistema de gerenciamento de rede.
- . Não possui nenhuma funcionalidade para detectar falha de nó, nem muda um nó master para outro, embora isso possa ser conseguido em conjunção com outras ferramentas.
- . Não é um sistema de replicação multi-master, mas há planos para transforma-lo em multi-master na versão 2, muito embora seja um projeto separado e ainda em desenvolvimento.

Slony não propaga mudanças em schemas, nem replica grandes objetos. Slony coleta as atualizações com triggers e nem mudanças em schemas ou operações com grandes objetos dão aos triggers habilidade para reportar ao Slony suas mudanças, portanto apenas tabelas e seqüências são replicadas.

Modelos de replicação Há alguns modelos distintos de replicação de bancos de dados, é difícil que um modelo se adapte à todas as necessidades. O modelo implementado no Slony-I é chamado de replicação assíncrona usando triggers para coletar as atualizações, onde uma origem comum é replicada em múltiplas cópias incluindo cópias cascadeadas.

9.4 Mysql

O MySQL é um servidor de bancos de dados SQL (Structured Query Language - Linguagem Estruturada para Pesquisas) muito rápido, multi-tarefa e multi-usuário. O Servidor MySQL pode ser usado em sistemas de produção com alta carga e missão crítica bem como pode ser embutido em programa de uso em massa. MySQL é uma marca registrada da MySQL AB [13].

9.4.1 Replicação em MySQL

Uma das dificuldades no trabalho com grandes bancos de dados MySQL é a manutenção de backup seguro sem a necessidade do desligamento do sistema. Um backup online, poderia deixar o sistema lento além de causar inconsistências de dados, já que tabelas podem estar sendo modificadas enquanto o backup é feito. O problema da inconsistência poderia ser resolvido com o desligamento do sistema, mas deixaria os usuários do sistema sem acesso ao serviço. O backup é de fato uma medida necessária, mas o desligamento diário do sistema é inaceitável. Um método alternativo simples para se assegurar backups confiáveis mantendo disponível o sistema é a replicação do MySQL.

A replicação é uma configuração onde um servidor MySQL, conhecido neste contexto como master, armazena dados e gerencia as conexões dos clientes enquanto outro (ou outros) servidores mantêm uma cópia completa dos dados do master, duplicando as declarações SQL executadas no master logo assim que elas aconteçam.

O sistema de replicação MySQL permite que múltiplos servidores slave mantenham seus dados sincronizados com um único servidor master. Entre as vantagens da replicação estão a facilidade de backup e recuperação, além de dar melhor suporte a grandes aplicações. É possível se conseguir

escalabilidade linear enviando as requisições de escrita (INSERT, UPDATE, DELETE) para o servidor master e as conexões de leitura (SELECT) para os servidores slave.

Replicação oferece robustez, velocidade e vantagens administrativas:

- . Robustez é incrementada com uma configuração master/slave, caso ocorra algum problema com o master, um slave pode assumir seu lugar.
- . Melhora no tempo de resposta aos clientes pode ser conseguida dividindo a carga para o processamento das consultas do clientes entre master e slaves. As consultas SELECT podem ser enviadas aos slaves para reduzir a carga do processamento de consultas no master. Declarações que impliquem em modificações devem ser enviadas ao master para manter os dados sincronizados.
- . Outro benefício do uso da replicação é que backups de bancos de dados podem ser realizados sem interromper o master. O master continua os processos de atualização enquanto o backup é feito.

O Processo de Replicação

Quando o sistema de replicação esta rodando, quaisquer declarações SQL executadas no servidor master. MySQL grava estas declarações em um log binário (bin.log) juntamente com um número que identifica sua posição no log. O servidor slave, com certa frequência, verifica este log em busca de mudanças. Se uma mudança é encontrada, o slave a copia para seu log de vigilância (relay.log). Além disso, o slave grava um novo número de identificação posicional em um arquivo (master.info). O slave, então, volta a verificar o master, quando alguma alteração é encontrada ela é executada e gravada no relay.log. Como salvaguarda, através de uma thread SQL o slave compara seus dados com os dados do master. Se a comparação se mostrar inconsistente, o processo de replicação para e uma mensagem de erro é gravada no log de erro do slave (error.log). Se o resultado for satisfatório (os dados estiverem corretos), um novo número de identificação de log é gravado no relay-log.info e o slave espera por outra mudança em seu arquivo relay.log.

O processo parece complicado à primeira vista, mas é rápido e não ocupa significativamente o master, além de assegurar replicação confiável, sendo também muito simples de configurar, significando algumas poucas linhas

adicionais ao arquivo de configuração do MySQL (my.cnf) em ambos os servidores master e slave. Se o servidor slave é novo, você precisará de uma cópia dos bancos de dados do master no slave para colocá-lo no ar. Então o problema se resumirá a iniciar o servidor slave para começar a replicação.

Visão geral da implementação da replicação

A replicação em MySQL é baseada no log binário das alterações feitas nos bancos de dados do servidor master. Cada servidor slave recebe do master as alterações salvas que foram gravadas executando-as em seu conjunto de dados.

É importante frisar que o log binário é simplesmente uma gravação começando de um ponto fixo a partir do momento em que este log foi habilitado no servidor. Qualquer slave precisará de cópias das bases de dados do master exatamente como existiam no momento em que o log binário foi iniciado, de outra forma a replicação falhará.

Após ser configurado, o servidor slave conecta ao master e espera por atualizações. Se o master cair ou a conexão for perdida, o slave continuará tentando se conectar periodicamente até que seja capaz de receber informações sobre modificações. O intervalo padrão é 60 segundos.

Replicação baseada em coluna

A propagação de declarações SQL do master para o slave é o princípio original da replicação em MySQL. Isto é chamado replicação baseada em declaração. A partir da versão MySQL 5.1.5, outro modelo passou a estar disponível, a chamada replicação baseada em coluna. Ao invés de enviar as declarações MySQL para o slave, o master escreve os eventos em um log binário indicando como as colunas individuais das tabelas foram afetadas. A versão 5.1.8 oferece uma terceira opção: a mista, que usa replicação baseada em declaração por padrão e a baseada em coluna em casos particulares.

Adicionalmente à mudança automática do formato de log, um servidor slave pode mudar o formato automaticamente. Isto acontece quando um servidor está rodando em modo STATEMENT ou MIXED e encontra uma coluna no log binário que foi escrita em formato ROW. Neste caso, o slave muda para modo de replicação coluna temporariamente para este evento, voltando para o modo prévio logo após.

Há duas razões para se ajustar o log de replicação baseado na conexão:

- . Com uma thread que faz pequenas modificações no banco de dados é preferível usar log baseado em coluna. Uma thread que realiza updates em várias colunas com uma cláusula WHERE deve usar log baseado em declaração por ser mais eficiente para gravadas no log poucas declarações que muitas colunas.
- . Algumas declarações requerem muito tempo de execução no master, mas resultam em poucas colunas modificadas. Deve então, ser benéfico replicá-las usando-se log baseado em coluna.

Há algumas exceção para as quais não se pode mudar o modo de replicação em tempo de execução:

- . Funções armazenadas e trigger.
- . Se NDB estiver habilitado.
- . Se a sessão aberta em modo de replicação em coluna e tabelas estiverem temporariamente abertas.

Não é recomendado mudar o modo de replicação em tempo de execução enquanto tabelas temporárias existirem, porque estas tabelas são gravadas no log apenas com replicação baseada em declaração, com replicação baseada em coluna elas não serão gravadas no log. Com replicação mista, tabelas temporárias são usualmente gravadas no log, exceções para os casos de funções definidas pelo usuário (UDF) e a função UUID().

Técnicas avançadas de replicação

MySQL cluster é uma arquitetura complexa que provê alta disponibilidade e performance. Sua principal vantagem é que cada nó atua como master diferente do sistema de replicação em que há um nó master e vários slaves e as aplicações devem escrever apenas no master.

As principais desvantagens do MySQL cluster são:

- . O banco de dados ficará em memória somente e isto requer mais recursos que um banco de dados MySQL normal. MySQL 5.1 introduz tablespaces com a capacidade de armazenar dados não indexados em disco.

- . Algumas características não estão presentes como procura full-text, integridade referencial e níveis de isolamento de transação maiores que READ COMMITTED¹.

Embora em alguns casos MySQL cluster seja uma solução perfeita, a replicação ainda é, na maioria das vezes, a melhor escolha.

Entretanto, replicação também tem seus problemas:

- . Há a distinção entre master e slaves e as aplicações devem apenas escrever no master.
- . Quanto a tolerância a falhas, quando o master cai, há os slaves prontos para assumir, mas o processo de detecção de falhas e substituição do master requerem intervenção do administrador.

9.4.2 MySQL Cluster

MySQL cluster é tecnologia que habilita clusterização de bancos de dados em memória sem dependência de hardware ou tecnologia específica. MySQL Cluster implementa uma arquitetura distribuída, altamente tolerante a falhas com nenhum ponto único de falha (SPOF), recuperação automática de nós garantindo a confiabilidade de um mainframe em hardware de baixo custo, constituindo uma solução de alta disponibilidade com excelente custo benefício.

O sistema integra um servidor MySQL padrão com componente para armazenamento clusterizado em memória chamado NDB, consistindo de um conjunto de computadores rodando processos que incluem servidores MySQL, nós de dados para o cluster NDB, servidores de gerenciamento e programas especializados para acesso a dados.

A engine de armazenamento NDB pode ser configurada com muitas opções de tolerância a falhas e balanceamento de carga. Cada parte do MySQL cluster é configurada independentemente dos servidores MySQL, sendo que

¹Nível de isolamento de transação define a interação e visibilidade do trabalho realizado por transações simultâneas. O SQL padrão define quatro níveis de isolamento de transação

- . READ UNCOMMITTED: uma transação enxerga as mudanças realizadas por outras transações ainda não finalizadas.
- . READ COMMITTED:

cada parte é considerada um nó².

Há três tipos de nós e em uma configuração mínima de um cluster MySQL, um de cada tipo:

- . **Nó de gerenciamento:** Sua função é gerenciar os outros nós do cluster exercendo funções como prover dados de configuração, iniciar e parar outros nós, backup entre outras coisas. Deve ser o primeiro a ser iniciado pois gerencia a configuração dos outros nós.
- . **Nó de dados:** Este tipo de nó armazena os dados do cluster. Há tantos nós de dados quantas réplicas vezes o número de fragmentos³.
- . **Nó SQL:** Este nó acessa os dados do cluster, é um servidor MySQL tradicional que usa a engine de armazenamento NDB.

Num ambiente real de produção, a configuração com três nós não inclui redundância. Para se beneficiar das propriedades de alta disponibilidade do MySQL Cluster é recomendável se usar múltiplos nós de gerenciamento, dados e SQL.

9.5 Middlewares independentes de Banco de Dados

9.5.1 Middleware Sequoia

Os dados aqui apresentados sobre o Sequoia tem como referência a documentação "User Guide" traduzida pela equipe do Ministério do Planejamento, que pode ser obtida no endereço <http://guialivre.governoeletronico.gov.br/guiacluster/>

²Em muitos contextos um nó é geralmente uma máquina, mas em MySQL cluster nó é um processo, sendo possível rodar qualquer número de nós em uma única máquina

³No contexto do MySQL Cluster, fragmento é uma parte de um banco de dados, uma tabela dividida entre vários nós para facilitar balanceamento de carga entre as máquinas e nós. Cada fragmento é armazenado como réplicas em outros nós para prover redundância.

O que é Sequoia

Sequoia é um middleware de cluster que permite a qualquer aplicação Java™ (aplicação standalone, servlet ou container EJB™) acessar, transparentemente, um cluster de banco de dados através de JDBC™. Não é necessário modificar aplicações clientes, aplicações servidoras ou servidor de banco de dados. Basta apenas que os banco de dados sejam acessados pelo Sequoia.

Sequoia é um projeto livre de código aberto, continuação do projeto C-JDBC (<http://c-jdbc.objectweb.org>) hospedado pelo projeto ObjectWeb Consortium (<http://www.objectweb.org>). Sequoia é liberado sob a licença Apache v2 (www.apache.org/licenses/LICENSE-2.0.html) enquanto C-JDBC esta sob GNU Lesser General Public License (<http://www.gnu.org/copyleft/lesser.html>) (LGPL).

Sequoia também provê driver para aplicações não escritas em Java. O desenvolvimento do driver está na página do projeto Carob (carob.continuent.org). Um plug-in Eclipse para Sequoia está disponível na página do projeto Oak (oak.continuent.org).

O que eu preciso para usar Sequoia

Para se usar Sequoia, é necessário o que se segue:

- . uma aplicação cliente que acesse um banco de dados através de **JDBC**.
- . uma **máquina virtual Java™** compilada para **JDK™1.4**(ou mais recente)⁴.
- . um banco de dados com driver **JDBC**(tipo 1, 2, 3 ou 4) ou um driver **ODBC** para ser usado com **JDBC-ODBC bridge**.
- . comunicação de rede com suporte a **TCP/IP** entre os nós do cluster.

Nota: Se sua aplicação cliente não suporta **JDBC**, você pode usar a API C++ ou o driver ODBC disponibilizado pelo projeto Carob (<http://carob.continuent.org>)

⁴Sequoia pode funcionar com versões mais antigas da máquina virtual, mas não foi testado.

Porque devo usar Sequoia

Se você tem uma aplicação cliente em Java ou uma aplicação de servidor baseada em Java que acessa um ou vários bancos de dados. A fila do banco de dados torna-se um gargalo para sua aplicação ou um ponto único de falha ou ambas as coisas. Sequoia pode ajudar a resolver este problema provendo:

- . Escalabilidade de performance pela adição de nós de banco de dados e balanço de carga entre os nós.
- . Alta disponibilidade para o banco de dados: se ocorrer problemas com os bancos, Sequoia oferece tolerância a falhas de maneira transparente usando técnicas de replicação.
- . Incrementa performance com cache de consultas de granulosidade fina e pool de conexão transparente.
- . Log de tráfego SQL para monitoramento e análise de performance.
- . Suporte para cluster de bancos heterogêneos.

Como funciona

Sequoia provê arquitetura flexível que permite alcançar escalabilidade, alta disponibilidade e tolerância a falhas para banco de dados. Sequoia implementa o conceito de RAIDb: Array Redundante não-oneroso de banco de dados (*Redundant Array of Inexpensive Databases*). O banco de dados é distribuído e replicado entre vários nós e Sequoia distribui a carga das consultas entre eles.

Sequoia dispõe de um driver **JDBC** genérico para ser usado pelos clientes. Este driver repassa as requisições para o controlador Sequoia que faz o balanceamento do cluster de banco de dados (leituras são balanceadas e escritas são difundidas). Sequoia pode usar qualquer SGBD (Sistema de Gerenciamento de Bancos de Dados Relacionais - *Relational DataBase Management System*) provendo um driver **JDBC**, isto é valido para todos os bancos de dados em Código Aberto de Comerciais existentes. Sequoia permite configurar um cluster que inclua uma mistura de bancos de diferentes fornecedores. As principais características disponibilizadas

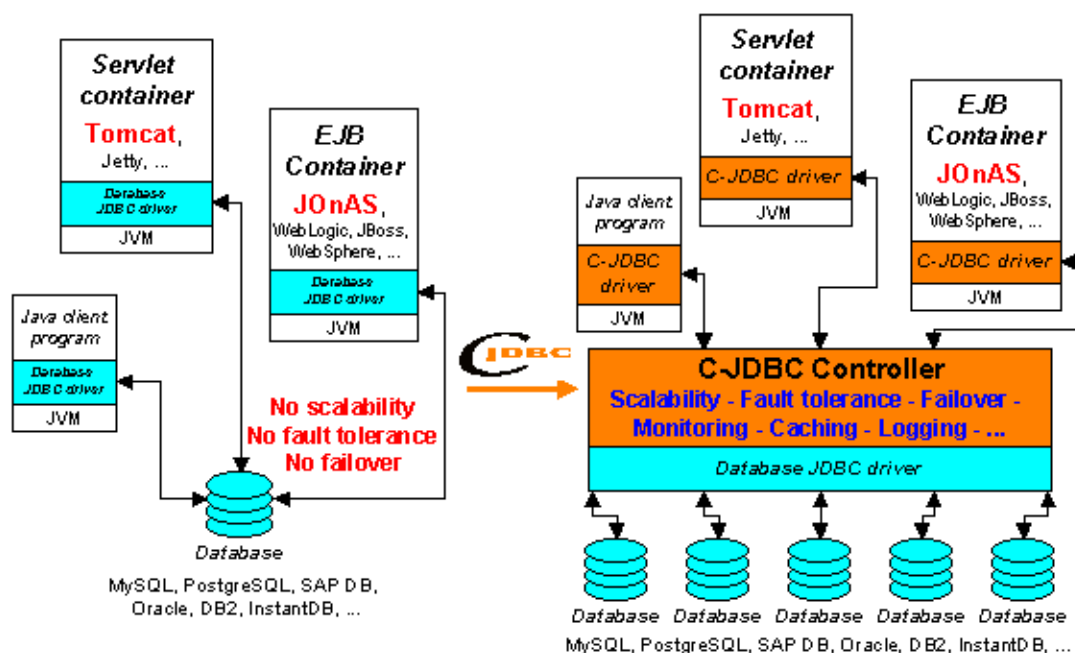


Figura 9.4: Princípio do Sequoia

por Sequoia são performance escalável, tolerância a falhas e alta disponibilidade. Características adicionais são monitoramento, log, cache de consultas SQL.

A arquitetura é completamente aberta permitindo que qualquer um adicione customizações como agendadores de requisições, balanceadores de carga, gerenciadores de conexão, políticas de caching...

Qual o custo

Sob o ponto de vista de software, Sequoia é um software de código aberto licenciado sob a Licença Apache v2 significando que seu uso (pessoal ou comercial) é livre de qualquer taxa. Se você estiver usando um SGBD comercial (Oracle, DB2, ...), haverá os custos das licenças adicionais para nós extras nos quais você instalará réplicas de seu banco. Mas é possível o uso de bancos de código aberto para hospedar réplicas de seu banco de dados principal.

Máquinas extras são necessárias para maior performance e maior tolerância a falhas. Sequoia foi desenhado para trabalhar com máquinas de baixo custo pois são estas as alvos primeiros de soluções em código aberto de baixo custo, mas ele funciona igualmente bem em grandes máquinas SMP. Uma placa de rede comum

é suficiente para uma boa performance.

Quais modificações são necessárias

Você não necessita de qualquer mudança em sua aplicação ou seu banco de dados.

É necessária, apenas, a atualização da configuração do driver **JDBC** usado por sua aplicação (usualmente é apenas a atualização de um arquivo de configuração) e os ajustes no arquivo de configuração do Sequoia.

RAIDb básico

A equipe de desenvolvimento do C-JDBC (antigo nome do projeto Sequoia) criou e implementou o conceito de RAIDb. RAIDb está para bancos de dados assim como RAID está para discos. RAID combina vários discos rígidos de baixo custo em um array de discos para obter performance, capacidade e confiabilidade que excedem as capacidades de um disco de grande capacidade. RAIDb visa melhor performance e tolerância a falhas pela combinação de múltiplas instâncias de bancos de dados em um array de bancos de dados.

RAIDb objetiva o uso de hardware e software de baixo custo como cluster de workstations e bancos de dados de código aberto. Clusters de workstations já são alternativa viável se comparadas às máquinas paralelas usadas em pesquisa científica pela vantajoso custo/benefício. Clusters desta natureza podem ser usados para prover alta disponibilidade e escalabilidade a ambientes de bancos de dados.

RAIDb esconde a complexidade da replicação disponibilizando ao cliente a visão de acesso a um único banco de dados, usando uma máquina (controller) que é colocada à frente dos recursos disponíveis. Os cliente direcionam suas requisições ao controlador RAIDb que por sua vez as distribui ao seu conjunto de sistemas de gerenciamento de bancos de dados (SGBD).

Os três níveis básicos do RAIDb são:

- . **RAIDb-0:** que particiona o banco de dados entre os nós mas não oferece tolerância a falhas.
- . **RAIDb-1:** para espelhamento completo.
- . **RAIDb-2:** que oferece replicação parcial.

Também estão definidos, RAIDb-1ec e RAIDb-2ec que adicionam verificação de erros aos níveis RAIDb-1 e RAIDb-2, respectivamente.

RAIDb-0: Com este nível de RAIDb pode-se particionar o banco de dados, distribuindo suas tabelas entre os backends (máquina que hospeda o banco). Uma tabela em si não pode ser particionada, mas tabelas diferentes podem estar em diferentes nós. Esta configuração requer no mínimo dois backends, provendo escalabilidade de performance mas sem tolerância a falhas.

A **figura 9.5** mostra um exemplo de configuração **RAIDb-0**.

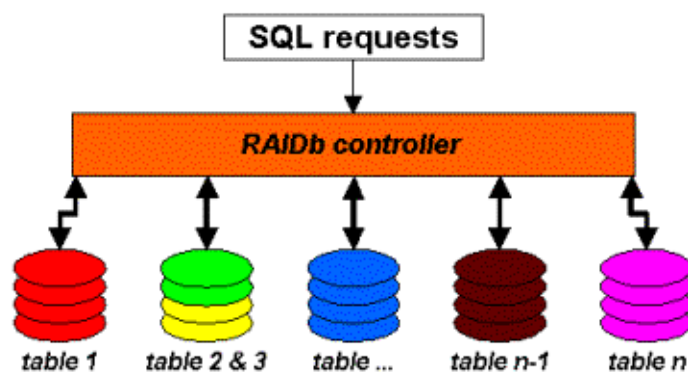


Figura 9.5: Exemplo de RAIDb-0

RAIDb-1 RAIDb-1 oferece espelhamento completo com replicação total dos bancos de dados nos backends. Este modelo de configuração é o apresenta melhor tolerância a falhas já que o sistema ainda estará disponível se apenas um nó estiver ativo. A desvantagem é que não há melhoria nos processos de escrita em banco (UPDATE, INSERT, DELETE) já que todos estes tipos de requisições serão

enviadas para todos os nós. Para assegurar integridade, RAIDb-1ec realiza verificação de erros ao RAIDb-1, objetivando detecção de erros bizarros. Requerendo ao menos 3 nós, esta configuração envia as requisições de escrita para a maioria dos nós que compõem o cluster e as respostas são comparadas. Se há consenso nas respostas, elas são enviadas aos clientes, caso contrário uma mensagem de erro é enviada em resposta a requisição.

A **figura 9.6** mostra um exemplo da configuração RAIDb-1.

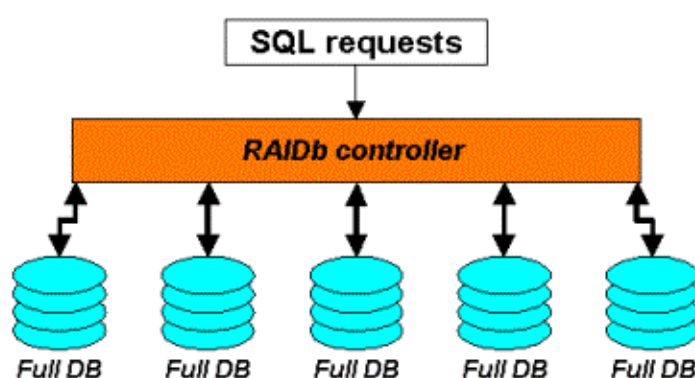


Figura 9.6: Exemplo de RAIDb-1

RAIDb-2 RAIDb-2 é a combinação de RAIDb-0 e RAIDb-1, provendo replicação parcial para diminuir a degradação no processo de replicação de cada tabela do banco de dados e obtendo melhores resultados de leitura e escrita. Este modelo requer que cada tabela esteja disponível em pelo menos 2 nós. Também implementa verificação de erro como RAIDb-1ec. Opera com um mínimo de 4 nós sendo que 3 deles configuram quorum para resposta. A escolha dos nós também é mais complexa que em RAIDb-1ec dada a possibilidade de replicação parcial, embora esta característica garanta uma melhor performance. A **figura 9.7** mostra um exemplo desta configuração.

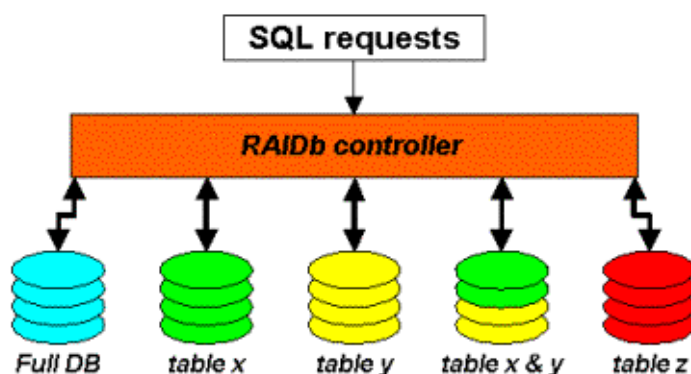


Figura 9.7: Exemplo de RAIDb-2

Níveis de RAIDb aninhados É possível usar vários níveis de RAIDb ao mesmo tempo para se configurar grandes sistemas ou atender necessidades específicas. Um controlador RAIDb escala um número limitado de bancos de dados, entretanto pode-se cascatear vários controladores para disponibilizar mais backends. Em princípio, não há limite para a profundidade do aninhamento de RAIDb. RAIDb de mesmo nível podem ser aninhados como por exemplo um sistema com RAIDb-1-1 para espelhamento de vários bancos de dados. Para evitar que o controlador se torne um ponto único de falha, dois ou mais controladores podem ser usados para atender às requisições. O middleware de comunicação de grupo JGroups é usado para sincronizar as modificações nos bancos de maneira distribuída. Os bancos de dados não necessitam ser compartilhados entre os controladores, mas se caso um controlador caia, os bancos associados a ele também ficarão indisponíveis. No caso de backends compartilhados, um controlador enviará as requisições aos backends informando ao outro controlador assim que as requisições estiverem completadas.

A figura 9.8. mostra um exemplo de uma configuração RAIDb-1-0.

O último exemplo (figura 9.9) mostra uma composição RAIDb-0-1. O nível mais alto é um controlador RAIDb-0 e a tolerância a falhas é conseguida graças a cada partição usando controlador RAIDb-1.

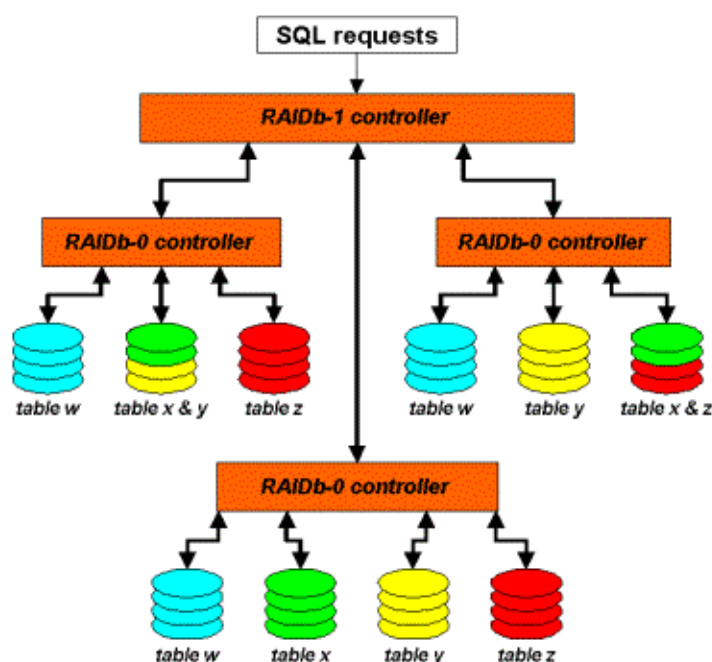


Figura 9.8: Exemplo de RAIDb-1-0

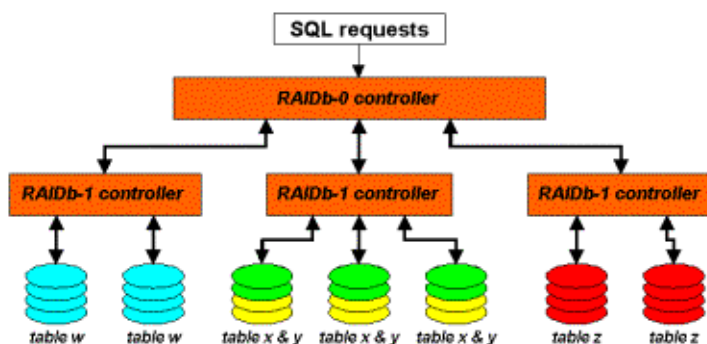


Figura 9.9: Exemplo de RAIDb-0-1

Balanceamento de carga

O balanceador de carga define a maneira que as requisições serão distribuídas entre os backends de acordo com o nível do RAIDb. É possível reforçar um nível de isolamento específico para todas as conexões (isto não terá efeito se o banco de dados usado não suporta isolamento de transações). Em geral, o nível de isolamento de transação padrão será usado e nenhum reforço será dado às conexões. Os seguintes balanceadores de carga estão disponíveis:

- . SingleDB: balanceador de carga para um instância única de banco de dados.

Disponível se apenas um controlador for usado.

- . **ParallelDB**: balanceador de carga para ser usado em banco de dados paralelos, como Oracle Parallel Server ou Middle-R. Ambas, escrita e leitura, são balanceadas entre os backends, deixando a replicação paralela dos bancos escrever.
- . **RAIDb-0**: particionamento completo de banco de dados (nenhuma tabela pode ser replicada) com política opcional que especifica onde novas tabelas serão criadas.
- . **RAIDb-1**: espelhamento completo de banco de dados (todas as tabelas são replicadas em qualquer lugar) com política opcional que especifica como a consumação de consultas distribuídas (write/commit/rollback) são manipuladas (quando o primeiro, a maioria ou todos os backends completam as consultas).
- . **RAIDb-1ec**: espelhamento completo de banco de dados, como em RAIDb-1, mais verificação de erros para detecção de falhas bizarras.
- . **RAIDb-2**: replicação parcial (cada tabela deve ser replicada pelo menos 1 vez) com políticas opcionais para criação de novas tabelas (como RAIDb-0) e consumo de consultas distribuídas (como em RAIDb-1).
- . **RAIDb-2ec**: replicação parcial (como RAIDb-2) com verificação de detecção de erros bizarras.

O elemento balanceador de carga é definido como:

```
<!ELEMENT LoadBalancer (SingleDB | ParallelDB | RAIDb-0 |
  RAIDb-1 | RAIDb-1ec | RAIDb-2 | RAIDb-2ec)>
<!ATTLIST LoadBalancer
  transactionIsolation (databaseDefault | readUncommitted |
    readCommitted | repeatableRead | serializable) "databaseDefault">
```

Balanceamento de carga SingleDB O balanceador de carga **SingleDB** não necessita de nenhum parâmetro específico. A definição é:

```
<!ELEMENT SingleDB EMPTY>
```

Balanceamento de carga ParallelDB O balanceador de carga ParallelDB deve ser usado com um agendador de requisições SingleDB. Este balanceador de carga provê duas implementações: ParallelDB-RoundRobin e ParallelDB-LeastPendingRequestFirst que provêm políticas de round robin e menor número de consultas pendentes primeiro para balanceamento de carga, respectivamente. Este balanceador de carga é desenhado para prover balanceamento de carga e tolerância a falhas em cima de banco de dados paralelos como Oracle Parallel Server ou Middle-R. Isto significa que consultas de leitura e escrita são enviadas apenas para backends ativos, o banco de dados paralelo é responsável por manter a consistência entre os backends. A definição do elemento ParallelDB é:

```
<!ELEMENT ParallelDB (ParallelDB-RoundRobin |
ParallelDB-LeastPendingRequestsFirst)>

<!ELEMENT ParallelDB-RoundRobin EMPTY>
<!ELEMENT ParallelDB-LeastPendingRequestsFirst EMPTY>
```

Nenhum ajuste específico é requerido para estes balanceadores de carga. Não requerem análise de requisições o que significa que as requisições são apenas repassadas para os backends (regras de reescritas ainda são aplicadas mas nenhuma transformação automática é realizada).

Balanceamento de carga RAIDb-0 O balanceador de carga RAIDb-0 aceita política para especificar onde novas tabelas serão criadas. A definição do elemento RAIDb-0 é dada por:

```
<!ELEMENT RAIDb-0 (MacroHandling?, CreateTable*)>

<!ELEMENT CreateTable (BackendName*)>
<!ATTLIST CreateTable
    tableName      CDATA #IMPLIED
    policy          (random | roundRobin | all) #REQUIRED
    numberOfNodes  CDATA #REQUIRED
>
```

```
<!-- BackendName simply identifies a backend by its logical name -->
<!ELEMENT BackendName EMPTY>
<!ATTLIST BackendName
    name CDATA #REQUIRED>
```

Se `MacroHandling` é omitido, um padrão é inserido.

`CreateTable` define a política a ser adotada para criação de tabelas novas. Esta política baseia-se na lista de nós `BackendName` dada, que dever ser um subconjunto do conjunto completo de backends. Se a lista for omitida, todos serão usados. Os atributos possuem os seguintes significados:

- . `numberOfNodes` representa o número de backends que serão usados da lista `BackendName` para aplicação da política, devendo ser 1 para `RAIDb-0` e nunca maior que o número de nós declarados em `BackendName`.
- . `policy`: funciona da seguinte forma:
 - . `random`: backends de `numberOfNodes` serão aleatoriamente tomados de `BackendName` e as tabelas neles criadas.
 - . `roundRobin`: backends de `numberOfNodes` serão tomados de `BackendName` através de algoritmo de round robin e as tabelas neles serão criadas.
 - . `all`: as tabelas serão criadas em todos os nós de `BackendName`, `numberOfNodes` será ignorada.

Um exemplo de um controlador `RAIDb-0` com três nós onde novas tabelas são criadas aleatoriamente nos primeiros dois nós:

```
...
<DatabaseBackend name="node1" ...
<DatabaseBackend name="node2" ...
<DatabaseBackend name="node3" ...
...
```

```

<LoadBalancer>
  <RAIDb-0>
    <CreateTable policy="random" numberOfNodes="1">
      <BackendName name="node1" />
      <BackendName name="node2" />
    </CreateTable>
  </RAIDb-0>
</LoadBalancer>

```

Balanceamento de carga RAIDb-1: espelhamento completo Um elemento RAIDb-1 é definido como:

```

<!ELEMENT RAIDb-1 (WaitForCompletion?, MacroHandling?,
  (RAIDb-1-RoundRobin | RAIDb-1-WeightedRoundRobin |
  RAIDb-1-LeastPendingRequestsFirst))>

<!ELEMENT RAIDb-1-RoundRobin EMPTY>
<!ELEMENT RAIDb-1-WeightedRoundRobin (BackendWeight)>
<!ELEMENT RAIDb-1-LeastPendingRequestsFirst EMPTY>

<!ELEMENT WaitForCompletion EMPTY>
<!ATTLIST WaitForCompletion
  policy (first | majority | all) "first"
  deadlockTimeoutInMs CDATA "30000"
>

<!ELEMENT BackendWeight EMPTY>
<!ATTLIST BackendWeight
  name CDATA #REQUIRED
  weight CDATA #REQUIRED>

```

Se `WaitForCompletion` for omitido, o comportamento padrão é retornar o resultado tão logo um backend o tenha feito. `deadlockTimeout` define o tempo em milisegundos de espera antes de se comece a detecção de deadlocks. Isto deve ser, tipicamente, maior que os ajustes de espera do banco de dados. O padrão é

30000 mas é razoável que seja maior que o tempo de execução da maior consulta. 0 desabilita a detecção de deadlocks.

Se `MacroHandling` for omitido, um elemento `MacroHandling` padrão é adicionado.

`RAIDb-1` aceita política para especificar a conclusão de consultas distribuídas. Várias políticas de balanceamento de carga são propostas:

- . `RoundRobin`: balanceamento round-robin simples. A primeira requisição é enviada para o primeiro nó, a segunda para o segundo, etc... Uma vez que se tenha enviado uma requisição para o último, a próxima será enviada para o primeiro backend e assim por diante.
- . `WeightRoundRobin`: o mesmo que é feito acima mas com um peso associado a cada backend. Um backend com peso 2 recebe 2 vezes mais requisições que um backend com peso 1.
- . `LeastPendingRequestFirst`: a requisição é enviada para o backend com o menor número de requisições pendentes.

A definição do elemento `RAIDb-1` é dada como seguinte:

`WaitForCompletion` define a política a ser adotada na espera de conclusão de requisição. A política funciona da seguinte forma:

- . `first`: retorna o resultado tão logo um nó o tenha completo.
- . `majority`: retorna o resultado tão logo a maioria dos nós ($n/(2+1)$) tenha o completo.
- . `all`: espera até que todos os nós tenham retornado o resultado para o cliente.

Balanceamento de carga `RAIDb-1ec` `RAIDb-1` com verificação de erro deve prover política de verificação de erro, como definida mais abaixo. A política opcional `WaitForCompletion` apenas diz respeito a requisições de escrita (`INSERT,DELECT,UPDATE,commit,...`).

Nota: RAIDb-1ec não é operacional em Sequoia v1.alpha. A definição do elemento RAIDb-1ec é definida como:

```
<!ELEMENT RAIDb-1ec (WaitForCompletion?, ErrorChecking,
  (RAIDb-1ec-RoundRobin | RAIDb-1ec-WeightedRoundRobin))>
<!ATTLIST RAIDb-1ec
  nbOfConcurrentReads CDATA #REQUIRED>

<!ELEMENT RAIDb-1ec-RoundRobin EMPTY>
<!ELEMENT RAIDb-1ec-WeightedRoundRobin (BackendWeight)>

<!ELEMENT ErrorChecking EMPTY>
<!ATTLIST ErrorChecking
  policy (random | roundRobin | all) #REQUIRED
  numberOfNodes CDATA #REQUIRED>
```

A política de verificação de erros (RAIDb-1ec e RAIDb-2ec) é usada para detecção de erros bizarros em nós, que ocorre quando um nó envia resultados estranhos de maneira não determinística. A verificação de erros permite que consultas de leitura sejam enviadas a mais de um banco de dados e os resultados são comparados. A maioria dos nós devem estar de acordo no resultado que será enviado ao cliente. A política de verificação de erros é definida por:

- . **random**: os backends em `numberOfNodes` são tomados randomicamente; a requisição de leitura é enviada a estes backends e o resultado comparado.
- . **roundRobin**: backends de `numberOfNodes` são tomados segundo algoritmo round robin; da mesma forma a consulta é enviada aos backends e os resultados comparados.
- . **all**: a requisição é enviada a todos os backends e os resultados comparados.

`numberOfNodes` deve ser maior ou igual a 3.

Balanceamento de carga RAIDb-2: espelhamento distribuído A definição do elemento RAIDb-2 é definida como:


```
<!ELEMENT RAIDb-2 (CreateTable*, WaitForCompletion?, MacroHandling?,  
  (RAIDb-2-RoundRobin | RAIDb-2-WeightedRoundRobin |  
    RAIDb-2-LeastPendingRequestsFirst))>
```

```
<!ELEMENT RAIDb-2-RoundRobin EMPTY>  
<!ELEMENT RAIDb-2-WeightedRoundRobin (BackendWeight)>  
<!ELEMENT RAIDb-2-LeastPendingRequestsFirst EMPTY>
```

Se `MacroHandling` for omitido, um elemento padrão `MacroHandling` é adicionado.

O balanceador de carga `RAIDb-2` aceita política que especifique onde novas tabelas serão criadas e como a conclusão de consultas distribuídas é manipulada. Várias políticas de balanceamento de carga são propostas:

- . `RoundRobin`: balanceamento round-robin simples. A primeira requisição é enviada para o primeiro nó, a segunda para o segundo, etc... Uma vez que se tenha enviado uma requisição para o último, a próxima será enviada para o primeiro backend e assim por diante.
- . `WeightRoundRobin`: o mesmo que é feito acima mas com um peso associado a cada backend. Um backend com peso 2 recebe 2 vezes mais requisições que um backend com peso 1.
- . `LeastPendingRequestFirst`: a requisição é enviada para o backend com o menor número de requisições pendentes.

A definição do elemento `CreateTable` é dada na seção 10.6.4.3.

A definição do elemento `WaitForCompletion` é dada na seção 10.6.6.4

Balanceamento de carga `RAIDb-2ec` A política de verificação de erros de `RAIDb-2ec` é dada da mesma forma que para `RAIDb-1ec` (seção 10.6.4.5). Os outros elementos são similares aos definidos para controlador `RAIDb-2` (seção 10.6.4.6)

Nota: RAIDb-2ec não é operacional em Sequoia v1.0alpha.

A definição do elemento RAIDb-2ec é dada como seguinte:

```
<!ELEMENT RAIDb-2ec (CreateTable*, WaitForCompletion?, ErrorChecking,
                      (RAIDb-2ec-RoundRobin |
                       RAIDb-2ec-WeightedRoundRobin))>
<!ATTLIST RAIDb-2ec
    nbOfConcurrentReads CDATA #REQUIRED
>

<!ELEMENT RAIDb-2ec-RoundRobin EMPTY>
<!ELEMENT RAIDb-2ec-WeightedRoundRobin (BackendWeight)>
```

9.5.2 ParGRES

ParGRES[268] é um projeto que tem como objetivo desenvolver um sistema de software livre para processar com eficiência consultas pesadas que envolvam grandes quantidades de dados, usando para isso, o SGBD PostgreSQL sobre clusters de PCs. No ParGRES, o processamento da consulta explora o paralelismo intra- e inter-consultas, usando replicação e fragmentação virtual de dados.

O paralelismo do ParGRES é voltado para as consultas pesadas típicas de aplicações OLAP e propomos uma solução para essa demanda implementando paralelismo intra-consulta em um cluster de BD. O paralelismo intra-consulta significa decompor consultas complexas em sub-consultas que serão executadas em paralelo. A idéia é que cada sub-consulta atue em um fragmento de dados diferente. Dessa forma, cada sub-consulta poderá então ser enviada para o nó que possui o respectivo fragmento dos dados. Assim, cada sub-consulta é enviada para um nó diferente e executada em paralelo com as demais. Embora esteja realizando o desenvolvimento sobre o PostgreSQL, o paralelismo é baseado no padrão SQL, não sendo dependente de nenhuma característica específica do SGBD (vide [16]).

Como as demais soluções para clusters de bancos de dados, o ParGRES consiste em uma camada intermediária de software (middleware) que orquestra instâncias de SGBDs em execução nos diferentes nós do cluster para a implementação

de técnicas de processamento paralelo.

Capítulo 10

Alta Capacidade de Processamento (HPC)

Cluster de Processamento (HPC), essa categoria de cluster possui como principal característica o processamento de alta performance/desempenho, grandes usuários dessa tecnologia no Brasil são: Universidades, centros de pesquisa, Petrobras.

10.1 Beowulf

Beowulf é o nome de um projeto para cluster de computadores para computação paralela, usando computadores pessoais, não especializados e portanto mais baratos. O projeto foi criado por Donald Becker¹ da NASA, e hoje são usados em todo mundo, principalmente para programação científica.

Um cluster de Beowulf é um grupo de computadores, normalmente PCs idênticos que executam como sistema operacional o GNU/Linux ou BSD. Eles trabalham em uma rede LAN TCP/IP pequena, e tem bibliotecas e programas instalados que permitem o compartilhamento do processamento entre eles, mais informações sobre essas bibliotecas e ferramentas podem ser obtidas na sessão 11.1.

¹Mais informações sobre Donald Becker podem ser encontradas na Wikipedia http://en.wikipedia.org/wiki/Donald_Becker

Não existe nenhum software em particular que defina um cluster como Beowulf. Existem bibliotecas de processamento paralelo que geralmente são usadas no cluster Beowulf, essas bibliotecas incluem: MPI[303] (Message Passing Interface) e PVM[305] (Parallel Virtual Machine). Ambos permitem o programador dividir uma tarefa entre um grupo de computadores conectados em rede, e recolher os resultados das tarefas processadas.

Sendo assim o nome Beowulf acaba sendo a descrição de ambientes de clusters de processamento paralelo freqüentemente encontrado. Existem várias distribuições e soluções em software livre e aberto para esses ambientes. Pode-se citar como exemplos de soluções desenvolvidas para a criação de ambientes Beowulf: Oscar, Rocks, Xcat, já citadas na sessão 4.3.1.

Além das soluções que facilitam a instalação e configuração deste ambiente existem outras ferramentas de suporte a este ambiente, como ferramentas para o monitoramento, controle e execução de tarefas nos nós.

10.2 Sistema de Imagem Única (SSI)

Sistema de Imagem Única (SSI) são métodos utilizados para se esconder à complexidade dos sistemas distribuídos, permitindo que os mesmos pareçam uma única máquina ao usuário, fazendo com que fatores como a heterogeneidade do hardware implementado não seja problema para o seu funcionamento, e questões como a gerência e utilização efetiva dos recursos sejam facilmente solucionadas, dando a visão de uma máquina única, parecendo ser uma máquina SMP só que na verdade utilizando-se de um ambiente de rede distribuído, construídos de várias máquinas. [101].

Em clusters, o SSI auxilia tanto na escalabilidade quanto na disponibilidade. Os recursos de um cluster SSI são a soma dos recursos disponíveis no cluster, assim o sistema é visto como um único recurso, a adição ou subtração de alguns desses recursos não afeta potencialmente o funcionamento do cluster, permitindo muitas vezes um incremento de recursos em tempo de execução, dependendo da escalabilidade suportada, e pode também assegurar que o sistema continue funcionando após alguma falha em um ou alguns de seus nós sem que haja perdas

consideráveis, permitindo que o cluster fique sempre disponível para as aplicações do usuário.

A visualização dos nós como um SSI permite a monitoração centralizada dos recursos de cada um deles, torna-se possível um balanceamento de carga efetivo, dividindo os processos entre os nós da melhor maneira fazendo com que os recursos sejam bem utilizados. Permite também uma hierarquia globalizada com múltiplos sistemas de arquivos e espaço único de memória e de dispositivos de entrada e saída, além de um endereçamento global para os processos.

Embora a implementação do SSI ainda seja muito limitada nos clusters, já apresenta vários benefícios, e que estão evoluindo a todo o momento, permitindo um incremento tanto da escalabilidade quanto do sistema de imagem única, podendo se gerar uma estrutura cada vez mais próxima da estrutura SMP, só que com uma excelente escalabilidade. O SSI pode ser implementado através de hardware, multiprocessadores 6.1.2, ou por software, este último geralmente é feito utilizando-se de uma camada de middleware ou uma camada adicional (patch) ao kernel [104].

O middleware é composto de uma camada de infraestrutura de disponibilidade que fica acima da camada do sistema operacional e por uma camada de infraestrutura de imagem única que fica logo acima da primeira, a camada de SSI é quem faz a interface com as aplicações dos usuários. Todos os pacotes trabalham em conjunto dando melhor suporte a essas camadas, providenciando também uma eficiente implementação para DSM, checkpoint e migração de processos.

10.2.1 As Principais Características de um Cluster SSI

A seguir estão descritas as principais características que um cluster deve possuir para ser considerado um Sistema de Imagem Única, segundo Buyya [104]:

- Ponto único de acesso: garantia de transparência ao usuário da máquina ao qual o mesmo está se logando em meio a todos os nós do cluster. Desse modo diferentes nós atendem diferentes usuários, dividindo a carga apresentada pelas requisições de cada usuário de forma transparente ao mesmo.

- Interface única de usuário: os usuários devem acessar o cluster através de uma única interface gráfica de usuário, tal como uma página web onde se possam usufruir os serviços associados ao cluster. Essa interface deve possuir as mesmas características para todos os usuários.
- Espaço de processo único: todos os processos, independentemente do local onde se encontrem, devem poder se comunicar com processos de quaisquer nós; devem poder migrar para qualquer nó e podem gerar novos processos. Um cluster SSI deve permitir a administração dos processos como se estivessem sendo executados localmente, isto é, deve-se garantir o controle dos processos independentemente do local onde estejam sendo executados.
- Espaço de memória único: deve-se garantir ao usuário a ilusão de um espaço único de memória, de forma que os diversos espaços locais dos inúmeros nós que compõem o cluster fossem uma única grande memória. Para isso existem diferentes abordagens tais como o uso de software garantindo uma camada acima da memória de cada nó, simulando a existência de um único espaço de memória; o outro modo é o desenvolvimento distribuído, isto é, implementação no próprio código através do uso de bibliotecas tais como MPI ou PVM, onde o compilador do sistema onde se executa a aplicação se encarrega de distribuir a estrutura dos dados entre os diversos nós do cluster.
- Espaço de E/S único: garantir a execução de operações de entrada/saída a periféricos e discos tanto localmente quanto remotamente, de forma transparente ao usuário. Fazendo assim um único espaço de endereçamento formado pelos diversos discos associados aos nós do cluster, RAIDs anexados à rede e um exemplo.
- Hierarquia única de arquivos: os usuários ao terem acesso ao sistema de arquivos do cluster SSI devem possuir uma visão única do mesmo, de modo com que todos identifiquem os diversos discos, periféricos e diretórios sob uma mesma raiz de uma única forma. Exemplos de hierarquia única para um sistema de arquivos são o NFS, o MFS e o GFS.
- Rede virtual única: um nó deve possuir a capacidade de acessar qualquer conexão de rede através do domínio do cluster, mesmo que a rede não esteja conectada a todos os nós do cluster.
- Sistema de administração de jobs único: um job de um usuário pode ser

submetido de qualquer nó do cluster e pode requisitar quantos nós quiser para executá-lo.

- Ponto de controle e administração único: todo o cluster, isto é, cada um dos nós que o compõe devem poder ser testados, configurados e monitorados através de um único conjunto de interfaces gráficas tais como uma página web.
- Migração de processos e checkpointing: deve-se fazer periodicamente a gravação das informações dos processos tais como estado e processo ou em memória ou em disco, garantindo em caso de falha do mesmo sua continuação em outro nó. Além disso devido à necessidade de balanceamento de carga deve-se garantir a migração de processos entre os diversos pontos do cluster.

10.2.2 Os principais benefícios de um sistema SSI

Os principais benefícios que um sistema SSI proporciona ao funcionamento de um cluster, segundo Buyya [104]:

- Provê uma simples e única visão de todos os recursos e atividades em execução no cluster;
- Exclui do usuário a necessidade de apontar onde se deve executar tal aplicação;
- Garante o uso de recursos independentemente da proximidade física aos mesmos;
- Permite maior facilidade para gerenciamento do sistema pelo administrador e uso do mesmo pelo usuário já que a interface e os comandos são um só para o acesso a todos os nós;
- Reduz a possibilidade de erros por parte do operador, isto é, de por exemplo se utilizar uma sintaxe de comandos diferentes para o acesso a um mesmo nó, e outra sintaxe diferente para um outro nó, garantindo assim performance e confiabilidade ao sistema;

- Como o controle deve ser apresentado centralizado permite o uso do sistema por pessoas sem que tenham, necessariamente, elevado conhecimento sobre como funciona o sistema, permitindo seu uso por usuários “comuns”;
- Redução de gastos com a implantação/manutenção do sistema;
- Prove comunicação de mensagens independente de localização;
- Como os programadores não devem se preocupar com a distribuição da carga para suas aplicações garante mais tempo aos mesmos para aumentar a complexidade e qualidade de seus sistemas.
- Promove a padronização de ferramentas para o seu controle e administração.

10.2.3 Memória Distribuída Compartilhada (DSM)

Técnica utilizada para compartilhamento de memória física dos nós, dando a ilusão de que o cluster possui uma única e grande memória que é formada pela agregação das memórias de seus nós, implementando a abstração de um espaço comum de endereçamento agrupando as memórias isoladas em uma entidade lógica, podendo ser implementada por software ou hardware, permitindo a troca de dados através de uma memória globalizada a todos os nós processadores [351].

Com essa técnica, não há mais a necessidade de usar paradigmas de passagem de mensagens explícitas como PVM ou MPI nos programas desenvolvidos especialmente para cluster, pois os programas que utilizam memória distribuída compartilhada têm acesso a variáveis em memória compartilhada da mesma forma como se faz em variáveis locais.

Cada processador tem uma visão de toda a memória, mas só tem acesso a parte que é destinada a ele, então, caso ele queira acessar dados que não estão localizados na parte onde ele é proprietário, deve fazer uma cópia para sua área, dando origem a cópias múltiplas da mesma porção de dados da memória compartilhada em diferentes memórias físicas, tendo assim, que manter a coerência destas cópias, permitindo que qualquer processador que acesse a memória compartilhada devolva a informação correta, sendo a comunicação e a sincronização feita através da própria memória.

Para se reduzir à latência de memória, ou seja, os tempos de acesso à memória e retorno da resposta, as caches privadas a cada processador são utilizadas, pois em sistemas DSM há uma grande sobrecarga com a localização dos dados e acesso a memória distribuída, ficando esses caches com a função de buffers temporários, para que não se desperdice ciclos do processador.

Quando um programa é feito, existe uma ordem específica de acesso a memória, chamada consistência seqüencial. Num cluster essa consistência é inviável pois os nós teriam que esperar a sua vez de acessar a memória para assim continuar processando, isso geraria perda de desempenho e tráfego excessivo na rede. Para isso foram desenvolvidos modelos de consistência que levam em consideração que nem todos os nós processadores precisam naquele momento do conteúdo atualizado.

Esses modelos implementam métodos que tentam evitar ao máximo as condições de corrida, ou seja, acesso simultâneo ao mesmo dado. Cada nó possui um mapeador de memória local onde ela é particionada em blocos, sendo a cache utilizada para reduzir a latência de acesso à memória remota e sendo a memória principal dos nós um cache da DSM, formando assim uma hierarquia simples de memória, ou seja, cada nó enxerga sua memória local como uma cache da DSM sendo cada bloco da memória a unidade básica de cache.

10.2.4 OpenMosix

O openMosix é um middleware para clustering “open-source” que possui dois módulos de grande valia ao serviço de clustering: um patch para memória compartilhada distribuída (migshm) e um módulo para checkpointing.

Memória compartilhada é usada na maior parte das aplicações modernas, tais como bancos de dados, servidores WEB, editores de texto, planilhas, e processamento de imagens. A operação de compartilhamento de memória ajuda a facilitar a troca de dados entre os processos.

Por exemplo, quando dois clientes fazem um select e um update em uma coluna de uma tabela MySQL, o MySQL deve criar dois subprocessos e servir os comandos SQL. Os processos “forked” se anexam ao processo MySQL principal que

mantém os dados em memória. Usando esse esquema, não existe a necessidade de se criar um “data mirror” e plugá-lo de volta à tabela real.

O benefício é que o servidor de banco de dados por reduzir a alocação de memória. Evidentemente, um mecanismo de “locking” é necessário para evitar um “deadlock” ou uma condição de corrida. Uma “thread” é uma versão leve de um mecanismo fork(). Ao invés de duplicar todo um segmento de memória de um processo pai para um processo filho, o processo recémcriado apenas precisa inicializar um pequeno segmento de dados e se anexar ao processo principal. Essa técnica (também chamada de “LightWeight Process”, ou LWP) oferece uma nova maneira de multiprocessamento usando o mínimo possível de memória. O apache, servidor WEB, utiliza threads para aumentar a velocidade de suas tarefas de forma a permitir aumentar o número de hits sem afetar sua performance.

O Migshm é um patch DSM (Distributed Shared Memory) para o openMosix. Ele permite a migração de processos que utilizam memória compartilhada no openMosix tais como o apache, entre outros.

“Checkpointing” é uma técnica que provê a possibilidade de se salvar contextos de processos em um arquivo de disco e dar um restore dos processos a partir do arquivo. Logo processos que tenham sofrido “checkpointing” e que sejam reiniciados posteriormente deveriam funcionar como se não tivessem sido interrompidos. Essa funcionalidade é útil para tarefas que possuam longos períodos de execução (como simulações numéricas) no caso de instabilidade de sistema, falhas de energia, reboot’s, etc. “Checkpointing” costuma ser um serviço de sistemas operacionais avançados de clustering.

O CHPOX (Checkpoint for Linux) é um módulo do kernel Linux que provê “checkpointing” de processos. Ele foi criado por Olexander O. Sudakov e Eugeny S. Meshcheryakov no Information and Computing Center, National Taras Shevchenko University, Kyiv, Ucrânia. O CHPOX usa uma abordagem de módulo de kernel, logo ele é pouco relacionado à versão do kernel atual e é dinamicamente inserido e removido do espaço de kernel. Devido à sua integração com o Mosix/openMosix, o CHPOX se tornou rapidamente aceito pela comunidade openMosix.

Tecnologia openMosix O openMosix é um conjunto de algoritmos para compartilhamento dinâmico de recursos que são utilizados para fornecer escalabilidade e performance em um cluster CC (Cache Coherent) de qualquer tamanho, onde o único componente compartilhado é a rede. A idéia principal da tecnologia do openMosix é a capacidade de múltiplos nós (workstations e servidores, incluindo SMP's) de trabalharem em cooperação como parte de um sistema único.

Com o sentido de compreender o que o openMosix faz, devemos comparar multicomputador de memória compartilhada (SMP) a um CC. Em um sistema SMP, múltiplos processadores compartilham a memória. As principais vantagens são o aumento do volume de processamento e a maior velocidade de comunicação entre os processos (através da memória compartilhada). Máquinas SMP podem suportar múltiplos processos trabalhando simultaneamente, com eficiente alocação e compartilhamento de recursos. A qualquer momento que um processo seja inicializado, finalizado, ou mude seu perfil computacional, o sistema se adapta instantaneamente ao ambiente de execução resultante. O usuário não é envolvido diretamente e, na maior parte dos casos, nem sabe da existência de tais atividades.

Ao contrário do SMP, Computing Clusters (CC) são feitos de coleções de servidores (até mesmo SMP's) e workstations que fisicamente nada compartilham, com diferentes velocidades e quantidades de memória, possivelmente de diferentes gerações. Na maioria das vezes, CC's são utilizados para ambientes "time-sharing" e multiusuário. Em sistemas CC o usuário é responsável por alocar os processos aos nós e administrar os recursos do cluster. Em vários sistemas CC, mesmo estando todos os nós utilizando o mesmo sistema operacional, a cooperação entre os nós é consideravelmente limitada pois a maioria dos serviços do sistema operacional são localmente confinadas ao nó.

Os principais pacotes de software para alocação de processos em CC's são PVM e MPI. Esses pacotes provêm um ambiente de execução que requer uma adaptação da aplicação e o conhecimento do usuário. Eles incluem ferramentas para alocação inicial (fixa) de processos para nós, os quais algumas vezes utilizam considerações sobre a carga, enquanto ignoram a disponibilidade de outros recursos tais como memória livre e "overheads" de E/S. Esses pacotes ficam na camada de usuário, assim como as aplicações comuns, entretanto são incapazes de responder a mudanças no nível de carga ou mesmo de outros recursos, ou até de

redistribuir a carga do trabalho (job) de forma adaptativa.

Na prática, o problema de alocação de recursos é muito mais complexo pois existem vários tipos diferentes de recursos, tais como CPU, memória, E/S, IPC (Comunicação Inter-Processos), etc, onde cada recurso é utilizado de uma maneira diferente e na maior parte das vezes seu uso é imprevisível. Outras dificuldades surgem do fato que diferentes usuários não coordenam suas atividades. Entretanto mesmo que um usuário saiba otimizar a alocação de recursos aos processos, as atividades de outros usuários costumam interferir em sua otimização.

Para o usuário, os sistemas SMP garantem eficiência, uso balanceado de recursos entre os processos em execução, independentemente dos requisitos de recurso. SMP's são fáceis de usar pois eles empregam administração adaptativa de recursos, o que é completamente transparente ao usuário. Os atuais CC's não possuem tais capacidades. Eles se baseiam na alocação estática controlada pelo usuário, o que é inconveniente e pode levar a significativas perdas de performance devido a cargas mal distribuídas.

O openMosix é um conjunto de algoritmos que juntos suportam compartilhamento adaptativo de recursos em um CC escalável pela migração dinâmica de processos. Ele pode ser visto como uma ferramenta que torna plataformas CC mais próximas de ambientes SMP. Ao ter a capacidade de alocar recursos globalmente, e distribuir o "workload" dinamicamente e de forma eficiente acaba por simplificar o uso de CC's ao tirar do usuário a responsabilidade de administrar os recursos do cluster. Isso é particularmente evidente em ambientes multi-usuários e "time-sharing", além de CC's não uniformes.

10.2.5 Kerrighed

Kerrighed é uma base operacional para sistemas de imagem única (Single System Image Operating System (SSI OS)) destinado para cluster construídos a partir de PCs padrões de mercado. Um sistema operacional SSI dá a ilusão de uma máquina de SMP aos programas que são executados em cima dele. Kerrighed é uma extensão kernel do Linux. Não obstante, um cluster rodando Kerrighed não é uma máquina de SMP física real.

As metas do Kerrighed são alto desempenho de aplicações, alta disponibilidade do cluster, administração de recursos eficiente, alta customizabilidade do sistema operacional e facilidade de uso.

Kerrighed é implementado como uma extensão a sistema operacional GNU/Linux (uma coleção de módulos e um pequeno “patch para o kernel Linux).

As principais características do Kerrighed são:

- Escalonador customizável para o cluster.
Processos e threads são automaticamente escalonados através dos nós do cluster para balancear o uso de CPU através do escalonador padrão do Kerrighed. Porém, Kerrighed oferece um toolkit para escrever escalonadores sob encomenda com facilidade, que podem ser adicionados “a quente” nos módulos do kernel.
- Memória Compartilhada.
Threads e segmentos de memória do sistema podem ser operados através do cluster, como em uma máquina SMP.
- Mecanismos de migração de fluxo de alta performance.
Podem ser migrados processos que usam fluxos (socket, pipe, fifo, char device, etc) sem penalidade no desempenho de comunicação depois de migração.
- Sistema de arquivo distribuído.
Um único espaço de nome de arquivo é visto no cluster. Todos os discos do cluster são fundidos em um único disco virtual em uma customização parecida como um RAID.
- Verificação de processos.
Os processos podem ser verificados e reiniciados em qualquer um nó do cluster.
- Interface de Thread Posix completa.
A interface de Thread Posix pode ser operada com threads espalhadas pelos nós do cluster.
- Interface de processos Unix visível em todo o cluster.
Toda a interface tradicional de comandos de gerenciamento de processos

Unix (top, ps, kill, etc) são operados pelo cluster. Além disso, os identificadores de processos (pid) são únicos no cluster.

- Características customizáveis da imagem única de sistema.
As características do SSI (memória compartilhada, escalonador global, migração de fluxos, etc) podem ser ativados ou não por base de processos.

Kerrighed não é:

- Um paralelizador automático.
Kerrighed não paraleliza automaticamente suas aplicações. Isso implica que caso você tenha um grande processo seqüencial, ele não rodará mais rápido no Kerrighed. Para rodar mais rápido, sua aplicação precisará ser paralelizada. Se sua aplicação roda mais rapidamente em uma máquina com vários processadores do que em uma máquina com apenas um processador, essa aplicação poderá rodar mais rápido com o Kerrighed. Se a aplicação não roda mais rápido em uma máquina com vários processadores, ela não rodará mais rápido com o Kerrighed.
- Um Middleware.
Kerrighed é um sistema operacional, e não um middleware. Ele roda dentro do kernel linux, e não em cima do kernel linux. Ele estende as funcionalidades do linux de gerenciamento de serviços de cluster.
- Uma máquina virtual.
Kerrighed não tem correlação nenhuma com tecnologias de virtualização como VMWare ou XEN. Ele não cria um cluster virtual, ele dá a ilusão que um cluster físico de PCs são uma máquina SMP única.

Capítulo 11

Ferramentas de Programação Paralela

11.1 Troca de Mensagens (Message Passing)

O paradigma de troca de mensagens tem sido tradicionalmente empregado em sistemas fracamente acoplados, representados pelas arquiteturas baseadas em memória distribuída (clusters), em que cada processador tem acesso somente à sua memória local e, por conseguinte, a comunicação, necessariamente, dá-se por meio de uma rede de interconexão.

Conceitualmente, a idéia de troca de mensagens é totalmente independente de hardware, sistema operacional, linguagens de programação e bibliotecas. Quando do desenvolvimento de um programa paralelo por troca de mensagens, o programador deve distribuir os dados explicitamente entre os processadores. Visto que os espaços de endereçamento dos processos que compõem o programa paralelo são distintos, concebeu-se a abstração de mensagem, que pode ser enviada de um processo a outro por um canal de comunicação. Tal conceito é denotado no programa através de primitivas do tipo *send* e *receive*, as quais supõem um processo que pretende enviar (*send*) uma mensagem a outro, que espera recebê-la (*receive*). Entre os processos comunicantes diz-se que existe um canal de comunicação.

Na prática, os programadores dispõem de bibliotecas de comunicação com primitivas à semelhança de *send* e *receive*. As bibliotecas de comunicação que obtive-

ram maior aceitação foram: MPI (Clarke [121]) e PVM (Geist [166]), ambas com suporte às linguagens de programação C e Fortran.

11.1.1 PVM

O PVM[305] é um pacote de software que permite o funcionamento de um conjunto de máquinas mono/multi-processadas e/ou paralelas, como um único recurso computacional paralelo. Com a utilização do PVM consegue-se uma forma eficiente e transparente de distribuir tarefas entre máquinas ligadas em rede, conseguindo um bom desempenho na gerência dos recursos computacionais, através da configuração de uma máquina paralela virtual.

Características

O PVM habilita uma coleção de computadores heterogêneos a se comportar como um único recurso computacional expansível e concorrente, o que contribuiu para torná-lo um padrão de fato. É um mecanismo de distribuição livre que oferece bastante recursos para computação paralela com uma utilização simples e de fácil compreensão.

Algumas características do PVM são:

- possibilita a atribuição das subtarefas de uma aplicação, de forma otimizada, aos nós que compõem o ambiente paralelo;
- apresenta uma interface de programação intuitiva e consistente;
- oferece suporte para tolerância à falhas, monitoração e profiling e
- é altamente “portável”.

Com isso, através da agregação e compartilhamento de processadores e memórias de computadores heterogêneos, problemas que exigem grande poder computacional podem ser resolvidos sem a necessidade de comprar um supercomputador. Assim, utilizar o PVM é uma forma de aumentar o desempenho, com um custo efetivo menor.

Funcionamento Básico .

O PVM é composto de duas partes. A primeira parte é um daemon, chamado pvmd3, que é executado em todos os computadores que vão formar a máquina virtual paralela (MORETTI[278]). Esse programa roda em background em cada um dos nós formando a máquina virtual, sendo responsável pela troca de mensagens entre eles além da coordenação das tarefas em execução.

A segunda parte é uma biblioteca de rotinas de interface, na qual se encontra um conjunto completo de primitivas, necessárias para a interação entre as tarefas de uma aplicação. As rotinas responsáveis pela comunicação entre os computadores interligados, gerência de processos, coordenação das tarefas, além da verificação e manutenção de estado da máquina virtual, estão contidas nessa biblioteca.

Os programas paralelos que utilizam o PVM fazem uso das rotinas disponibilizadas na biblioteca de interface do PVM. Para programação, essas bibliotecas são distribuídas em linguagens como Java, Python, Perl, além das linguagens tradicionais como C, C++ e Fortran.

Ao utilizar uma aplicação PVM, o usuário executa o pvmd3 em um dos computadores do cluster, normalmente o nó mestre, que chama os demais processos pvmd3 escravos em cada computador que vai compor a máquina virtual, através de remote shell - rsh, coordenando assim as comunicações entre os processadores e o sistema. Logo, em cada nó, é necessário ter o pvmd3 instalado, sendo que existem versões disponíveis para vários sistemas operacionais. No caso de transmissão entre arquiteturas diferentes, automaticamente é feita uma conversão dos dados pelo formato XDR (External Data Representation), conforme RFC 1832.

11.1.2 Message Passing Interface (MPI)

Segundo Gropp et al. [205], Foster [179] e Pacheco [295], o MPI é um padrão de interface para a troca de mensagens em máquinas paralelas com memória distribuída e não se devendo confundi-lo com um compilador ou um produto específico.

Histórico do MPI .

O MPI é o resultado do esforço de aproximadamente 60 pessoas, pertencentes a 40 instituições, principalmente dos Estados Unidos e Europa. A maioria dos fabricantes de computadores paralelos participou, de alguma forma, da elaboração do MPI, juntamente com pesquisadores de universidades, laboratórios e autoridades governamentais. O início do processo de padronização aconteceu no seminário sobre Padronização para Troca de Mensagens em ambiente de memória distribuída, realizado pelo Center for Research on Parallel Computing, em abril de 1992. Nesse seminário, as ferramentas básicas para uma padronização de troca de mensagens foram discutidas e foi estabelecido um grupo de trabalho para dar continuidade à padronização. O desenho preliminar, foi realizado por Dongarra, Hempel, Hey e Walker, em novembro 1992, sendo a versão revisada finalizada em fevereiro de 1993 (pode ser obtido em <ftp://netlib2.cs.utk.edu/mpi/mpil.ps>).

Em novembro de 1992 foi decidido colocar o processo de padronização numa base mais formal, adotando-se o procedimento e a organização do HPF (the High Performance Fortran Forum). O projeto do MPI padrão foi apresentado na conferência Supercomputing 93, realizada em novembro de 1993, do qual se originou a versão oficial do MPI (5 de maio de 1994).

Ao final do encontro do MPI-1 (1994) foi decidido que se deveria esperar mais experiências práticas com o MPI. A sessão do Forum-MPIF de Supercomputing 94 possibilitou a criação do MPI-2, que teve início em abril de 1995. No Supercomputing'96 foi apresentada a versão preliminar do MPI-2. Em abril de 1997 o documento MPI-2 foi unanimemente votado e aceito. Este documento está disponível via HTML em <http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html>

O Padrão MPI .

No padrão MPI, uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Inicialmente, na maioria das implementações, um conjunto fixo

de processos é criado. Porém, esses processos podem executar diferentes programas. Por isso, o padrão MPI é algumas vezes referido como MPMD (multiple program multiple data).

Elementos importantes em implementações paralelas são a comunicação de dados entre processos paralelos e o balanceamento da carga. Dado o fato do número de processos no MPI ser normalmente fixo, neste texto é enfocado o mecanismo usado para comunicação de dados entre processos. Os processos podem usar mecanismos de comunicação ponto a ponto (operações para enviar mensagens de um determinado processo a outro). Um grupo de processos pode invocar operações coletivas (collective) de comunicação para executar operações globais. O MPI é capaz de suportar comunicação assíncrona e programação modular, através de mecanismos de comunicadores (communicator) que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna.

Os algoritmos que criam um processo para cada processador podem ser implementados, diretamente, utilizando-se comunicação ponto a ponto ou coletivas. Os algoritmos que implementam a criação de tarefas dinâmicas ou que garantem a execução concorrente de muitas tarefas, num único processador, precisam de um refinamento nas implementações com o MPI.

11.2 Relações Entre o Hardware e o Software para Exploração do Paralelismo

Esta sessão tem por objetivos caracterizar os pontos de interdependência entre o software e o hardware para paralelismo, e analisar as características de um modelo ideal de programação, buscando delimitar as condições necessárias para que se potencialize o desenvolvimento de programas destinados às arquiteturas paralelas.

11.2.1 Relação entre Algoritmos e Arquiteturas Paralelas

O foco central desta seção é discutir a relação entre as características de um algoritmo paralelo e aquelas da arquitetura sobre a qual este irá ser processado. Uma clareza deste inter-relacionamento é fundamental para a qualidade, tanto do projeto do hardware paralelo, como dos respectivos softwares a nível de sistema e aplicativo. Este tema vem despertando o interesse dos teóricos já há algum tempo. Um primeiro estudo publicado foi o de UNG [367]. A relação dos critérios para inter-relacionamento que será utilizada foi extraída de MOLDOVAN [277]. O tema tratado nesta seção é amplo e factível de uma abordagem teórico-formal. Foi escolhido um tratamento de modo resumido, procurando mostrar da maneira mais genérica possível a relação entre o hardware e o software paralelo (vide tabela 11.1). Para potencializar esta escolha, foram extraídos dos textos CULLER [128], HWANG [222] e MORSE [280] exemplos para quantificar a natureza das relações.

Crítérios para Caracterização de Algoritmos

O critérios que serão utilizados para caracterizar o perfil do algoritmo paralelo são: granulosidade do módulo, controle da concorrência, mecanismo de dados, geometria das comunicações e tamanho do algoritmo.

Granulosidade do módulo: os módulos podem ser programas, processos, rotinas ou instruções, dependendo do nível no qual o paralelismo está sendo explorado. A granulosidade do módulo indica o volume de computação que o mesmo contém. É relativamente usual existir abundância de paralelismo com baixa granulosidade, o qual, via de regra, não pode ser explorado com eficiência. Isto porque o trabalho com baixa granulosidade aumenta o volume de comunicação de dados entre os módulos. O desempenho da arquitetura paralela é obtido através de um equilíbrio entre granulosidade e comunicação. As comunicações, além do custo de tempo que lhes é inerente (o qual é dependente da rede de interconexão dos processadores), normalmente estabelecem sincronizações entre processos.

Controle da concorrência: diz respeito à estratégia de selecionar módulos para execução. A estratégia de controle deve considerar as dependências de dados e controle para que a execução do algoritmo seja correta. Algumas estratégias de

Algoritmo Paralelo	Arquitetura Paralela
Granulosidade do módulo	Complexidade do processador
Controle da concorrência	Modo de operação
Mecanismo de dados	Estrutura da memória
Geometria das comunicações	Rede de interconexão
Tamanho do algoritmo	Número de processadores e tamanho da memória

Tabela 11.1: Relação entre as características do hardware e do software paralelo

controle são executadas com base na disponibilidade dos dados (*dataflow*), controle centralizado (*synchronized*), ou sob demanda (*demand-driven*). Algoritmos com um alto grau de regularidade, por exemplo, multiplicação de matrizes, são indicados para um controle centralizado e são otimamente mapeados em processadores sistólicos ou outros processadores matriciais (SIMD). Por sua vez, algoritmos que incorporam transferências condicionais ou outras irregularidades no fluxo de execução, são melhor indicados para arquiteturas assíncronas tais como os multiprocessadores.

Mecanismo de dados: refere-se à maneira como os operandos das instruções são manipulados. Os dados gerados por uma instrução podem tanto ser utilizados como “dados puros”, padrão do modelo de controle por disponibilidade de dados (*dataflow*), ou podem ser colocados em um lugar de armazenamento, e então referenciados pelo seu endereço, como no modelo de Von Neumann e suas extensões.

Geometria das comunicações: trata do padrão como ocorrem as interconexões entre os módulos em execução. A geometria das comunicações de um algoritmo é dita regular quando o padrão das interconexões repete ao longo dos ciclos da computação, e irregular quando as interconexões são aleatórias. É comum geometrias regulares assumirem formas semelhantes a árvores, malhas e cubos.

Tamanho do algoritmo: indica o número total de computações que o algoritmo deve executar. Por exemplo, a manipulação de matrizes 1000 x 1000 é considerado um problema grande. O tamanho do algoritmo diz respeito ao número de processadores e à disponibilidade de memória da arquitetura paralela.

Crítérios para Caracterização das Arquiteturas Paralelas

Segundo a proposta de MOLDOVAN [277], as arquiteturas dos computadores paralelos podem ser caracterizadas pelos seguintes critérios: complexidade do processador, modo de operação, estrutura da memória, rede de interconexão e número de processadores/tamanho da memória.

Complexidade do processador: trata do poder computacional e da estrutura interna de cada elemento de processamento. Sistemas cujos processadores tem capacidade idêntica são ditos homogêneos. Aqueles sistemas cujos processadores são diferentes, ou são direcionados para realizar funções específicas, são ditos heterogêneos. A complexidade do processador varia bastante de uma classe de arquitetura para outra. Em processadores sistólicos, por exemplo, as células de processamento são simples, e os dados são apenas processados, nunca armazenados. Em processadores matriciais (SIMD), alguma memória precisa estar associada ao elemento de processamento.

Em multiprocessadores, por sua vez, cada nodo de processamento pode incorporar memória local, memória cache, e gerenciamento de memória. A complexidade do processador está associada à granulosidade do algoritmo. Arquiteturas de grão-elevado tem poucos processadores bastante poderosos; um exemplo típico é a conhecida arquitetura do Cray X-MP, a qual atinge no máximo quatro processadores, porém extremamente poderosos. Um exemplo de arquitetura de grão-médio é o multiprocessador Cedar que emprega oito *clusters* do Alliant FX-8, cada *cluster* com oito processadores. Uma arquitetura de grão-fino utiliza um grande número de processadores pequenos. Uma representante típica desta arquitetura é a *Conection Machine* que atinge 64.536 pequenos processadores.

Modo de operação: refere-se tanto ao controle de instruções como à manipulação dos dados. O modo tradicional de operação é comando-por-fluxo (*command-flow*), assim chamado porque a seqüência de eventos é controlada por comandos derivados do fluxo de instruções. Outro método é disparar as operações à medida que os seus operandos tornam-se disponíveis, de acordo com o modelo comando-por-dados (*dataflow*); neste caso, o controle é determinado pela disponibilidade dos dados. Outra alternativa de controle ainda, é o comando-por-demanda (*demand-flow*), no qual as computações ocorrem somente se seus resultados são solicitados por outras. É possível a combinação destes modos de controle. O modo

de operação da arquitetura é relacionado com o modo de controle da concorrência do algoritmo.

Estrutura da memória: refere-se ao modo de operação e à organização da memória do hardware paralelo. A memória pode ser acessada utilizando tanto endereços como conteúdo de dados (como nas memórias associativas). Em arquiteturas não convencionais, como as orientadas a conexão (*Connection Machine*) ou redes neurais, a memória consiste de interconexões ponderadas, cujo peso relativo indica a alternativa a ser utilizada. Nas arquiteturas convencionais, a organização e o tamanho da memória estão fortemente associadas ao mecanismo de dados utilizado pelo algoritmo.

Rede de interconexão: diz respeito às conexões de hardware entre processadores, bem como destes com os bancos de memória. Considerando o desempenho, a rede de interconexão deve ser o mais semelhante possível à geometria das comunicações do algoritmo paralelo. Computadores paralelos com redes de interconexão simples e/ou fixas conseguem ser eficientes para um determinado conjunto de tipos de algoritmos. Redes de interconexão complexas, ao contrário, podem ser configuradas para atender uma ampla faixa de aplicações; naturalmente isto torna o hardware mais caro, e também possivelmente irá crescer o custo (*overhead*) decorrente de um número maior de comutações no caso de uma rede reconfigurável dinamicamente.

Número de processadores e tamanho da memória: indica quantos processadores o sistema paralelo contém, e o tamanho da memória disponível. Para a tecnologia atual, e considerando apenas o número de processadores e não o seu poder computacional, sistemas com 1 a 100 processadores são considerados pequenos, sistemas com 100 a 1000 processadores são considerados médios, e sistemas com mais de 1000 processadores são considerados grandes ou muito grandes. Como regra geral, um número maior de processadores confere à arquitetura um maior poder computacional, o que faculta ao sistema paralelo trabalhar com problemas mais complexos.

Quando o tamanho do algoritmo é maior que o tamanho do sistema, se faz necessário particionar o mesmo durante a execução; isto implica que à medida que os módulos de processamento (vide granulosidade do módulo) são atribuídos aos processadores e às memórias, os resultados intermediários precisam ser armazenados. O particionamento de algoritmos pode introduzir efeitos colaterais

(*side-effects*) indesejáveis. Do ponto de vista ideal, o número de processadores deve ser compatível com o tamanho do algoritmo.

A abordagem desta seção enfatizou que o processamento com desempenho otimizado em arquiteturas paralelas, exige uma sintonia entre as características do hardware e do software. A próxima seção deste capítulo, objetiva apresentar as propriedades necessárias em um modelo de programação paralela, para que o esforço do programador para obter esta sintonia seja minimizado.

11.2.2 Propriedades de um Modelo de Programação para o Processamento Paralelo

Uma alternativa para a situação de falta de portabilidade e curto tempo de vida útil do software paralelo é o desenvolvimento de um modelo que seja abstrato o suficiente para ocultar os aspectos da arquitetura à medida que eles se alteram, ao mesmo tempo que mantém o desempenho da aplicação paralela desenvolvida. Em essência, um modelo deste tipo contempla uma máquina abstrata, para a qual o software seria desenvolvido, e que seria eficientemente emulada nas diferentes arquiteturas paralelas. Assim, este modelo atuaria como uma fronteira entre as arquiteturas paralelas sempre em evolução, e o desenvolvimento de software com sua exigência de vida longa, desassociando os aspectos do projeto do software, daqueles de sua implementação. A figura 11.1 resume esta proposta.

Nesta seção serão discutidos os aspectos de um modelo ideal para o desenvolvimento de software paralelo. A organização adotada foi proposta em SKILLICORN [331]. Os recursos mínimos que este modelo deve oferecer são os seguintes:

Facilidade para o Desenvolvimento do Software

Os aspectos pertinentes ao controle da execução paralela são bastante complexos. Na medida do possível, o modelo deve retirar do programador a responsabilidade sobre os mesmos. Para tanto, deve ficar ao encargo do compilador e do ambiente de execução, a inserção e a gerência dos mecanismos necessários.

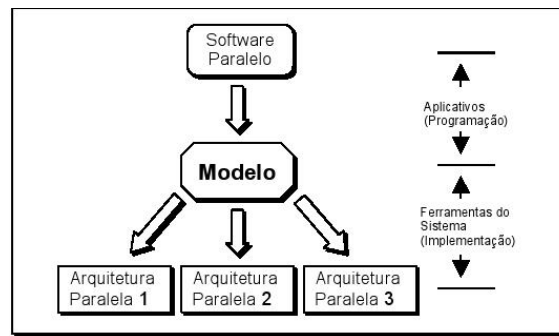


Figura 11.1: Modelo Para Computação Paralela

Assim, o modelo deve prover:

- decomposição do programa em tarefas paralelizáveis. Para sua execução paralela, o programa deve ser dividido em partes passíveis de serem executadas concorrentemente pelos diversos processadores da arquitetura. Isto implica em fracionar o código e a estrutura de dados em partes, cujo número e tamanho são função das características da arquitetura;
- mapeamento das tarefas paralelas aos processadores. Uma vez que o programa tenha sido dividido, se faz necessária a decisão de em qual processador será alocada cada tarefa. Um dos principais critérios para decisão é o volume de comunicações, de tal forma que tarefas com volume de comunicações elevado entre si tenham posições o mais próximo possível na rede de interconexões. No caso de arquiteturas heterogêneas (vide item 11.2.1), pode ser muito significativo para o desempenho levar em conta as características do processador no momento da alocação de uma determinada tarefa;
- comunicação entre as tarefas paralelas. Sempre que uma tarefa necessita de um dado não disponível localmente, algum mecanismo de comunicação deve ser ativado para obtê-lo. A forma específica de atuação do mecanismo é dependente da arquitetura, porém é indispensável que as tarefas paralelas que trocam dados tenham suporte para tal, evitando atrasos no processamento em função de dados que custam a vir, ou que eventualmente nem mesmo virão;
- sincronização entre as tarefas paralelas. É relativamente usual no processamento paralelo, duas ou mais tarefas precisarem confirmar se todo o grupo já atingiu determinado ponto comum da computação. Neste caso, também

a especificidade do mecanismo que será utilizado é fortemente dependente da arquitetura. O tema sincronização é bastante amplo e complexo. Existe na relação entre comunicação e sincronização um grande potencial para inconsistências no estado de execução (*deadlocks*).

Uma Metodologia para o Desenvolvimento de Software

O recurso previsto no item anterior (11.2.2), deixa clara a distância semântica entre a informação a ser disponibilizada pelo programador e a necessária para execução do programa em paralelo. Para superar isto, se faz necessária uma sólida fundamentação semântica, sobre a qual as técnicas de transformação de código possam ser aplicadas.

A usual metodologia de testes e depuração utilizada na programação sequencial, se mostra inviável para o desenvolvimento de software paralelo, sobretudo pelos seguintes motivos:

- o particionamento e o mapeamento das tarefas, os quais podem em alguns casos depender dos dados, ampliam a um ponto tal a complexidade da análise dos possíveis estados da computação paralela que seu uso efetivo é praticamente inviável;
- a equipe de desenvolvimento teria acesso apenas a algumas das arquiteturas paralelas nas quais o software poderia vir a ser executado. Este aspecto fica reforçado pela heterogeneidade que as arquiteturas paralelas podem assumir, bem como pelo constante surgimento de arquiteturas com novas tecnologias no mercado.
- a existência, nas arquiteturas paralelas, de componentes que dificultam a reprodução exata de execuções, como por exemplo o não determinismo inerente à maioria das redes de interconexão.

Como a comprovação do comportamento dos possíveis estados do software paralelo após seu desenvolvimento, se mostra complexo demais para uso prático, somente um processo, que leve na direção de um software correto por metodologia de construção, colocará o desenvolvimento de software em um patamar de segurança e conforto razoáveis (vide item 5.1.8).

Independência de Arquitetura

O modelo deve ser independente de arquitetura, de tal forma que os programas possam migrar entre as arquiteturas paralelas, sem exigir alterações no código ou outras modificações não triviais. Por outro lado, as arquiteturas paralelas, como aglutinam diversas tecnologias, todas em constante evolução, tem sua vida útil efetiva de poucos anos. Isto torna pouco razoável considerar a rescrita do software a cada necessidade de trocar o hardware.

Atender este aspecto de independência de arquitetura de forma individual é uma tarefa factível. Existem diversos modelos cujo nível de abstração satisfazem este aspecto (vide item [11.3.1](#)).

O complexo é atender este requisito em conjunto com os outros, que um bom modelo para o desenvolvimento de aplicações paralelas deve suprir.

Facilidade para Ser Entendido

Um modelo, para que se dissemine largamente, deve ser fácil de ser entendido. Se o processamento paralelo pretende ampliar sua fatia de mercado, o modelo para sua programação deve poder ser assimilado com facilidade pelos desenvolvedores, oferecendo para isto uma interface que oculte ao máximo a complexidade inerente ao paralelismo e seja de uso simples.

Garantia de Desempenho

Um modelo deve garantir o desempenho dos programas para ele desenvolvidos, nos diversos tipos de arquiteturas disponíveis. Segundo SKILLICORN [\[331\]](#), somente um modelo com otimização nas comunicações pode ter previsibilidade no desempenho. Por sua vez, considerando a diversidade dos problemas reais, heurísticas complexas para otimizar o mapeamento das tarefas paralelas, considerando o custo das comunicações, podem introduzir custo computacional elevado. Assim, somente modelos que limitem a frequência das comunicações, podem esperar ter uma garantia mínima de desempenho em diferentes hardwares

paralelos.

Medidas de Custo

O desenvolvimento de software para arquiteturas paralelas tem sempre presente a premissa de buscar o maior desempenho possível, o que tem reflexos diretos no tempo que o programa exige do hardware para completar sua execução.

Além do desempenho global, a taxa de utilização individual dos processadores, e o custo de desenvolvimento, são importantes indicadores para análise do custo total decorrente das etapas de projeto, programação e execução do software paralelo.

A proporcionalidade de desempenho entre as plataformas seqüenciais, faculta que uma análise da complexidade dos algoritmos empregados no programa traga uma expectativa de desempenho, independentemente do hardware específico que será utilizado. Na programação paralela tal situação não ocorre; pequenas modificações no programa ou a troca do hardware paralelo podem mudar completamente o comportamento do programa (vide item [11.2.1](#)).

Segundo SKILLICORN [\[331\]](#) um modelo oferece medidas de custo se for possível determinar o custo de um programa, em particular a partir de:

- código fonte do programa;
- propriedades mínimas da arquitetura paralela destino (por exemplo, o número de processadores);
- informações a respeito do tamanho dos dados de entrada (não os seus valores);

No texto SKILLICORN [\[331\]](#), também é caracterizada a importância de considerar os aspectos de modularidade. É relativamente usual que o software de grande porte seja desenvolvido por módulos, e possivelmente cada módulo por equipes diferentes. Isto implica que a medida de custo precisa ter um comportamento composicional, isto é, o custo total possa ser inferido a partir do custo das partes.

Considerando as outras características indispensáveis para um modelo de programação paralela, por exemplo, a necessidade de abstração tratada no item 11.2.2 a medida de custo se torna uma característica difícil de ser atingida.

11.3 A Exploração do Paralelismo: Níveis de Abstração e Modelos

O objetivo deste capítulo é caracterizar a potencialidade para exploração do paralelismo em diferentes modelos de programação.

A classificação utilizada tem como critério o nível de abstração com que a exploração pode ser feita. Estes níveis de abstração estão sugeridos em SKILLICORN [331] e BAL [74]. A partir de textos específicos sobre os modelos, foram feitas considerações, e selecionados exemplos de sistemas paralelos.

11.3.1 Modelos nos quais o Paralelismo é Explorado de Forma Totalmente Implícita

Nestes modelos, o programador descreve somente o propósito da computação, o que o programa deve fazer, e não como deve ocorrer o processamento para que o programa atinja seu propósito.

O desenvolvimento de software não precisa levar em consideração se o programa irá executar em paralelo ou não. Em função disto, estes modelos trabalham em um nível de abstração elevado, e os programas desenvolvidos pensando no paralelismo não são necessariamente mais complexos que os elaborados para execução seqüencial.

Como exemplos característicos deste nível de abstração na exploração do paralelismo, destacam-se a programação funcional e a programação em lógica.

```
fib n
| n <= 1 = 1
| otherwise = 1 + fib(n-1) + fib(n-2)
```

Figura 11.2: Números de Fibonacci em Programação Funcional

Programação Funcional

A programação funcional é uma alternativa elegante de programação declarativa, na qual o programa é definido por um conjunto de equações e funções, cujo resultado da computação é a solução das mesmas.

Os dois aspectos que tornam este paradigma atrativo é tanto o nível de abstração em que pode ser feito o desenvolvimento de software, como o fato do mesmo ser suscetível de uma avaliação formal.

A entidade central da programação funcional é a função. Deste modo, funções podem devolver como resultados outras funções, as quais podem ser passadas como argumentos. A semântica de um programa funcional puro garante a ausência de efeitos colaterais (*side-effects*) entre as sub-expressões de uma função; isto faculta uma análise paralela das mesmas, sem riscos de dependências de dados e/ou controle, JONES [238].

Um clássico programa que ilustra a possibilidade de avaliação paralela de sub-expressões é o algoritmo recursivo de Fibonacci:

Como as duas chamadas recursivas de fibonacci são independentes, podem ser executadas em paralelo.

Fonte Implícita de Paralelismo

A técnica utilizada para linguagens funcionais puras (de alta ordem e sem anotações) é denominada redução de grafo (*Graph Reduction*), PEYTON-JONE [298]. As funções são expressas como árvores, com sub-árvores comuns (para representar as sub-funções compartilhadas). A computação consiste em selecionar subestruturas do grafo, reduzi-las a formas mais simples e atualizar o grafo com as mesmas. Quando não existirem reduções a serem feitas, o grafo que resta é o resultado da computação.

Utilizando regras de exclusão para evitar sobreposições, múltiplos processadores podem pesquisar simultaneamente diferentes regiões do grafo correspondente ao programa. As reduções das sub-árvores poderiam, então, ser realizadas em paralelo.

Potencialidade de Exploração do Paralelismo

Uma vez que o grafo correspondente ao programa funcional sem nenhum tipo de anotação varia de forma muito dinâmica durante sua redução, é bastante difícil gerenciar a distribuição de carga entre processadores, o total de novas tarefas criadas e o volume de comunicações. Segundo HANUS [210], a exploração totalmente implícita do paralelismo na programação funcional resulta em um grande número de tarefas paralelas de pequena granulosidade.

Os modelos atualmente existentes para exploração totalmente implícita do paralelismo com programação funcional, tem obtido desempenho razoável em equipamentos com memória compartilhada, não conseguindo boa eficiência com equipamentos de memória distribuída (HUDAK [219]).

Programação em Lógica

Uma importante característica das linguagens de programação em lógica é que as mesmas são de atribuição única. Assim, de forma diferente das linguagens imperativas, elas não permitem atribuições destrutivas às variáveis, nem controle explícito do fluxo de execução do programa. Isto confere às linguagens de programação em lógica, uma semântica clara (declarativa) e uma decorrente construção de programas elegantes, compactos e menos sujeitos a erros oriundos de aspectos de implementação (STERLING [344]).

Este forte aspecto declarativo permite que a avaliação de um programa em lógica possa ser feita por diferentes estratégias de controle de fluxo de execução. Isto implica que a ausência de efeitos colaterais decorrentes da semântica declarativa faculta que muitas das operações em um programa em lógica possam ser executadas em qualquer ordem (não determinismo), sem que com isto seja afetada a consistência de execução do mesmo.

Fontes Implícitas de Paralelismo

Existem duas principais fontes de paralelismo implícito na programação em lógica (KERGOMMEAUX [244]):

Paralelismo OU: este tipo de paralelismo refere-se a uma estratégia de busca paralela na árvore de metas do programa lógico. Assim, quando o processo de busca encontra uma ramificação na árvore de metas, ele pode iniciar processos paralelos de busca em cada ramo descendente (vide figura 11.3). O nome paralelismo OU deriva do fato que, em programas lógicos não determinísticos, existem várias respostas (vários caminhos) que satisfazem o objetivo.

Paralelismo E: em termos da árvore de metas (vide figura 11.3), o paralelismo E corresponde à construção paralela de uma ramificação. Neste caso, quando o processo de busca reconhece que um número de passos deve ser efetuado para completar um ramo, ele pode iniciar processos paralelos para avaliar estes passos.

Outra fonte de paralelismo implícito na programação em lógica é o paralelismo de unificação. O paralelismo disponibilizado é de baixa granulosidade e, para ser explorado com eficiência, exige hardware especializado.

O paralelismo E, por sua vez, pode ser explorado entre literais independentes (sem possibilidade de conflito na atribuição de valores a variáveis), ou entre quaisquer literais, às custas de um mecanismo mais complexo de detecção e gestão do paralelismo.

Exemplo de Exploração do Paralelismo

A grande maioria dos modelos que exploram paralelismo na programação em lógica, implementam a linguagem Prolog (STERLING [344]), e utilizam a WAM (Warren Abstract Machine - WARREN [378], AIT-KACI [49]) como estratégia de compilação.

O OPERA (BRIAT [96], GEYER [195]) é um exemplo de modelo que explora de forma implícita o paralelismo OU. Neste modelo, o paralelismo é explorado de forma multisequencial, no qual cada processador ativo está, em determinado momento, trabalhando de forma independente, sobre um ramo específico da árvore de busca.

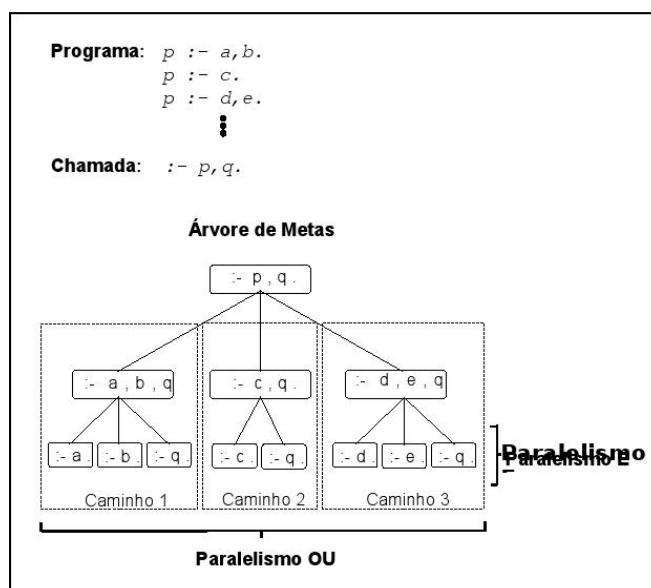


Figura 11.3: Fontes de Paralelismo na Programação em Lógica

O modelo &-Prolog (HERMENEGILDO [213]) é um dos mais maduros modelos explorando paralelismo E independente. Ele combina uma detecção de independência a nível de compilação com um eficiente ambiente de execução implementado em multiprocessadores com memória compartilhada. Sua proposta é fundamentada no Paralelismo E Restrito (DEGROOT [148]).

O paralelismo pode ser explorado automaticamente, a partir de um programa em Prolog padrão (opcionalmente o programador pode fazer anotações). O compilador do modelo executa a transformação de programas Prolog para &-Prolog. A análise estática do compilador detecta a independência entre literais, mesmo na presença de predicados com *side-effects* (MUTHUKUMAR [281] e [282]).

Os programas &-Prolog são compilados em uma extensão da WAM denominada PWAM, cuja principal diferença em relação a WAM é a adição de uma *pilha de literais paralelizáveis*, onde processadores inativos retiram trabalho.

Potencialidade de Exploração do Paralelismo

As alternativas de exploração do paralelismo na programação em lógica são fortemente ortogonais entre si; isto significa que podem ser explorados simultaneamente, sem que um comprometa o outro.

O paralelismo OU, presente em programas com não determinismo, caracteriza-se

por uma granulosidade mais elevada, o que facilita sua exploração em máquinas de memória distribuída. Por sua vez, o paralelismo E, passível de ser explorado praticamente em qualquer programa em lógica, apresenta uma granulosidade pequena, daí a maioria dos modelos existentes serem orientados a equipamentos de memória compartilhada (baixo custo de comunicação) (KERGOMMEAUX [244]).

Os modelos para exploração do paralelismo na programação em lógica de forma totalmente implícita, apresentam um elevado nível de abstração. Este aspecto, que lhes confere elegância, portabilidade e possibilidade de aproveitamento de toda cultura da programação seqüencial disponível, também dificulta a garantia de desempenho destes modelos frente à diversidade das aplicações reais. Porém, é importante ter presente que situações de bom ganho de desempenho foram registradas. Por sua vez a elevada dinâmica da programação em lógica dificulta sobremaneira qualquer medida de custo.

A exploração do paralelismo na programação em lógica se mostra promissora. A maioria dos modelos implementados explora somente um tipo de paralelismo, uns poucos exploram dois tipos, e nenhum explora efetivamente todas as alternativas de paralelismo possíveis.

O custo-benefício da exploração implícita do paralelismo ainda não atingiu patamares satisfatórios (KERGOMMEAUX [244]).

11.3.2 Modelos com Assinalamento do Paralelismo Explícito

Nestes modelos, o programador deve estar ciente da natureza do paralelismo que será explorado, e deve expressar todo o potencial para o mesmo no código, porém não precisa se envolver como este será efetivamente tratado pelo ambiente de execução. Portanto, fica a cargo do modelo como irá ocorrer a decomposição, o mapeamento, a comunicação e a sincronização das tarefas paralelas.

Assim, o programa deve expressar o máximo paralelismo inerente ao algoritmo. A implementação do mesmo (compilação e execução), por sua vez, reduzirá este nível de paralelismo ao possível de ser explorado em função da arquitetura destino e dos decorrentes custos de escalonamento, comunicação e sincronização.

Como exemplo de paralelismo neste nível de abstração temos exploração do paralelismo de dados.

Paralelismo de dados - exploração de laços

O paralelismo de dados tem uma origem histórica no uso de processadores pipeline. Neste caso, a exploração de paralelismo ocorria com a aplicação de forma independente e repetida de uma mesma operação sobre dados diferentes. Tal situação permitia o uso de processadores com registradores vetoriais, com os quais o paralelismo era explorado, associado a um custo reduzido de controle das repetições.

Com o desenvolvimento das arquiteturas matriciais (SIMD), as mesmas se tornaram ótimas candidatas para processar em paralelo o código vetorizado. Por sua vez, o código para arquiteturas matriciais pode ser eficientemente executado em multiprocessadores.

Deste modo, o paralelismo de dados, inicialmente destinado a pipelines vetoriais, pode atualmente ser explorado em diversas arquiteturas paralelas.

Assim, genericamente, o paralelismo de dados é uma estratégia na qual as rotinas paralelas são composições de operações que são aplicadas a dados de um determinado tipo, e que produzem resultados do mesmo tipo.

Fonte de paralelismo

No caso mais usual, o paralelismo de dados envolve estruturas de dados organizadas na forma de matrizes de dimensões variadas (estruturas regulares), e o paralelismo se viabiliza quando estes dados são passíveis de manipulação independente. Esta manipulação independente é comum na álgebra matricial.

Exemplo de Exploração do Paralelismo

Como exemplos típicos de modelos que exploram paralelismo de dados surgem os diversos dialetos FORTRAN. O HPF (*High Performance Fortran* - THAKUR [357], MEHROTRA [273]), particularmente, potencializa a exploração do parale-

lismo de dados inerente aos laços, disponibilizando construtores que devem ser utilizados nos programas para determinar como as estruturas de dados devem ser alocadas aos processadores.

Para fazer isto o programador precisa ter clareza da relação entre os dados.

Potencialidade de Exploração do Paralelismo

O código para exploração do paralelismo de dados é simples de ser escrito e depurado. Isto decorre do paralelismo ser explicitamente trabalhado pelos sincronismo e fluxo de controle inerentes aos equipamentos matriciais.

Os programas para paralelismo de dados necessitam, para sua execução, de uma pré-distribuição dos conjuntos de dados que serão manipulados. Deste modo, a organização das estruturas de dados utilizadas neste tipo de paralelismo é determinante para o seu desempenho.

Portanto, este paralelismo é centrado em computações locais e operações de manipulação de dados (replicações, permutações, reduções, etc.). Pode ser explorado com sucesso mesmo em problemas de baixa granulosidade, desde que estes utilizem conjuntos de dados com estruturas multidimensionais regulares.

Dependendo da granulosidade inerente ao problema e do algoritmo adotado, o paralelismo de dados pode também ser explorado em multiprocessadores tanto de memória compartilhada como distribuída.

Porém, seu emprego é mais facilmente potencializado nas arquiteturas síncronas, as quais conseguem explorar eficientemente o paralelismo a nível de instrução.

O paralelismo de dados, tem sido utilizado com ótimos desempenhos em diversas arquiteturas. A simplicidade do seu código facilita a codificação e eventuais recodificações quando de migrações entre arquiteturas diferentes. Faculta uma previsão de desempenho e também medida de custo. Porém, é possível sua exploração com desempenho somente quando os dados tiverem as características de independência e regularidade mencionadas.

11.3.3 Modelos com Assinalamento e Decomposição do Paralelismo Explícitos

Estes modelos delegam para o programador a decisão de como será decomposto o trabalho paralelo em partes. As implicações desta decisão, no entanto, serão tratadas de forma transparente pelo modelo.

Uma vez dividido o problema, a atribuição das partes aos processadores e a forma como estas vão se comunicar e sincronizar não precisam ser explicitadas no programa.

Existem poucas propostas neste nível de abstração (somente duas), e sua implementação ocorre na forma de bibliotecas utilizadas a partir das linguagens C e FORTRAN. Como exemplo deste nível de abstração, surge a exploração do paralelismo com restrição de comunicações e sincronização.

Exemplo de Exploração do Paralelismo

Um modelo que explora o paralelismo neste nível é o BSP (*Bulk Synchronous Parallelism model* - SKILLICORN [330], VALIANTE [369]). As computações em BSP são divididas em fases, alternando comunicações globais e computações locais.

Cada fase inicia com a ativação das operações de comunicação. O processamento nas tarefas paralelas ocorre utilizando somente referências a dados locais. Enquanto isto, de forma independente, a rede de interconexões realiza as trocas de dados necessárias entre os processadores. Ao final de cada fase, chamadas de *superstep*, todas as comunicações são entendidas como prontas (para garantir isto é utilizada uma barreira de sincronização) e todas as atribuições feitas à memória global (que dizem respeito à tarefa paralela do processador) são reproduzidas localmente. Do ponto de vista do programador, as solicitações de dados à memória global remota serão feitas pelo BSP no *superstep* anterior àquele no qual eles serão utilizados.

Uma característica que se destaca no BSP é o fato do mesmo expressar as propriedades da rede de interconexão utilizada na arquitetura paralela a partir de uns poucos parâmetros. A máquina abstrata do BSP consiste de uma coleção de

p processadores, cada qual com sua memória local, todos conectados através da rede de interconexão. Os parâmetros da rede de interconexão utilizados são: l tempo necessário para a realização da barreira de sincronização, e g - a razão na qual dados locais com endereço aleatório podem ser liberados/recebidos. Estes parâmetros são determinados experimentalmente para cada computador paralelo.

Deste modo, se o tempo consumido com computações locais exigir um total w e o volume de valores recebidos e enviados pelo processador for h , então o tempo total gasto em um *superstep* é:

$$t = w + hg + l$$

Considerando que a computação seqüencial tem diversas métricas de complexidade disponíveis, e trabalhando com o maior valor previsto para as variáveis anteriores, o tempo máximo que um *superstep* irá despendar pode ser facilmente obtido.

O outro modelo que trabalha neste nível de abstração é o logP (CULLER [129]). Também neste modelo são utilizadas *threads* com contextos locais e com atualizações de dados por comunicações globais.

Potencialidade de Exploração do Paralelismo

O nível de abstração dos programas em BSP é bastante elevado.

Apesar de sua decomposição em *threads* ser feita pelo programador, a alocação das mesmas e toda comunicação/sincronização é feita de forma transparente para o mesmo.

Podem ser obtidos bons desempenhos com o modelo do BSP, porém sua previsibilidade é influenciada pelos seguintes fatores: Comunicações: a otimização das comunicações depende de como ocorre a alocação (automática) das *threads* aos processadores, tendo em vista as características da rede de interconexão (topologia, largura de banda, etc.);

Sincronização: o custo total introduzido pelas barreiras de sincronização entre cada fase de execução, depende da natureza do problema e do algoritmo utilizado (sua granulosidade, possibilidade de ser particionado em partes homogêneas, etc.).

Por outro lado, um ponto forte do modelo BSP é a existência de medida de custo, através da qual é possível obter o custo real de execução de um programa para qualquer arquitetura, desde que sejam conhecidos os parâmetros l e g para a mesma.

A versão corrente do BSP trabalha com uma biblioteca SPMD (*Simple Program Multiple Data*), que fornece operações para colocar dados na memória remota de outro processo, para receber dados de um processo remoto e para sincronização (pode ser utilizada a partir de C ou FORTRAN).

11.3.4 Modelos com Assinalamento, Decomposição e Mapeamento do Paralelismo Explícitos

Nestes modelos, o programador, além de dividir o programa em partes, tem a incumbência de avaliar qual o melhor processador para cada parte. Uma vez que a proximidade dos processadores é determinante para o desempenho das comunicações, é indispensável para o programador ter conhecimento das características da rede de interconexões utilizada na arquitetura. Este comprometimento do código com as características do equipamento, trazem repercussões nos custos de migração de software entre diferentes arquiteturas.

Por sua vez, o ambiente de execução se responsabiliza pelo gerenciamento das comunicações e das sincronizações entre as tarefas paralelas.

Exemplo de Exploração do Paralelismo

Um modelo que trabalha neste nível de abstração é o “Linda” (CARRIERO [106]). Neste conhecido modelo, as comunicações ponto-a-ponto são substituídas por um espaço compartilhado, no qual os dados são colocados pelos processadores

e a partir do qual são recuperados associativamente. Este espaço é denominado *espaço de tuplas*.

As três operações básicas do modelo de comunicações utilizado pelo Linda são: uma que lê o *espaço de tuplas* buscando aquelas que satisfazem os campos informados e a correspondente aridade, uma segunda que faz a leitura porém remove a tupla que satisfaz a condição do espaço de tuplas e uma terceira que coloca uma tupla no *espaço de tuplas*.

As operações de comunicação em Linda desconectam o emissor do receptor, isto é, a *thread* que envia informação, em nenhum momento precisa saber qual vai recebê-la. Não existe conexão nem espacial e nem temporal entre os mesmos.

Potencialidade de Exploração do Paralelismo

Como as comunicações entre programas exigem que ambos os lados (emissor e receptor) tenham seus parâmetros devidamente concordantes, e considerando que os programas paralelos usualmente contemplam muitas comunicações, esta tarefa de especificar comunicações introduz um custo elevado no desenvolvimento do software paralelo.

Os modelos que operam neste nível de abstração reduzem muito este custo, oferecendo primitivas de alto nível para especificação das trocas de dados. Normalmente esta simplificação é feita separando, nos programas, os aspectos de processamento dos de comunicação. Normalmente é disponibilizada uma linguagem a parte (subconjunto da linguagem) para tratar das comunicações. Esta divisão faz com que a computação e a comunicação fiquem ortogonais entre si, de tal forma que uma particular estratégia para as comunicações possa ser utilizada com diversas linguagens seqüenciais. Este é o caso particular de Linda.

É preciso ter presente que a combinação de decomposição explícita do paralelismo, com um mecanismo de troca de dados que garante um desacoplamento entre as partes comunicantes, é uma fonte de possíveis inconsistências no estado de execução do programa paralelo (*deadlocks*). Por exemplo, uma *thread* pode ficar bloqueada aguardando uma tupla que nunca será inserida no espaço de tuplas. Apesar de ser possível um desacoplamento entre emissor e receptor, o algoritmo

introduz necessidades de sincronização (que neste caso é feita através dos dados) que são inerentes à natureza do algoritmo que está sendo computado.

Como a eficiência da implementação das abstrações de alto nível utilizadas na construção e gerência do espaço de tuplas (comunicações) é dependente da rede de interconexão utilizada na arquitetura, não é possível trabalhar com medidas de custo.

11.3.5 Modelos com Assinalamento, Decomposição, Mapeamento e Comunicação Explícitos

Nestes modelos, o programador tem a seu encargo especificar quase todos os detalhes da implementação paralela, exceto os aspectos pertinentes a sincronização.

Para oferecer isto, via de regra, estes modelos empregam uma semântica assíncrona, na qual as mensagens são enviadas, porém o emissor não fica atrelado ao tempo necessário para que elas atinjam seu destino.

Exemplo de Exploração do Paralelismo

O mais importante modelo nesta classe é Atores (*Actors* - AGHA [47]). Ele consiste de uma coleção de objetos chamados atores, todos contendo uma pilha de mensagens de entrada. Todo ator executa repetidamente a sequência: leitura da mensagem disponível na pilha de entrada; envia suas mensagens a todos os outros atores que ele conhece e a partir das mensagens que chegaram define seu novo contexto, o qual determinará suas respostas às próximas mensagens que receber.

As mensagens são enviadas de forma assíncrona e sem preocupação de ordem.

Os nomes dos atores podem ser distribuídos através de mensagens. A exploração de concorrência e distribuição em objetos é amplamente discutida em BRIOT [97].

Potencialidade de Exploração do Paralelismo

Além dos custos de comunicação, outro ponto de estrangulamento do desempenho do modelo de atores é o processamento seqüencial das mensagens na fila de entrada. Para minimizar este aspecto, o modelo ActorSpace (AGHA [45]) propõe estratégias para reduzir o número de mensagens distribuídas.

A comunicação entre processadores no modelo atores e seus derivados é grande, o que aumenta consideravelmente o indeterminismo introduzido pelo desempenho da rede de interconexões da arquitetura. Em função disto, não é possível nestes modelos, nem garantia de desempenho, nem medida de custo.

11.3.6 Modelos nos quais o Paralelismo é Explorado de Forma Totalmente Explícita

Nestes modelos o programador precisa especificar todos os aspectos da implementação. Em função disto, se torna uma tarefa trabalhosa desenvolver software empregando tais modelos, porque tanto a sua correção quanto seu desempenho, somente podem ser atingidos pela criteriosa atenção de um grande número de detalhes.

Os primeiros modelos para o paralelismo, na sua grande maioria, atuavam neste nível, normalmente voltados para um particular tipo de arquitetura, e com sua execução paralela gerenciada de forma totalmente explícita.

Exemplo de Exploração do Paralelismo

Um conhecido exemplo de exploração de paralelismo neste nível é a linguagem *SR* (*Synchronizing Resources* - ANDREWS em [63] e [64] e ANDREWS e OLSSON em [66], [67] e [68]). Nesta linguagem, estão presentes as características usuais do paradigma convencional (imperativo), tais como: tipos, variáveis, atribuição destrutiva, comandos de controle de repetição, comandos de seleção simples e múltipla, procedimentos, etc. Para exploração do paralelismo *SR* fornece mecanismos específicos para gerenciamento da concorrência, comunicação e sincronização.

Em SR um programa pode ser formado por diversos espaços de endereçamento (máquinas virtuais), os quais podem estar localizados em múltiplos computadores (máquinas físicas). Os processos residentes em um mesmo espaço de endereçamento podem compartilhar memória. Desta forma, a linguagem SR suporta programação em ambientes distribuídos e ambientes com memória compartilhada. SR é baseada no conceito de recurso (*resource*). O recurso é um módulo que pode conter diversos processos.

Um recurso é dinamicamente criado pelo comando *create* (portanto explicitamente) e os seus processos comunicam-se utilizando semáforos para sincronização. A comunicação entre processos de recursos remotos pode ser feita através de troca de mensagens assíncronas, chamada remota de procedimentos (RPC) e *rendezvous*.

Potencialidade de Exploração do Paralelismo

De forma análoga a outros modelos de sua categoria, a linguagem SR não oferece recursos de abstração para detecção, decomposição, comunicação ou sincronização do paralelismo. A SR, também como outros modelos análogos, costuma oferecer mecanismos alternativos para o programador gerenciar o paralelismo, porém, mesmo assim, é inviável conciliar em um programa independência de arquitetura e máximo desempenho.

Dependendo da arquitetura, problemas de qualquer granulosidade podem ser computados com desempenho em SR. Para uma arquitetura em particular, SR oferece garantia de desempenho e medida de custo, no entanto, pelos detalhes que exige, torna o desenvolvimento de software paralelo uma tarefa onerosa.

Capítulo 12

Escalonadores de Tarefas

Sistemas de “job scheduler”, ou de agendamento/escalonamento de tarefas, para clusters e grids, são softwares desenvolvidos para controlar a execução dos “jobs” ou tarefas no ambiente. Esse tipo de software é composto normalmente por um gerenciador de filas de processamento (“batch queue manager”) que prioriza e controla a execução de múltiplas tarefas. “Job schedulers” são também, normalmente, conhecidos como gerenciadores de distribuição de recursos ou gerenciadores de filas de execução.

As características básicas esperadas de um “job scheduler” são:

- Interface para se definir o fluxo e/ou a árvore de dependência de execução dos trabalhos;
- Submissão automática das tarefas para execução;
- Interfaces para monitoramentos das execuções;
- Controle das prioridades e das filas de tarefas, para controlar a ordem de execução das tarefas.

Vários sistemas como ERPs, Bancos de dados, sistemas de cópias de segurança, incluem ou podem usar algumas características de sistemas de agendamento de tarefas.

Alguns exemplos de ferramentas utilizadas em ambientes de cluster para “job scheduler” serão descritos a frente:

- openPBS 12.1;
- TORQUE 12.2;
- MAUI 12.3.

12.1 OpenPBS

O propósito do sistema OpenPBS é prover controles adicionais sobre a inicialização ou o seqüenciamento de execução de grupos de tarefas, e permitir a distribuição destes trabalhos entre vários nós de um cluster. O sistema de controle de tarefas (batch server) permite que sejam definidas e implementadas regras para diferentes tipos de recursos e para a quantidade de recursos pode ser usada por diferentes tarefas. Este sistema também provê um mecanismo com o qual um usuário pode assegurar que uma tarefa tenha os recursos necessários para ser completada.

Este sistema de controle de tarefas é constituído por um conjunto de componentes, um servidor, por clientes e por comandos de usuários. O servidor gerencia um número de diferentes objetos, como tarefas e filas de execução.

Interações típicas entre os componentes são baseadas no modelo cliente-servidor, quando um cliente faz a requisição para o servidor de uma tarefa, o servidor executa o trabalho em um de seus clientes. Clientes não podem criar ou modificar os objetos diretamente, eles dependem de uma ordem do servidor que gerência estes objetos.

O servidor de tarefas é um processo permanente ou um conjunto de processos, um “daemon”. O servidor de tarefas gerencia os objetos (trabalhos a serem executados), assim como, as filas e as tarefas. Ele provê serviços como: criação, execução, modificação, exclusão e roteamento de tarefas para os clientes (nós computacionais) responsáveis pela execução dessas tarefas.

12.2 TORQUE

TORQUE é um gerenciador de recursos de código aberto que provê controle sobre tarefas (batch jobs) em nós computacionais distribuídos. Ele é um esforço da comunidade de software livre, baseado no código original do projeto *PBS, e que já conta com mais de 1.200 correções e melhorias de código, com melhorias significativas em áreas como escalabilidade, tolerância à falhas e novas extensões. Alguns contribuidores do projeto são: NCSA, OSC, USC , U.S. Dept of Energy, Sandia, PNNL, U-Buffalo, TeraGrid e outras organizações líderes em desenvolvimento de HPCs.

Características:

- Tolerância à falhas:
 - Suporte a checagem de nós fora do ar;
 - Várias condições de checagens de falhas nos nós.
- Interface de seqüenciamento:
 - Interface de busca estendida, provendo informações mais acuradas sobre o escalonamento das tarefas;
 - Interface de controle estendida para permitir maior controle sobre as tarefas, seus atributos e execução;
 - Permite a obtenção de dados estatísticos de tarefas já executadas.
- Escalabilidade:
 - Servidor de monitoramento;
 - Capacidade de trabalhar com cluster muito grande (acima de 15TF e 2500 processadores);
 - Capacidade de trabalhar com um grande volume de tarefas (acima de 2000 processos);
 - Capacidade de suportar grande número e tamanho de mensagens de servidores.
- Usabilidade:
 - Mecanismo de log mais completo;
 - Logs com características de leitura mais simples.

12.3 MAUI

Maui é um avançado escalonador de tarefas com um grande conjunto de características desenvolvidas especialmente para plataformas de computação de alto desempenho (HPC). Ele usa políticas de escalonamento agressivas para otimizar a utilização de recursos e minimizar o tempo de respostas (execução) das tarefas (job). E, simultaneamente, provê ferramentas de controle administrativas sobre os recursos e o volume de tarefas sendo executadas, permitindo uma grande capacidade de configuração em diversas áreas como: priorização de tarefas, seqüenciamento, alocação, distribuição de cargas e políticas de reserva de recursos.

Maui foi projetado com base em experiências coletivas dos maiores e mais avançados centros de HPC do mundo. Ele prove ferramentas que estes centros precisavam para controlar, rastrear e otimizar o uso de seus recursos. Uma coleção extensiva de estatísticas e ferramentas de perfil, modo de teste de operação e um avançado simulador de características que permite a checagem de ambientes de produção, para saber se todas as alterações de configurações foram completamente feitas de forma segura.

Capítulo 13

Grids Computacionais

Resumo

Grids Computacionais surgiram em meados da década de 90 com a promessa de viabilizar a execução de aplicações paralelas em recursos geograficamente dispersos e pertencentes a múltiplas organizações. Tal proposta tinha dois grandes apelos. O primeiro era o de fornecer uma plataforma muito mais barata para execução de aplicações distribuídas (que os supercomputadores paralelos). O segundo era possibilitar (através da aglomeração de recursos dispersos) a execução de aplicações paralelas em uma escala simplesmente impossível em um único supercomputador. Com a evolução da tecnologia de grids percebeu-se que a composição automática de um conjunto de recursos para servir uma aplicação criava a oportunidade de oferecer serviços sob demanda. Assim, surgiu a idéia de um Grid onde seja possível prover sob demanda qualquer serviço computacional (não somente serviços para computação de alto desempenho). Como consequência, as tecnologias de Grids Computacionais se fundiram com Web Services e se posicionaram como uma tecnologia fundamental para computação no século XXI. O texto a seguir descreve a evolução dos Grids Computacionais, cobrindo as principais tecnologias existentes, e apresentando os aspectos importantes para criação de Grids de Serviços genéricos, bem como características específicas de Grids para alto desempenho.

13.1 Introdução

Grids Computacionais são atualmente uma das áreas mais “quentes” da Computação. O que começou em universidades e institutos de pesquisa ganhou o mundo empresarial e hoje faz parte da estratégia de corporações como IBM, HP, Sun, NEC, Microsoft e Oracle. Em suma, temas relacionados a Grids Computacionais figuram hoje como um assunto em moda.

Porém, o que afinal vem a ser um Grid Computacional? A visão original estabelece uma metáfora com A Rede Elétrica (*The Electric Grid*) [183]. A Rede Elétrica disponibiliza energia elétrica sob demanda e esconde do usuário detalhes como a origem da energia e a complexidade da malha de transmissão e distribuição. Desta forma, se temos um equipamento elétrico, simplesmente o conectamos na tomada para que ele receba energia. O Grid Computacional (*The Computational Grid*), portanto, seria uma rede na qual o indivíduo se conecta para obter *Serviços Computacionais* que agregam recursos sob demanda (ex.: ciclos, armazenamento, software, periféricos, etc). A Figura 13.1 ilustra esta idéia.

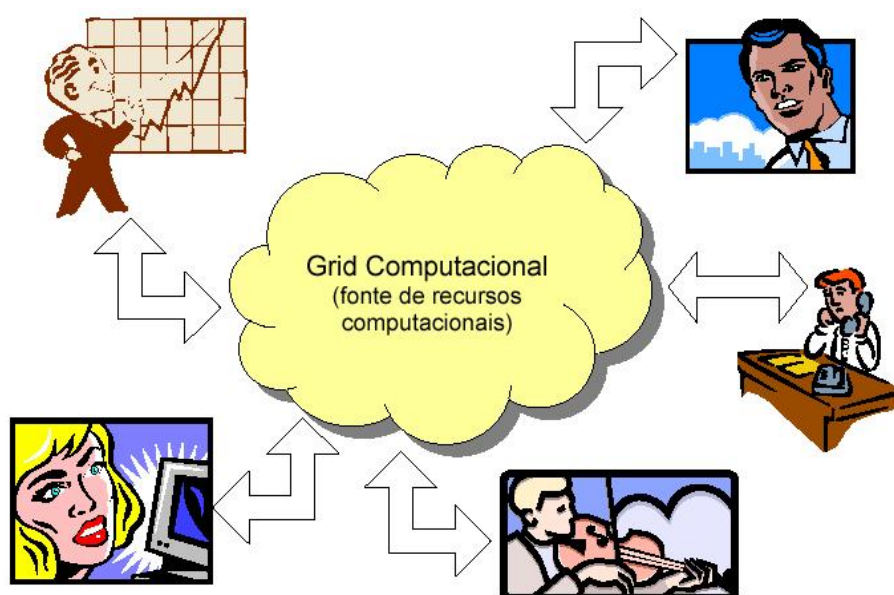


Figura 13.1: Acesso transparente a serviços e recursos

Um sistema que forneça serviços computacionais sob demanda de forma transparente é certamente desejável para várias instituições e aplicações. Note que, para

muita gente, a Internet é este sistema. De fato, para aqueles cujas necessidades de processamento são satisfeitas por um computador pessoal, a Internet atende os requisitos básicos de um Grid Computacional. Por exemplo, quando usamos *home banking*, nosso computador pessoal, uma série de roteadores e os computadores do nosso banco se agregam sob demanda para nos fornecer um serviço de forma transparente.

É importante esclarecer que não queremos, com o exemplo anterior, sugerir que um Grid Computacional é o mesmo que a Internet. De uma certa forma, o Grid é a evolução da Internet. A idéia é que *qualquer* serviço computacional possa ser obtido no Grid, e que se possa agregar automaticamente vários serviços mais simples, gerando um serviço mais sofisticado. Voltando ao nosso exemplo de *home banking*, este serviço é pensado para viabilizar o acesso de um humano (um cliente do banco) à seus dados bancários. É muito complicado usar o serviço em outros contextos, como parte de uma aplicação maior, que por exemplo acesse os dados bancários na preparação do imposto de renda.

Grids Computacionais nasceram da comunidade de Processamento de Alto Desempenho, motivada pela idéia de se utilizar computadores independentes e amplamente dispersos como plataforma de execução de aplicações paralelas [183]. Porém, com a evolução da pesquisa sobre Grids Computacionais e tecnologias utilizadas pela indústria para computação distribuída, houve naturalmente uma convergência entre o mundo acadêmico e empresarial. Assim, a idéia é prover uma infraestrutura que viabilize serviços sob demanda, permitindo uma maior colaboração entre várias instituições, através do compartilhamento de seus serviços e recursos, e utilizando mecanismos que facilitem a interoperabilidade.

Os atrativos desta idéia incluem a possibilidade de fornecimento de *qualquer* serviço computacional sob demanda, o qual pode ser composto dinamicamente por outros serviços e agregar recursos localizados em várias instituições distintas e geograficamente dispersas. Além disso, os recursos podem ser alocados em uma quantidade enorme (e.g. centenas de milhares de computadores conectados via Internet) e por um custo muito menor do que alternativas tradicionais (baseadas em supercomputadores paralelos).

Um aspecto importante, que deve ser considerado, é o fato de mesmo havendo a convergência entre tecnologias para alto desempenho e da indústria, existe uma leve diferença entre Grids que fornecem e que não fornecem um *serviço de execução*

remota. Esse serviço é fundamental para a comunidade de alto desempenho, uma vez que permite a execução de aplicações paralelas no Grid. Mas, é concebível imaginar Grids mais comerciais, onde o foco é o acesso e composição de serviços sob demanda, que funcionem perfeitamente bem sem disponibilizar o serviço de execução remota.

O serviço de execução remota é crucial porque ele introduz importantes desafios para infraestrutura do Grid. Os Grids que fornecem execução remota tendem a ser mais complexos, pois ao permitir uma maior flexibilidade aos clientes do serviço, que podem *converter* o serviço de execução remota em *qualquer* serviço, deve-se preocupar com questões como segurança (ex.: como proteger o recurso de aplicações maliciosas?) e escalonamento (ex: que serviço de execução é mais adequado para a aplicação?).

Neste texto, usaremos *Grids de Serviços* quando estivermos tratando questões pertinentes a qualquer Grid, disponibilize ele o serviço de execução remota, ou não. Usaremos *Grids para Alto Desempenho* quando estivermos tratando das questões adicionais que são introduzidas pelo serviço de execução remota. Assim, nesta nossa terminologia, todo Grid para Alto Desempenho é um Grid de Serviços, embora o contrário não seja necessariamente verdade.

Outra forma de ver Grids é encará-los como plataformas de computação distribuída. Há várias plataformas tradicionais de computação distribuída, seja de propósito mais comercial (CORBA, DCOM, etc), seja de propósito mais técnico (clusters, supercomputadores paralelos). Para esclarecer um pouco mais a diferença entre os Grids e outras plataformas de computação distribuída, podemos citar algumas características que são intrínsecas aos Grids. De modo geral, os Grids são mais distribuídos, diversos e complexos que outras plataformas. Aspectos que evidenciam esta distribuição, diversidade e complexidade são:

- *Heterogeneidade*: Os componentes que formam a infraestrutura tendem ser extremamente heterogêneos. Ou seja, é importante ter em mente que qualquer solução para Grids Computacionais deverá lidar com recursos de várias gerações, softwares de várias versões, instrumentos e serviços dos mais variados tipos.
- *Alta dispersão geográfica*: Essa característica se refere a escala que um Grid pode atingir. Nesse sentido, Grids podem ter escala global, agregando ser-

viços localizados em várias partes do planeta.

- *Compartilhamento*: Em contraste com soluções *space-shared*, um Grid não pode ser dedicado a uma aplicação de forma exclusiva por um determinado período de tempo. Isso tem impacto no desenvolvimento de aplicações que executam sobre a infraestrutura de um Grid Computacional.
- *Múltiplos domínios administrativos*: Grids congregam recursos de várias instituições. Sendo assim, além da heterogeneidade mencionada anteriormente, é possível também a existência de várias políticas de acesso e uso dos serviços, de acordo com as diretrizes de cada domínio que faz parte do Grid.
- *Controle distribuído*: Tipicamente não há uma única entidade que tenha poder sobre todo o Grid. Isso é um reflexo da dispersão dos componentes do Grid, pois cada instituição pode implementar sua política em seus recursos locais, mas não interfere diretamente na implementação de políticas no acesso aos serviços de outras instituições participantes.

Note que esta discussão propõe um conceito e não uma definição para Grid Computacional. Uma plataforma de fornecimento de serviços computacionais que apresenta as características acima listadas certamente é um Grid. Contudo, a ausência de alguma das características não deve automaticamente desqualificar uma determinada plataforma como Grid. Por outro lado, o leitor deve estar atento que o termo Grid Computacional pode ser usado tão e somente como ferramenta de marketing [180]. Devido a sua popularidade e a seu impacto positivo, o termo *Grid Computing* tem sido utilizado de forma muito liberal, como no passado outros termos já foram (como: Orientação a Objetos, Sistemas Abertos, Downsizing, Reengenharia, Internet/Intranet/Extranet, entre outros).

Portanto, com o objetivo de desmistificar e posicionar os esforços atuais na concretização da visão original dos Grids Computacionais, discutiremos vários aspectos importantes dos Grids de Serviços, e também das questões particulares dos Grids para Alto Desempenho, que incluem serviços de execução remota.

Este texto está organizado da seguinte forma: na sessão 13.2 apresentamos os Grids de Serviços, suas principais características, benefícios, desafios que tais características sugerem e os investimentos de padronização. Na sessão 13.3 serão apresentados as questões exclusivas a Grids para Alto Desempenho, que envolvem o desenvolvimento de um ambiente de execução de aplicações paralelas em

escala global. Na sessão 13.4 faremos um estudo de caso com algumas soluções para Grids Computacionais. Finalmente, na sessão 13.5 concluiremos uma breve discussão sobre as tendências em Grids Computacionais.

13.2 Grids de Serviços

Antes de se iniciar uma discussão sobre aspectos relacionados a Grids de Serviços é necessário definir o que é um serviço. Na teoria econômica, *um serviço é uma mercadoria imaterial provida por uma entidade legal (provedor) para satisfazer as necessidades de outra entidade (cliente)* [190].

Utilizando essa definição como analogia, consideramos como *serviço computacional* qualquer recurso (ou outro serviço) que possa ser acessado remotamente e descrito através de uma interface (por um provedor), a qual pode ser interpretada de forma automática (por um cliente). Desta forma, tudo pode ser considerado como serviço, desde que exista a possibilidade de se criar uma abstração que forneça essa interface.

Neste capítulo discutiremos as tecnologias que permitem o desenvolvimento de infraestruturas baseadas em serviços computacionais, bem como aspectos importantes para a implementação dessas infraestruturas. Em seguida, abordaremos também padrões emergentes para Grids de Serviços.

13.2.1 Acesso a Serviços

De modo geral, a idéia por trás de uma arquitetura baseada em serviços computacionais não é uma novidade. Ou seja, o grande interesse atual da comunidade científica e da indústria por arquiteturas orientadas a serviços, bem como o crescimento de esforços nessa área se deve, em grande parte, pelo sucesso e amplo uso de Web Services [168, 348].

Portanto, é possível citar várias tecnologias anteriores a Web Services, que em sua essência forneciam mecanismos para a criação de arquiteturas baseadas em serviços. Por exemplo, em um sentido amplo, podemos considerar CORBA [35]

e RMI (*Remote Method Invocation*) [21] como tecnologias que permitem a construção de infraestruturas de computação distribuída baseadas em serviços. Todavia, aspectos como a ampla dispersão de recursos, falta de controle central e vasta diversidade de clientes, as quais são características intrínsecas dos Grids, introduzem um requisito que se torna fundamental: *interoperabilidade*.

Em RMI o provedor do serviço (um objeto remoto) requer, invariavelmente, que seu cliente, não só seja Java, como também conheça antecipadamente (em tempo de compilação) qual é sua interface. Para tornar um serviço acessível, o provedor deve registrá-lo no RMI *registry* [21]. O serviço é identificado e incluído em um catálogo mantido pelo registro. Do outro lado, o cliente usa uma URL para acessar o registro e obter uma referência para um dos serviços publicados em seu catálogo. O acesso ao serviço é feito usando a referência obtida através do RMI *registry*. A Figura 13.2 ilustra o acesso a um serviço usando RMI.

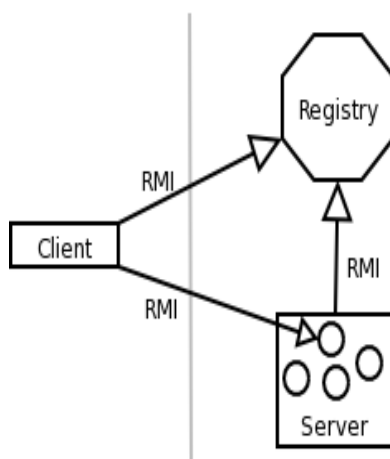


Figura 13.2: Acessando um serviço usando RMI

Ao contrário de RMI, CORBA oferece maior interoperabilidade entre clientes e provedores. Isso é possível pois serviços CORBA são descritos através de uma linguagem de descrição (*Interface Definition Language - IDL*), que é independente da linguagem em que o serviço e cliente são implementados. Esse aspecto torna CORBA mais flexível que RMI, pois permite que o cliente e o serviço sejam implementados em linguagens diferentes. Por exemplo, podemos ter um cliente desenvolvido em C++ e um serviço escrito em Java.

Em CORBA, um serviço é acessado de forma semelhante a RMI. A diferença subs-

tancial está na existência de uma entidade chamada *Object Request Broker* (ORB). A Figura 13.3 ilustra a operação de acesso a um serviço na plataforma CORBA. Note que o ORB é responsável por prover transparência no acesso do cliente ao serviço. Assim, tanto a localização e implementação do serviço, quanto do cliente tornam-se transparentes para ambos. Essa transparência na localização torna-se viável pelo uso do protocolo IIOP (*Internet Inter Orb Protocol*), que provê o transporte das invocações do cliente ao serviço.

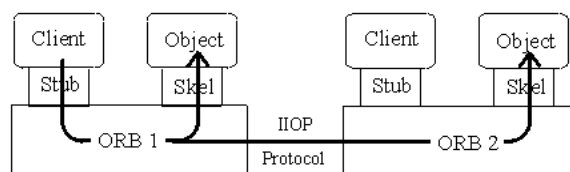


Figura 13.3: Acessando um serviço usando CORBA [14]

Apesar das vantagens de CORBA, alguns autores defendem que CORBA falhou na garantia de interoperabilidade entre os ORBs, o que tornaria a plataforma mais amplamente utilizada. O argumento é que a competição entre os desenvolvedores de ORBs se tornou acirrada acarretando problemas de interoperabilidade [199].

Por outro lado, Web Services surgiu como uma nova tecnologia para computação distribuída, que aproveitou vários padrões já bem estabelecidos como seu alicerce. O primeiro, e talvez maior, contraste entre Web Services e CORBA é a camada de transporte utilizada por cada uma das plataformas. Enquanto CORBA é baseado no protocolo IIOP, Web Services aproveita o protocolo HTTP para envio de mensagens entre cliente e provedor. A Figura 13.4 ilustra como ocorre a comunicação entre o cliente e um Web Service. Note que a interface do serviço é “descoberta” em tempo de execução. Após obtenção da referência para o serviço, o cliente pode iniciar a comunicação com o serviço através do envio de mensagens.

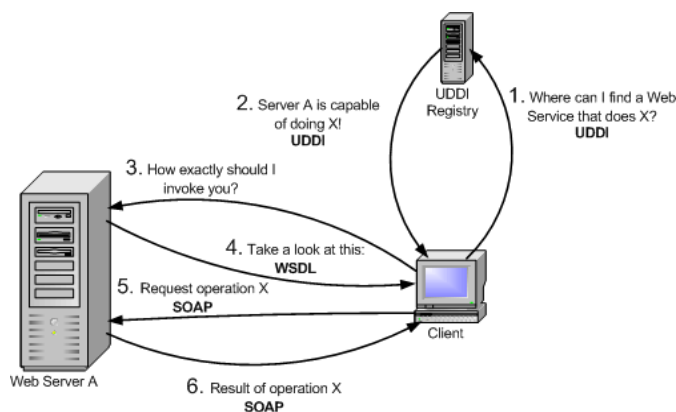


Figura 13.4: Interação entre cliente e provedor (Web Services) [343]

13.2.2 Descoberta de Serviços

Apesar de ser possível que clientes acessem serviços diretamente (sem envolver um catálogo), permitir que os serviços sejam descobertos dinamicamente é fundamental para construir uma infraestrutura de serviços sob demanda.

Sendo assim, em Grids computacionais é fundamental que seja possível descobrir os serviços existentes em uma infra-estrutura que podem atender a demanda de uma determinada aplicação. A idéia é que um serviço de descoberta funcione como as páginas amarelas de um catálogo telefônico, que permitem os usuários da rede telefônica encontrarem prestadores de serviços, a partir de alguns critérios, como classificação da atividade, localização do estabelecimento e outras informações divulgadas no catálogo.

Em sistemas CORBA por exemplo, o serviço de descoberta se chama Trader [35]. Ele possui o cadastro dos objetos distribuídos do sistema com suas respectivas propriedades, e permite que qualquer objeto consulte este cadastro para encontrar objetos cujas propriedades atendam um determinado critério de pesquisa.

Se em um sistema CORBA, restrito a uma única organização, um serviço de descoberta é importante, o que dizer de sistemas em Grid, que executam em um ambiente multi-institucional. Eles são fundamentais para que seja possível compartilhar recursos e serviços computacionais em escala global. No contexto da Computação em Grid, os recursos compartilhados podem ser ciclos de CPU, espaço em disco, software, sensores, dentre outros, que podem tornar-se acessíveis

na rede como Web Services [39].

Neste sentido, existem várias propostas de se construir um serviço de descoberta de Web Services. O serviço de descoberta mais discutido e incorporado à arquitetura WSRF [40] é do serviço UDDI (Universal Description, Discovery, and Integration) [27], já adotado por vários produtos voltados para a tecnologia de Web Services, de grandes empresas como Sun Microsystems, Microsoft e Oracle. O UDDI é ele mesmo um Web Service que permite a formação de um catálogo global de todos os Web Services compartilhados na Internet. Este catálogo é organizado em nós e registros. Um nó é um servidor UDDI onde estão os dados dos serviços cadastrados. Um conjunto de nós é chamado de registro, e cada nó só pode pertencer a um único registro.

Um registro pode ser privado, público ou afiliado. Os registros privados, ou corporativos, são aqueles que guardam informações de Web Services internos de uma empresa, protegido por um firewall, que devem ter seu acesso restrito às aplicações da empresa. Já os registros afiliados compartilham e trocam seus dados de forma controlada com outros registros, baseado em acordos entre as instituições envolvidas. Seus Web Services estão disponíveis apenas a usuários autorizados. Um exemplo de registros afiliados é o de Web Services sendo utilizados por aplicações de empresas de uma holding. Cada empresa poderia ter seu próprio registro e eles juntos formarem um grupo de registros afiliados, permitindo que as aplicações de cada empresa localizasse os serviços umas das outras. Há ainda os registros públicos que, como o próprio nome sugere, guardam informações sobre Web Services que podem ser acessados livremente na Web. Os dados mantidos pelos registros públicos podem ser compartilhados ou transferidos livremente entre eles. Um exemplo de um registro público é o UBR (UDDI Business Registry), hoje estruturado em quatro nós, sob responsabilidade das empresas IBM, Microsoft, NTT Communications e SAP. Qualquer entidade pode publicar ou consultar serviços nesses nós, gratuitamente. O UBR está para os Web Services assim como o Google está para as páginas Web. O consumidor de serviços que quiser localizar serviços na rede, provavelmente terá mais sucesso em sua busca se consultar o UBR. Igualmente, o provedor que quiser ter seus serviços encontrados terá que publicá-los no UBR.

No UDDI, cada Web Service tem cadastrado um documento WSDL (Web Service Description Language, baseado em XML) que descreve o serviço oferecido, for-

necendo informações que ajudam a localizá-lo no catálogo, assim como estabelecer a forma correta de invocá-lo. O cadastro e a consulta deve ser feito pelas aplicações através de APIs que se comunicam com os nós centrais utilizando o protocolo SOAP.

Alguns trabalhos criticam a natureza centralizada do UDDI, dizendo que, apesar de ser adotado como padrão na arquitetura WSRF, ele não oferece uma solução eficiente, escalável e tolerante a falhas como serviço de descoberta de Web Services, da forma como vem sendo implementado. Eles argumentam que, por ser centralizado, o UDDI apresenta baixo desempenho na atualização e consulta dos registros, que essa degradação tende a ser maior quanto maior for a quantidade de serviços catalogados e que é um ponto único de falha.

Uma alternativa ao UDDI é o WSPDS (Web Service Peer-to-peer Discovery Service) [75]. O WSPDS baseia-se no fato de que os Web Services podem formar uma rede peer-to-peer, onde os peers se conhecem e podem trabalhar de forma cooperativa, para atender as consultas. A idéia é que quando um peer precise realizar uma consulta na rede, ele a repasse primeiramente a seus peers conhecidos. Estes por sua vez, propagam a consulta aos peers de sua própria lista, até um limite definido por contadores de propagações contido nas mensagens trocadas. Cada peer que recebe a mensagem, realiza a consulta em sua lista local e retorna os resultados positivos para o peer original da consulta, que é entregue ao executor da consulta. Esse protocolo é empregado por aplicações populares como o Gnutella [32], na consulta de arquivos compartilhados por seus usuários.

O WSPDS traz algumas vantagens e desvantagens em relação ao UDDI. Ele é de fato uma solução mais tolerante a falhas, já que não possui pontos únicos de falha. Não há servidores, responsáveis por atender às consultas por serviços. A escalabilidade é também um ponto forte seu, já que o aumento da quantidade de serviços não influencia o desempenho das consultas. No entanto, não há uma garantia de que um serviço que atenda aos critérios de uma consulta seja localizado. Um resultado de consulta negativo não necessariamente significa a ausência de serviços na rede que satisfaçam os critérios de pesquisa. Pode acontecer que os peers que participam da pesquisa não tenham contato com o serviço que atende à consulta.

Uma alternativa híbrida entre as duas abordagens é a de federações de registros UDDI [26]. A idéia é fazer com que os registros estejam conectados entre si, em

uma rede peer-to-peer. Desta forma, quando uma consulta for feita a um registro e este não puder atendê-la, ele repassará a mesma consulta a outros registros, e assim sucessivamente, de forma semelhante a como ocorre no WSPDS. Cada registro realizará a consulta em seus próprios nós e devolverá ao registro original da consulta os resultados, que serão entregues ao executor da consulta. Esta abordagem foi originalmente adotada pelo serviço Trader do CORBA. Ela aumenta a robustez, tolerância a falhas, eficiência e escalabilidade do serviço de descoberta. A versão 3.0 do UDDI já fornece suporte para múltiplos registros e permite a formação das federações. Com o crescimento esperado do uso de Web Services e conseqüentemente do serviço UDDI, este parece ser o caminho natural de evolução do serviço de descoberta.

13.2.3 Autenticação e Autorização

Na Seção 13.2.1 descrevemos como ocorre o acesso a serviços usando várias tecnologias para computação distribuída. É importante ressaltar que apresentamos uma simplificação da realidade. Pois, devido à ampla distribuição e existência de múltiplos domínios administrativos, o acesso aos recursos que compõem um Grid não é tão simples.

Quando comparamos o Grid com outras plataformas fica claro que a ampla dispersão de serviços e clientes cria a necessidade de um maior controle sobre o acesso aos serviços e recursos. Por exemplo, em uma rede local, ao efetuar login no sistema, o usuário é identificado e autenticado, em geral, para todos os recursos conectados e serviços disponíveis na rede. Pois, normalmente se mantém um cadastro de usuários que é válido para toda a rede.

Já em Grids, é necessária uma forma de acesso para cada recurso (ou subconjunto de recursos, quando estes compartilham do mesmo cadastro de usuários). Obviamente, tal forma de acesso tem que oferecer garantias sobre autenticação dos usuários, caso contrario os administradores de sistema não serão simpáticos à idéia de permitir que usuários de outros domínios tenham acesso aos recursos locais através dos serviços presentes naquele domínio administrativo.

As iniciativas atuais de Grids têm tratado esse problema através do uso de esquemas baseados em chaves pública e privada, bem como certificados digitais. Desta

forma, cada domínio administrativo pode manter sua política local de autenticação e autorização, e exporta um serviço que cuida da autenticação e autorização do acesso de clientes externos aos seus serviços.

Como veremos em mais detalhes na Seção 13.4.1 e na Seção 13.4.3, algumas infraestruturas possuem uma camada que fornece uma infraestrutura de segurança para que seja possível autenticar e autorizar o acesso e uso de serviços do Grid, enquanto outras usam certificados digitais, não apenas para autenticar usuários, mas também para prover comunicação segura entre os clientes e os serviços.

13.2.4 Privacidade de Dados

Além das demandas por segurança dos provedores de serviços, os clientes desses serviços também impõem necessidades de segurança. Logo, alguns clientes requerem privacidade no tratamento dos seus dados enviados para os serviços. Desta forma, é desejável que apenas o cliente e o serviço que recebe os dados tenham acesso a eles. Várias aplicações necessitam esse nível de privacidade. Garantir esse aspecto é algo desafiador em um ambiente amplamente distribuído e dinâmico como o Grid.

A necessidade de proteção dos dados existe por dois motivos. O primeiro se refere a possibilidade de acesso não autorizado a informações confidenciais. O segundo aspecto, está relacionado a possibilidade de sabotagem, onde outra aplicação modifica os dados a serem processados a fim de manipular os resultados que serão obtidos com aquela computação.

A Entropia [30] provê mecanismos de criptografia para garantir a segurança dos dados da aplicação nos recursos do Grid. Assim, apenas o proprietário do dado armazenado poderá acessar e manipular os dados usando sua chave de criptografia [114]. O problema é que é possível que alguém possa modificar o código da Entropia para obter as chaves armazenadas por eles.

Assim, ainda existem muitos problemas em aberto nessa área. Hoje, instituições que necessitam de um serviço de execução distribuído e têm como requisito principal privacidade não utilizam uma infraestrutura tão dispersa quanto o Grid. Essas aplicações executam em ambientes controlados e proprietários onde a pri-

vacidade pode ser garantida. Um exemplo disso foi a infraestrutura montada pela HP para prover a renderização das imagens do filme Sherek 2 [34].

13.2.5 Composição de Serviço

Em nossa sociedade, estamos bem acostumados com a prestação de serviços por parte de empresas, profissionais e governo. Muitas vezes, para executar um serviço, um prestador de serviço torna-se cliente de outros prestadores, sem que seus clientes tomem conhecimento disso. Uma agência de turismo, por exemplo, vende pacotes de viagem e para prestar este serviço, contrata serviços de companhias aéreas, centrais de intercâmbio estudantil, hotéis, empresas de aluguel de carro, etc. Para o cliente que compra o pacote, o serviço é prestado por uma única entidade. É como se a venda da passagem aérea, de quartos em hotéis, aluguel de carro, fossem feitos pela própria agência de turismo. Todavia, é muito difícil uma agência de viagens se estruturar e funcionar, tendo que ser responsável por tantas atividades, que não são sua atividade fim. O serviço torna-se mais caro e complexo de ser administrado. A estratégia adotada é geralmente terceirizar estas atividades. Assim, a venda do pacote de viagem torna-se uma *composição de serviços* prestados por diversas entidades.

Da mesma forma, em Grids Computacionais, os serviços podem ser compostos por outros serviços. A composição de serviços traz uma série de benefícios para a computação distribuída. Os dois principais são: *i)* Abstração da complexidade do serviço para o cliente; *ii)* Reutilização das funcionalidades implementadas por outros serviços.

Para o cliente, quanto mais complexo for o serviço, menos envolvido ele desejará estar. No exemplo citado anteriormente, sem o trabalho das agências de turismo, o preparo de uma viagem implicará em muito mais trabalho para o cliente. A venda de pacotes turísticos simplifica muito o trabalho dos que pretendem tirar férias. Da mesma forma, no mundo da computação, quanto mais compostos forem os serviços, mais simples e alto nível deve ser programação das aplicações clientes.

O segundo aspecto positivo da composição de serviço é a reutilização de código. Um serviço já desenvolvido e disponível no sistema rede pode ser usado na com-

posição de novos serviços. Desta forma, seu código não tem que ser reimplementado. Um exemplo real são os Web Services fornecidos pela Receita Federal, que permitem consulta e validação de CPF e CNPJ. Estas funcionalidades estão presentes em diversas aplicações por todo o país. Como serviços, elas podem ser utilizadas por essas diversas aplicações, evitando que sejam implementadas novamente.

Entretanto, a composição traz também desafios. Um deles é como um serviço composto pode garantir qualidade, uma vez que ele não é o responsável direto por executar todas as suas etapas. Como no mundo real, para os clientes que usam um serviço, é como se ele fosse único, prestado por um único provedor. No exemplo usado anteriormente, a responsabilidade por atender aos requisitos do cliente que compra o pacote de viagem é do provedor do serviço. O serviço da agência teria que atender os requisitos de segurança, tempo de processamento, limites de custo informados pelo cliente, para o qual, a composição do serviço invocado é transparente.

Para a especificação apropriada de serviços compostos, precisamos de modelos que definam as várias características da composição. Ou seja, a identificação dos possíveis componentes, quando, como e em que condições serão usados, regras para seu funcionamento, dentre outras. Assim, um modelo de composição possui as seguintes dimensões:

- *Modelo de componentes*: define a estrutura dos componentes que fazem parte da composição;
- *Modelo de orquestração*: especifica a ordem em que cada componente deverá ser acionado, e sobre quais condições;
- *Dados e modelo de acesso aos dados*: especifica a estrutura dos dados usados na composição, bem como a forma de acesso a eles;
- *Modelo de seleção de serviço*: especifica como o usuário pode selecionar cada serviço, e o papel que cada componente desempenha na composição;
- *Transações*: especifica o tratamento de transações pela composição - o que deve ser feito quando uma falha ocorre durante a execução de uma transação.

Na tentativa de suprir a demanda por linguagens e ferramentas especialmente voltadas para a composição de serviços, vários trabalhos foram desenvolvidos até então, por exemplo, XLANG [359], WSFL [255] e BPEL4WS [131]. Apesar do alto nível da especificação e riqueza de detalhes que estas linguagens permitem alcançar nas especificações, elas têm sofrido várias críticas da comunidade.

A principal crítica diz respeito ao fato de que ao especificar a composição de serviços, é necessário conhecer antecipadamente todos os serviços que fazem parte da composição, bem como a ordem e condições em que cada tarefa deve ser executada. Isto torna impossível montar novas composições em tempo de execução, automaticamente. Isto abre uma lacuna na área, criando uma demanda por modelos de descrição e ferramentas mais ricos, para que se possa superar este obstáculo, e oferecer serviços cada vez mais complexos e dinâmicos.

13.2.6 Disponibilização de Serviços

Para que Grids sejam úteis, é preciso que eles existam, é preciso criá-los. Ou seja, após ser possível descobrir os serviços, eles devem ser agregados para criar uma infraestrutura de serviços computacionais, que permita a execução de aplicações. Esta sentença é bastante óbvia. Contudo, ela levanta alguns problemas técnicos não-triviais. Suponha que duas instituições *A* e *B* decidem formar um Grid, unificando seus recursos e fornecendo melhores serviços computacionais a seus usuários. Porém, como incentivar domínios administrativos a participarem do Grid com seus serviços e recursos?

Suponha que *A* tem mais que o dobro dos recursos de *B*. Um compartilhamento igualitário seria prejudicial a *A*, que então relutaria em formar um Grid com *B*. Por outro lado, assuma que *A* não pode fornecer acesso a seus recursos durante o expediente bancário (10:00 as 16:00). Como se pode perceber, o contrato entre *A* e *B* para compartilhamento de recursos e construção de um Grid comum pode ser algo bastante sofisticado. Tal sofisticação gera uma pergunta óbvia de como as regras de compartilhamento acordadas serão implementadas e policiadas.

Se a criação de um Grid entre duas instituições pode oferecer tal complexidade, imagine a criação de Grids envolvendo centenas ou milhares de entidades. A abordagem que vem sendo sugerida para este problema é a utilização de mode-

los econômicos para guiar esse processo de criação de Grids [98, 118]. A idéia básica é a construção de um mercado computacional onde as diversas entidades envolvidas no Grid possam trocar recursos. O aspecto mais atraente desta abordagem é que acordos de compartilhamento sofisticados não são mais necessários. Recursos são comprados e vendidos de forma independente, sem um supervisor onisciente do mercado. Desta forma, entidades podem decidir independentemente quando comprar ou vender recursos. Inicialmente, a moeda utilizada será provavelmente algum dinheiro virtual que daria apenas poder de compra de recursos computacionais. Entretanto, é razoável imaginar que o câmbio entre este dinheiro virtual e dinheiro real logo se torne possível, incentivando a criação de empresas que forneçam serviços computacionais sob demanda.

É importante destacar três elementos que formam a base desta economia: provedores de serviços, consumidores de serviços e os serviços propriamente ditos. Note que provedor e consumidor são papéis que podem ser assumidos concorrentemente. Abaixo listamos e discutimos brevemente alguns modelos econômicos [99]:

- *Commodity Market*: Provedores divulgam de forma competitiva seus serviços e os respectivos preços. Consumidores decidem baseado no custo/benefício qual provedor irão contratar.
- *Posted Price Market Model*: Muito similar ao *Commodity Market*, a diferença consiste no tempo que dura a oferta de preços feita pelos provedores. Nesse caso, as ofertas duram mais tempo.
- *Bargaining Model*: Provedores e consumidores negociam preços dos serviços, onde cada um tenta maximizar o resultado de acordo com seus objetivos. Neste caso as negociações são entre pares Consumidor-Provedor. Sendo assim, os consumidores tomam a decisão (contratar ou não) baseado em seus objetivos locais.
- *Tender/Contract Net*: Uma espécie de licitação. Um convite de oferta parte do consumidor para vários provedores que respondem com seus preços e condições de serviço. O consumidor decide qual contratar fazendo a análise do custo do serviço e das condições de atendimento.
- *Auction*: Neste modelo os provedores enviam convites de oferta aos consumidores. Os consumidores pode modificar sua oferta (em incrementos

positivos). A negociação finaliza quando os consumidores param de efetuar ofertas.

- *Bid based Proportional Resource Share*: Neste modelo a quantidade de recursos / serviços é alocada aos consumidores de forma proporcional ao valor de suas ofertas.
- *Community/Coallition/Bartering Model*: A idéia básica deste modelo é a criação de um ambiente cooperativo de compartilhamento de recursos. Ou seja, provedores que contribuem terão garantida a possibilidade de consumir quando necessitarem.

A seguir, apresentaremos estratégias usadas para incentivar a participação de entidades no Grid. A idéia é promover a agregação de recursos de vários domínios administrativos, com o intuito de formar um Grid que beneficie os clientes de cada domínio.

GRACE - GRid Architecture for Computational Economy

É desejável que uma solução para o problema de incentivo à participação no Grid forneça flexibilidade no que se refere as políticas de compartilhamento de recursos. Ou seja, é necessária a existência de mecanismos que garantam o compartilhamento de recursos de forma escalável. Além disso, dever ser possível para o cliente expressar seus requisitos, bem como os provedores expressarem as condições de fornecimento serviço.

Assim, acompanhando a metáfora usada inicialmente baseada na idéia do *The Electric Grid*, que serviu para traçar os objetivos dos Grids Computacionais, a aplicação de modelos econômicos também se baseia no fato que já existem abordagens baseadas em leilão para o mercado de energia elétrica [25].

Portanto, a introdução de modelos de compartilhamento mais sofisticados, baseados em economia, promete uma infraestrutura mais flexível e poderosa para o compartilhamento de recursos e construção de Grids. Um exemplo de investimento nessa área de pesquisa é o GRACE (GRid Architecture for Computational Economy) [99]. A arquitetura do GRACE foi pensada levando em consideração os requisitos que uma infraestrutura de economia computacional deve preencher.

Logo, inspirado pela idéia de mercados, os princípios de projeto da arquitetura são:

1. Um diretório onde seja possível publicar informações sobre as entidades que formam o Grid (i.e. consumidores e provedores), tal como descrito na Seção 13.2.2.
2. Modelos para o estabelecimento de valores para os recursos / serviços.
3. Esquemas de cotação e mecanismos de oferta de serviços.
4. Modelos econômicos e protocolos de negociação de contratação de serviços
5. Mediadores que atuam como reguladores e estabelecem valores para os recursos / serviços, criam moeda padrão e ajudam na resolução de impasses entre os negociadores.
6. Mecanismos para contabilização, cobrança e pagamento.

Para tanto, a arquitetura do GRACE é composta dos seguintes componentes: *a)* Grid Resource Broker (e.g., Nimrod/G); *b)* Grid Resource and Market Information Server; *c)* Grid Open Trading Protocols and API; *d)* Trade Manager; *e)* Grid Trade Server.

O *Resource Broker* funciona como um procurador do usuário (ou de sua aplicação) perante ao Grid. Sendo assim, o Resource Broker desempenha atividades que permitem a execução da aplicação do usuário atendendo seus requisitos (e.g. menor preço pelo serviço de execução). Além disso, um aspecto importante é que o Resource Broker exhibe o Grid para o usuário como um conjunto unificado de recursos, essa abstração facilita a visão do usuário sobre o ambiente.

Certamente, o Resource Broker depende da existência de vários serviços. Por exemplo, serviços de informação sobre os recursos que são oferecidos no Grid, seus preços e condições de uso. Esse serviço é fornecido pelo *Grid Resource and Market Information Server*, o qual utiliza como base o serviço de descoberta de serviços do Globus (i.e. MDS [181]).

Além de obter informação sobre serviços disponíveis e suas cotações, é necessário que haja um padrão (um protocolo bem conhecido pelo cliente e pelo provedor

de serviços) para a negociação. Logo, a arquitetura do GRACE possui um conjunto de protocolos e uma API que define regras e o formato para troca de comandos entre o cliente GRACE (i.e. *Trade Manager*) e o provedor do serviço (*Trade Server*).

Vale mencionar que o *Trade Manager* é uma parte importante do Resource Broker, pois tem o papel de guiar a seleção dos recursos que a aplicação necessita para atingir seus objetivos. Por outro lado, o *Trade Server* é o agente de negociação do lado do provedor, sua função é maximizar a utilização do recursos e o lucro do provedor.

Portanto, a negociação entre os Trade Managers (clientes) e os Trade Servers (provedores) ocorrerá de acordo com algum modelo econômico. Uma das implementações possíveis do GRACE utiliza como *broker* o Nimrod/G [102]. O modelo econômico implementado foi o *Posted Price Market Model* descrito anteriormente [99]. Nesse caso, os vários *Trade Servers* do sistema devem divulgar seus preços, de forma a atrair consumidores, esperando que os *Trade Managers* requeiram o serviço, tomando como base para escolha a comparação de preços entre os preços divulgados.

Rede de Favores

Apesar ser bastante pertinente, a introdução de modelos econômicos a fim de controlar o compartilhamento de recursos entre sites, um grande número de aplicações, que igualmente necessitam de uma infraestrutura de recursos/serviços computacionais de larga escala, não requerem um contrato tão forte entre clientes e provedores, como as providas por uma arquitetura baseada em modelos econômicos. Ao manter o foco neste tipo de aplicação, se torna possível desenvolver uma solução prática que pode ser usada efetivamente por uma comunidade de usuários.

Claramente, estamos apresentando um dilema entre ter uma infraestrutura de escopo mais geral, porém mais complexa o que dificultaria sua implantação efetiva, e uma infraestrutura mais simples, o que facilitaria sua implantação, porém com um escopo mais restrito. Pensando nisso, foi desenvolvida a solução OurGrid [36, 61]. A idéia é ser uma solução simples que permita a criação de Grids Computacionais que fornecem poder computacional seguindo a política

best-effort. O foco está em aplicações Bag-of-Tasks (i.e. aquelas aplicações cujas tarefas são independentes), com essa redução de escopo foi possível ter um Grid Computacional em produção [117]. O OurGrid é formado por três componentes: MyGrid Broker [120], OurGrid Peer [61] e uma solução de Sanboxing baseado no Xen [81].

OurGrid explora a idéia de que um Grid é composto de vários sites que têm o interesse em trocar *favores computacionais* entre si. Portanto, existe uma rede peer-to-peer de troca de favores que permite que os recursos ociosos de um site seja fornecido para outro quando solicitado. Para manter o equilíbrio do sistema, em uma situação de contenção de recursos, sites que doaram mais recursos (quando estes estavam ociosos) deverão ter prioridade junto à comunidade quando solicitar recursos.

A Figura 13.5 ilustra a idéia da rede de favores, onde cada peer controla um conjunto de recursos de um site. Ao surgir uma demanda interna por recursos que o peer de um determinado site não consegue suprir, este PEER irá fazer requisições à comunidade. A idéia é que os peers utilizem um esquema de prioridade baseado em quanto eles consumiram dos outros. Os resultados mostram que haverá um compartilhamento justo de recursos entre os peers [60].

13.2.7 Padronização

Nesta seção exploraremos um pouco mais a visão que motiva boa parte da pesquisa sobre Grids Computacionais atualmente, orientação a serviços. Neste sentido, faremos também um breve histórico sobre os padrões para arquiteturas baseadas em *Grid Services* e qual o estado atual desses esforços.

A visão

A experiência prática adquirida no desenvolvimento dos Grids de alto desempenho tornou possível identificar os problemas que dificultam a implantação de uma infraestrutura com esta escala e complexidade. A questão central que deve guiar o desenvolvimento de tecnologias para Grids Computacionais pode ser entendida como sendo a integração de recursos e serviços distribuídos de forma

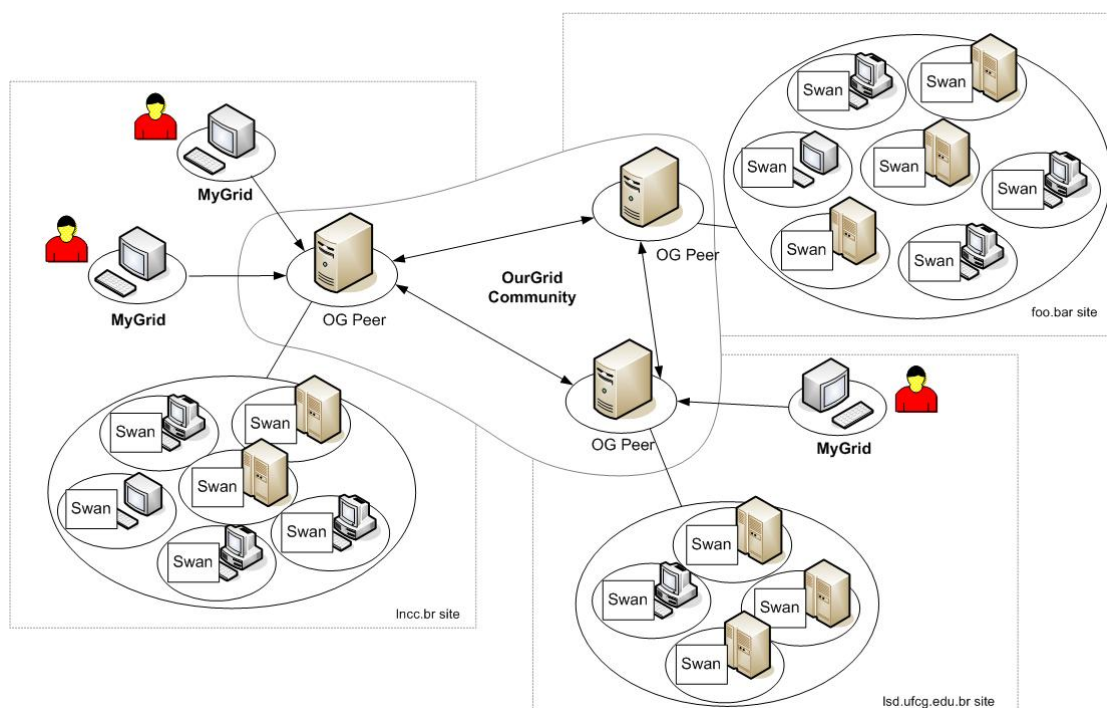


Figura 13.5: Ilustração da arquitetura OurGrid [36]

transparente e eficiente. Assim, foi identificado que este requisito motiva tanto a comunidade científica como a indústria.

Portanto, do lado acadêmico, a crescente necessidade de cooperação científica entre grupos geograficamente distribuídos, através do compartilhamento de recursos e serviços, do outro lado a indústria que tem como necessidade a integração de sistemas comerciais. Essas demandas impulsionaram a convergência de tecnologias de ambas as comunidades.

Como resultado, os Grids evoluíram para utilizar uma abordagem orientada a serviços baseado em padrões e tecnologias para Web Services. Partindo de um conjunto de serviços básicos, como autenticação e transferência de dados, a idéia é a construção de Grids que forneçam de serviços sob demanda.

OGSA/OGSI/Globus 3.x

No intuito de realizar a visão da orientação a serviços, houve um convergência de tecnologias da área de computação de alto desempenho e de padrões bem

consolidados pela indústria, isso ocorreu através da união de tecnologias e conceitos Grids com *web services* [250]. A partir disto foi definida uma arquitetura de serviços básicos para a construção de uma infraestrutura de Grids Computacionais baseados em Serviços. Esta arquitetura foi denominada *Open Grid Services Architecture* (OGSA) [184, 53].

A definição do OGSA contempla a idéia de interconexão de sistemas e a criação de ambientes virtuais multi-institucionais. Além disso, os recursos que podem ser agregados ao Grid são representados por serviços e estes serviços são chamados de *Grid Services* [184]. Os *grid services* são essencialmente *web services* que seguem convenções estabelecidas na especificação da OGSA e suportam interfaces padronizadas para garantir algumas operações adicionais, como gerenciamento do ciclo de vida do serviço.

As interfaces padrões definidas pelo OGSA facilitam a virtualização de recursos e serviços. Isso possibilita o uso de vários tipos de recursos de forma transparente. Outra vantagem da virtualização é que interfaces padrões permitem um baixo acoplamento entre o cliente e o provedor do serviço. Esse baixo acoplamento facilita a mudança na implementação dos serviços sem causar, necessariamente, mudanças na implementação do cliente, bem como o inverso.

Após a definição do modelo da arquitetura e identificação de serviços básicos através do padrão OGSA foi necessária a especificação do comportamento desses serviços. Sendo assim, o passo seguinte foi a especificação dessa infraestrutura serviços básicos, no intuito de permitir a implementação do modelo da arquitetura definida pela OGSA. A nova especificação foi denominada *Open Grid Services Infrastructure* (OGSI) [366] e tem o objetivo de definir as interfaces básicas e os comportamentos de um *Grid Service* [53]. OGSI é a materialização da arquitetura definida pelo padrão OGSA, pois os serviços especificados servem como base para construção dos Grids. Em termos práticos, a especificação OGSI define:

- Um conjunto de extensões para a linguagem WSDL (Web Service Description Language).
- Padrões de estrutura e operação, em WSDL, para representação pesquisa e atualização de dados sobre os serviços.
- As estruturas *Grid Service Handle* e *Grid Service Reference* usados para refe-

reenciar um serviços.

- Formato para mensagens que indicam falhas, sem modificar o modelo de mensagens de falha da linguagem WSDL.
- Conjunto de operações que permitem a criação e destruição de Grid Services. Esse conjuntos de operações devem fornecer destruição explícita de serviços, como também *garbage collection* (destruição implícita do serviço).
- Conjunto de operações para criação e uso de coleções de Web Services por referência.
- Mecanismos que permitam notificação assíncrona, caso haja mudança em valores dos elementos de dados dos serviços.

O *Globus Toolkit 3.0* (GT3) [181] forneceu a primeira implementação da OGSi 1.0 em julho de 2003. No intuito de esclarecer melhor o papel de OGSA, OGSi e Globus, Figura 13.6 ilustra como essas entidades se relacionam formando um cenário de padrões e implementações de tecnologias para Grids de Serviço.

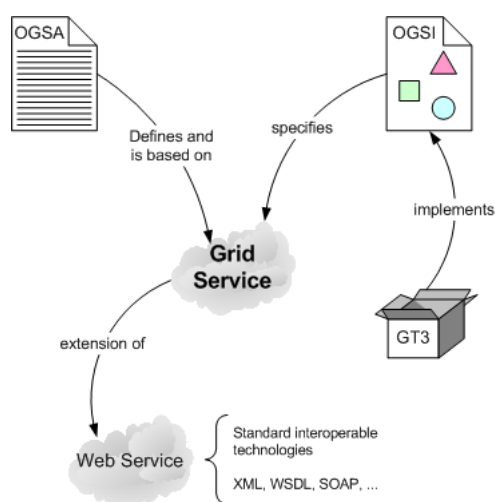


Figura 13.6: Relacionamento entre OGSA, OGSi e Globus [343]

Note, portanto, que *Grid Service* é um conceito central no relacionamento destas tecnologias e padrões. Assim, OGSA define Grid Services, OGSi especifica o comportamento de Grid Services, Web Services são estendidos para se tornar Grid Services e Globus 3 implementa a especificação OGSi. Além disso, Globus provê

serviços básicos necessários para a construção de serviços de mais alto nível (e.g. serviços de autenticação via GSI).

WSRF/**Globus 4.x**

Uma vez que toda a base de *Grid Services* surgiu das tecnologias para Web Services, alguns aspectos da especificação OGSi precisavam ser refinados devido a evolução da arquitetura Web Services.

O principal ponto de crítica foram os mecanismos de endereçamento para os serviços (i.e. *Grid Service Handler* e *Grid Service Reference*). Nesse caso, *WS-Addressing* surgiu pra fornecer um mecanismo de endereçamento independente da camada de transporte [132].

Além disso, outras características de OGSi precisavam ser modificadas para acompanhar a evolução da tecnologia Web Service, melhorando assim a especificação OGSi. Assim, WSRF (*Web Service Resource Framework*) é basicamente o resultado do refinamento de OGSi no intuito de aproveitar a existência dos novos padrões que surgiram para para Web Services (e.g. *WS-Addressing*, *WS-Notification*) e assimilar a demanda da comunidade Web Services.

O primeiro efeito do refatoramento foi a divisão de OGSi em várias especificações separadas, porém agrupadas em uma família. A idéia é reduzir a complexidade de uma especificação longa, que dificulta a adoção incremental de funcionalidades.

Outra medida importante foi a recuperação da compatibilidade com as ferramentas existentes para XML e Web Services, pois OGSi usa GWSDL, a qual provê acréscimos à WSDL 1.1 que estarão disponíveis na WSDL 1.2/2.0. Ao contrário de OGSi, ao invés de estender a definição de *portType* WSDL 1.1, ou mesmo esperar pelo da nova versão WSDL 2.0, *WS-Resource Framework* utiliza puramente WSDL 1.1, o que garante compatibilidade com as ferramentas existentes para Web Services.

Alguns entusiastas da área de Web Services, em geral, argumentam que *Web Services não devem manter estado ou ter instâncias*. Ou seja, OGSi modela um recurso que mantém estado como um Web Service que encapsula o estado do recurso.

WS-Resource Framework modifica esse modelo para criar uma distinção explícita entre *serviço* e entidades que mantêm estado e são manipuladas através do serviço. Esta composição é denominada *WS-Resource* pelo padrão WSRF, que introduz a idéia de recursos que mantêm estados e podem ser acessados através de Web Services via o uso convencional de WS-Addressing.

Portanto, em linhas gerais, a evolução de OGSi obedeceu três fases de forma incremental. A primeira, a introdução do conceito de *WS-Resource*. Em seguida a divisão de funcionalidades em várias especificações melhorando a compatibilidade com ferramentas usadas para Web Service. Finalmente, uma melhoria nos mecanismos de notificação.

O padrão WSRF deve se materializar, de forma estável, através do lançamento do Globus 4. Atualmente, Março de 2005, o Globus se encontra na versão 3.9.5, que apesar de instável já incorpora várias das funcionalidades contempladas no padrão WSRF. Por exemplo, o GRAM do Globus 3, será substituído pelo WS-GRAM, o qual segue as especificações definidas na família de padrões WSRF.

13.3 Grids para Alto Desempenho

Neste capítulo os Grids Computacionais são apresentados como uma plataforma de execução para aplicações paralelas. Sendo assim, faremos comparações com plataformas de execução convencionais. Em seguida definiremos quais são os tipos de aplicações paralelas mais adequadas para executar em um Grid Computacional. Finalmente, apresentaremos dois estudos de caso.

13.3.1 Plataformas para Processamento Paralelo

Uma aplicação paralela é composta por várias tarefas. As tarefas que compõem uma aplicação paralela executam em vários processadores, caracterizando desta forma o paralelismo da execução da aplicação e conseqüente redução no seu tempo de execução. Os processadores usados por uma determinada aplicação constituem a plataforma de execução da aplicação.

Plataformas de execução de aplicações paralelas variam em diversos aspectos, dos quais destacamos conectividade, heterogeneidade, compartilhamento, imagem do sistema e escala.

- *Conectividade* diz respeito aos canais de comunicação que interligam os processadores que compõem a plataforma de execução. Atributos que definem a conectividade de uma plataforma são a topologia, largura de banda, latência e compartilhamento.
- *Heterogeneidade* trata das diferenças entre os processadores, que podem ser de velocidade e/ou arquitetura.
- *Compartilhamento* versa sobre a possibilidade dos recursos usados por uma aplicação serem compartilhados por outras aplicações.
- *Imagem do sistema* se refere à existência de uma visão única da plataforma, independente do processador sendo utilizado. Por exemplo, todos os processadores da plataforma enxergam o mesmo sistema de arquivos?
- *Escala* estabelece quantos processadores tem a plataforma.

Entender as diferenças entre plataformas é fundamental porque cada aplicação paralela tem uma série de requisitos, que podem ser melhor ou pior atendidos por uma dada plataforma. Em princípio, procuramos executar uma aplicação paralela em uma plataforma adequada às características da aplicação. Por exemplo, considere conectividade, um aspecto em que plataformas diferem consideravelmente. Obviamente, para obter boa performance de uma aplicação paralela cujas tarefas se comunicam e sincronizam freqüentemente, necessitamos utilizar uma plataforma de execução com boa conectividade.

Podemos agrupar as plataformas de execução hoje existentes em quatro grandes grupos: SMPs, MPPs, NOWs e Grids. SMPs (ou multiprocessadores simétricos) são máquinas em que vários processadores compartilham a mesma memória. Multiprocessadores possibilitam um fortíssimo acoplamento entre os processadores e executam uma única cópia do sistema operacional para todos os processadores. Portanto, eles apresentam uma imagem única do sistema e excelente conectividade. Todavia, multiprocessadores apresentam limitações em escalabilidade, raramente ultrapassando algumas dezenas de processadores. Multipro-

cessadores são relativamente comuns no mercado e vão desde máquinas biprocessadas Intel até grandes servidores como os da série HP 9000. A Figura 13.7 ilustra a arquitetura de um multiprocessador.

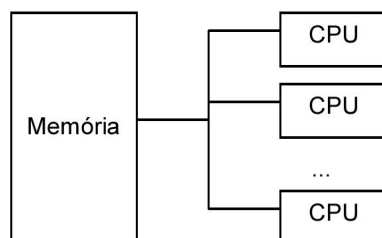


Figura 13.7: Arquitetura multiprocessada

MPPs (ou processadores maciçamente paralelos) são compostos por vários nós (processador e memória) independentes, interconectados por redes dedicadas e muito rápidas. MPPs incluem supercomputadores paralelos como o IBM SP2 e Cray T3E, como também clusters de menor porte montados pelo próprio usuário. Tipicamente cada nó roda sua própria cópia do sistema operacional, mas uma imagem comum do sistema é implementada através da visibilidade dos mesmos sistemas de arquivo por todos os nós. O MPP é controlado por um escalonador que determina quais aplicações executarão em quais nós. Ou seja, não se pode utilizar um nó que não tenha sido alocado à aplicação pelo escalonador. Isto possibilita dedicar partições (um conjunto de nós) às aplicações, permitindo que estas não precisem considerar compartilhamento. Mas, uma vez que aplicações executam em partições dedicadas, pode acontecer que não haja nós suficientes para executar uma aplicação assim que ela é submetida. Neste caso, a aplicação espera em uma fila até que os recursos que solicitou estejam disponíveis. A Figura 13.8 exemplifica a arquitetura de um MPP.

NOWs (ou redes de estações de trabalho) são simplesmente um conjunto de estações de trabalho ou PCs, ligados por uma rede local. NOWs são arquiteturalmente semelhantes aos MPPs. Ambas plataformas são formadas por nós que agregam processador e memória. Uma diferença entre NOWs e MPPs é que os nós que compõem uma MPP tipicamente são conectados por redes mais rápidas que as que conectam os nós de NOWs. Mas a principal diferença entre ambas arquiteturas é que NOWs não são escalonadas de forma centralizada. Isto é, em uma NOW, não há um escalonador para o sistema como um todo. Cada nó tem

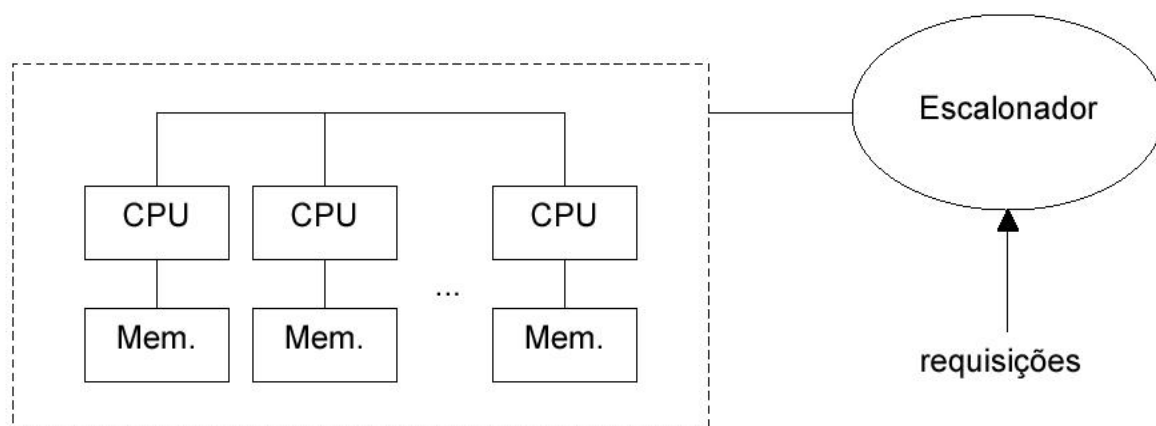


Figura 13.8: Arquitetura de um MPP

seu próprio escalonador local. Como resultado, não há como dedicar uma partição da NOW a uma só aplicação paralela. Assim, uma aplicação que executa sobre a NOW deve considerar o impacto da concorrência por recursos por parte de outras aplicações sobre sua performance. A Figura 13.9 mostra esquematicamente uma NOW.

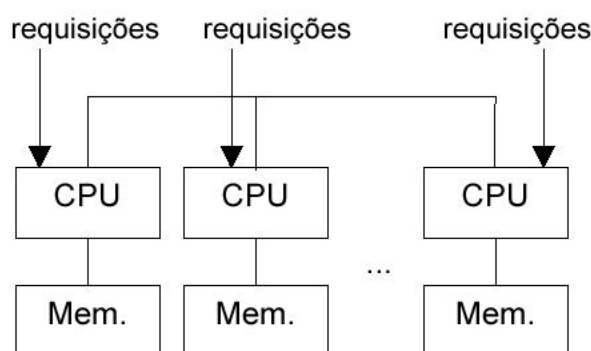


Figura 13.9: Arquitetura de uma NOW

Grids são o passo natural depois dos NOWs no sentido de mais heterogeneidade e maior distribuição. Os componentes de um Grid não se restringem a processadores, podendo ser SMPs e MPPs como também instrumentos digitais. Grids tipicamente não fornecem uma imagem comum do sistema para seus usuários. Componentes do Grid podem variar drasticamente em capacidade, software instalado, sistemas de arquivo montados e periféricos instalados. Além disso, os

componentes de um Grid tipicamente estar sobre controle de diferentes entidades e, portanto, em domínios administrativos diversos. Conseqüentemente, um dado usuário pode ter acesso e permissões bastante diversas nos diferentes componentes de um Grid. Obviamente, o Grid não pode ser dedicado a um usuário, embora seja possível que algum componente possa ser dedicado (um MPP, por exemplo). É importante salientar que uma aplicação que executa sobre o Grid deve estar preparada para lidar com todo este dinamismo e variabilidade da plataforma de execução, adaptando-se ao cenário que se apresenta com o intuito de obter a melhor performance possível no momento. A Figura 13.10 exemplifica um possível Grid, composto por um MPP e computadores de vários tipos conectados via Internet. Note que um destes computadores realiza instrumentação (no exemplo, através de um microscópio), enquanto outro computador dispõe de grande capacidade para armazenamento de dados.

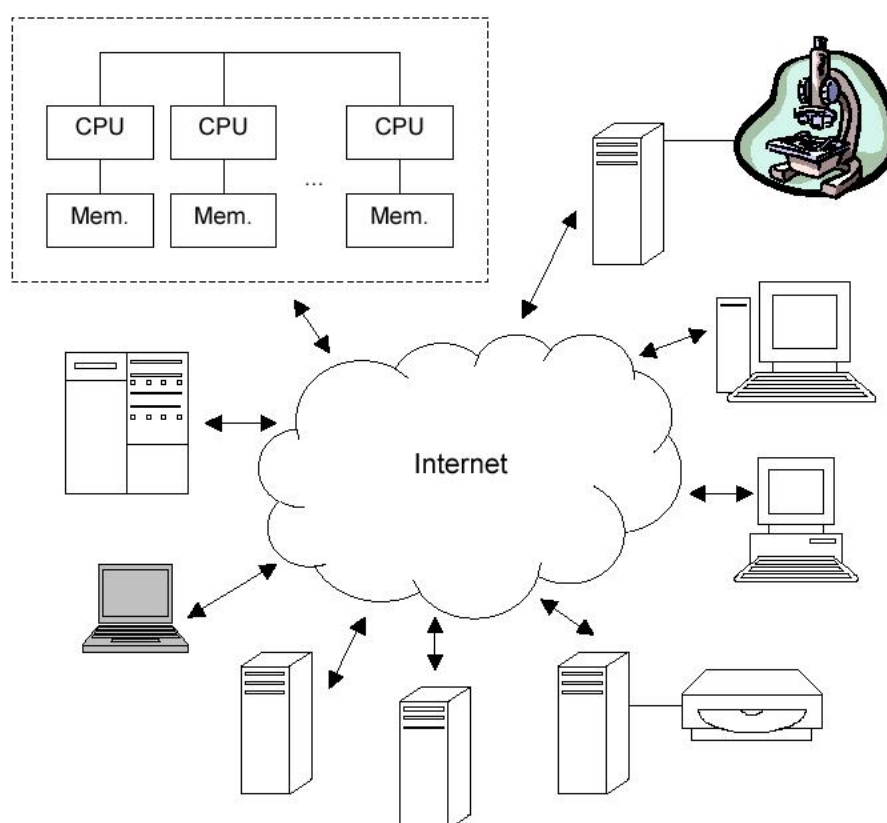


Figura 13.10: Arquitetura de um Grid Computacional

A Tabela 13.1 resume as características das plataformas de execução de aplicações paralelas discutidas aqui. Mantenha em mente, entretanto, que a Tabela 13.1

descreve características típicas dos diferentes tipos de plataformas de execução. Certas plataformas podem apresentar características arquiteturais adicionais que impactam na performance das aplicações paralelas que nela executam. Por exemplo, alguns MPPs oferecem suporte de hardware a memória compartilhada, através de uma tecnologia denominada DSM (Distributed Shared Memory), o que melhora o desempenho de aplicações baseadas em memória compartilhada. Uma vez que Grids são o nosso foco neste texto, caso o leitor queira mais detalhes sobre plataformas de execução de aplicações paralelas tradicionais (SMPs, MPPs e NOWs) sugerimos a leitura do trabalho de De Rose e Navaux [312].

Mesmo quando não há distinções arquiteturais, diferentes plataformas do mesmo tipo podem diferir consideravelmente. Em particular, um Grid pode diferir radicalmente de outro. Por exemplo, considere o TeraGrid [38], o SETI@home [59] e o PAUÁ [117].

O TeraGrid é um Grid que interliga 10 centros de supercomputação norteamericanos através de canais de altíssima velocidade (40 GigaBits/segundo). Cada um dos centros possui milhares de processadores dedicados ao TeraGrid, gerando um poder agregado de aproximadamente 20 TeraFlops. O SETI@home, por outro lado, utiliza a capacidade computacional ociosa de computadores que se juntam voluntariamente ao sistema através da instalação do software cliente do projeto. Em Março de 2005, SETI@home contava com aproximadamente 5.3 milhões de processadores espalhados em 226 países. O PAUÁ, por sua vez, tem uma escala intermediária, pois congrega 10 domínios administrativos espalhados pelo Brasil (ver <http://pauastatus.lsd.ufcg.edu.br>) e tem como infraestrutura o OurGrid, que se baseia em uma rede peer-to-peer para o compartilhamento de recursos entre os sites (i.e. Rede de Favores [61]). Apenas considerando essas breves descrições é notável a diferença entre os três Grids. Outro aspecto interessante de se verificar é que apesar do TeraGrid congregar um número semelhante de sites, comparado ao PAUÁ, o TeraGrid tem muito mais recursos PAUÁ.

O conceito de acoplamento do Grid (i.e. quão próximos estão seus componentes) é fundamental para compreendermos quais aplicações podem executar eficientemente em um Grid. Note que acoplamento tem uma relação com conectividade. Voltando ao exemplo acima, o SETI@home e o PAUÁ têm seus focos em aplicações fracamente acopladas, enquanto o TeraGrid pode propiciar condições para execução eficiente de aplicações fortemente acopladas).

	SMP	MPP	NOV	Grid
Conectividade	excelente	muito boa	boa	regular/ruim
Heterogeneidade	nula	baixa	média	alta
Compartilhado	não	não	sim	sim
Imagem	única	comum	comum	múltipla
Escala	10	1.000	1.000	100.000

Tabela 13.1: Comparação entre as plataformas de execução para aplicações paralelas

13.3.2 Execução Remota

Na Seção 13.3.1 apresentamos o Grid como uma plataforma de execução de aplicações paralelas. Além disso, apresentamos o que diferencia os Grids de outras plataformas mais tradicionais para processamento de alto desempenho. Vale ressaltar que o componente fundamental dos Grids para Alto Desempenho é o *serviço de execução remota*. Este serviço é responsável por qualificar o grid como plataforma de execução de aplicações paralelas.

Um Grid que fornece serviços de execução remota possui várias vantagens. Uma delas é a possibilidade de converter este serviço genérico de execução de aplicações em qualquer outro serviço mais específico. Por exemplo, oferecer um serviço de processamento de imagens que utiliza várias instâncias do serviço de execução remota dispersos por todo Grid.

Em contrapartida, a introdução dessa flexibilidade adquirida através do fornecimento de um serviço de execução genérico, que pode ser convertido em outros serviços mais específicos, aumenta a complexidade da infraestrutura. Portanto, novas questões devem ser consideradas para que seja possível fornecer um serviço de execução remota eficiente. Por exemplo, quais serviços de execução remota, disponíveis no Grid, devem ser usados, de forma a obter uma execução eficiente da aplicação de processamento de imagens? Como proteger o serviço de aplicações maliciosas? Discutiremos várias dessas questões nas próximas seções.

13.3.3 Escalonamento

Um dos aspectos mais importantes para o desenvolvimento dos Grids é a implementação de estratégias de alocação de recursos para as várias aplicações que usam a infraestrutura. Sendo assim, tanto é possível pensar em escalonamento

de aplicações, bem como escalonamento de recursos. Em geral, os objetivos dessas duas visões contrastam entre si. A primeira procura melhorar o desempenho da aplicação. A segunda busca aumentar a utilização dos recursos. Nesta seção discutiremos essas duas visões sobre escalonamento, bem como heurísticas de escalonamento de aplicação.

Escalonamento de aplicação vs. Escalonamento de recurso

Tradicionalmente, há um escalonador que controla os recursos do sistema (i.e., não há como usar os recursos sem a autorização do escalonador). Por exemplo, o sistema operacional controla o computador no qual roda, decidindo quando e aonde (no caso de multiprocessadores) cada processo executa. Chamaremos estes escalonadores de *escalonadores de recursos*. Uma característica importante dos escalonadores de recurso é que eles recebem solicitações de vários usuários e, portanto, tem que arbitrar entre estes vários usuários o uso dos recursos que controlam.

Devido à grande escala, ampla distribuição e existência de múltiplos domínios administrativos, não é possível construir um escalonador de recursos global para Grids. Uma razão para isto é que sistemas distribuídos que dependem de uma visão global coerente (necessária ao controle dos recursos) apresentam problemas de escalabilidade. Além disso, é muito difícil (senão impossível) convencer os administradores dos recursos que compõem o Grid a abrir mão do controle de seus recursos.

Assim sendo, para utilizar recursos controlados por vários escalonadores de recurso distintos, alguém tem que (i) escolher quais recursos serão utilizados na execução da aplicação, (ii) estabelecer quais tarefas cada um destes recursos realizará, e (iii) submeter solicitações aos escalonadores de recurso apropriados para que estas tarefas sejam executadas. Esta é o papel do *escalonador de aplicação*. Escalonadores de aplicação não controlam os recursos que usam. Eles obtêm acesso a tais recursos submetendo solicitações para os escalonadores que controlam os recursos.

Ou seja, em um Grid, as decisões de escalonamento são divididas em duas camadas, com parte da responsabilidade pelo escalonamento sendo transferida dos

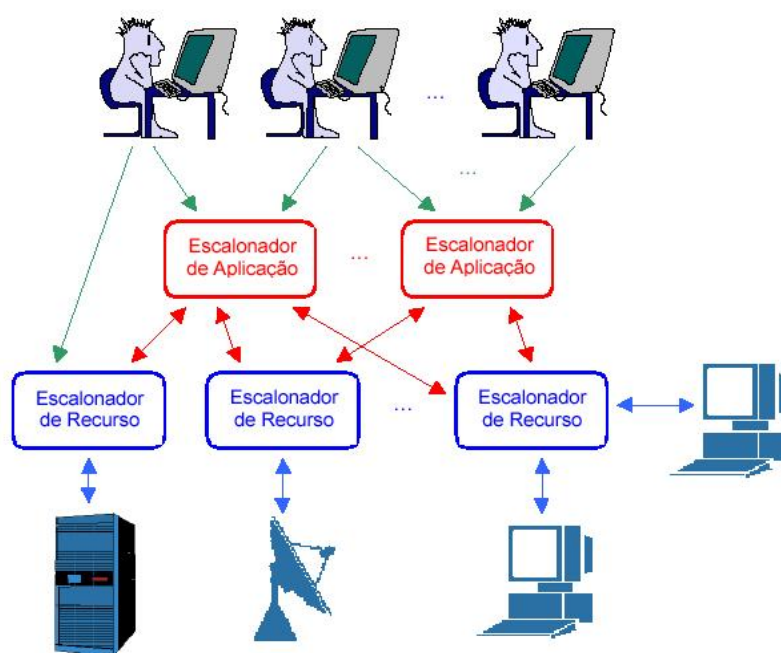


Figura 13.11: Ilustração de um cenário composto de vários escalonadores

escalonadores de recurso para o nível de aplicação. A Figura 13.11 ilustra um cenário de escalonamento em um Grid Computacional.

As decisões tomadas pelo escalonador de aplicações (quais recursos serão utilizados e quais tarefas cada um destes recursos realizará) são normalmente baseadas em um modelo do desempenho da aplicação e em dados sobre o estado atual dos vários recursos que formam o Grid [89, 111, 327, 380, 379, 167]. Portanto, escalonadores de aplicação têm que conhecer detalhes das aplicações que escalonam, i.e. eles são construídos com uma aplicação (ou classe de aplicações) em mente. Além disso, escalonadores de aplicação normalmente precisam saber quanto tempo cada recurso vai levar para processar uma dada tarefa. Sem esta informação, é difícil escolher os melhores recursos a utilizar, como também determinar qual tarefa alocar a cada um desses recursos. Para obter tal informação, foram desenvolvidos sistemas que monitoram e prevêm o comportamento futuro de diversos tipos de recursos [262, 389]. Este esquema de obtenção de informação baseado em monitoração tem se mostrado eficaz quando os recursos monitorados são redes TCP/IP ou computadores compartilhados no tempo, mas ainda há questões quanto a escalabilidade dos sistemas de monitoração [189].

Previsão de Desempenho

Apesar de serem úteis e ajudarem no desenvolvimento de heurísticas de escalonamento eficientes, as infraestruturas de monitoração e previsão de desempenho têm um desafio imposto pela escala que um Grid computacional pode alcançar. Além disso, devido a descentralização do controle sobre os recursos no Grid, é possível que por questões de segurança alguns domínios administrativos não adotem a implantação de sistemas de monitoração, a fim de fornecer informações de previsão de desempenho para escalonadores de aplicação.

Mesmo assim, é pensando na possibilidade de prover um melhor desempenho no escalonamento de aplicações que alguns sistemas de previsão de desempenho foram desenvolvidos. Como consequência várias heurísticas foram desenvolvidas dependendo das informações fornecidas por estas infraestruturas de monitoração [111, 89].

Uma serviço utilizado por várias heurísticas de escalonamento é o *Network Weather Service* (NWS) [389]. O NWS é um sistema que monitora e dinamicamente provê estimativas de desempenho de recursos computacionais (ex.: processadores e rede). O processo de monitoração é feito através de sensores que mede periodicamente o estado dos recursos. As informações coletadas pelos sensores podem ser requisitadas diretamente pelas heurísticas, que utilizam essa informação para melhorar a eficiência do escalonamento gerado.

Além da informação sobre o desempenho de um recurso, em um dado instante, fornecido pelos sensores, as heurísticas podem fazer uso de previsões de desempenho que são geradas pelo NWS a partir do histórico obtido com a monitoração. Assim, através do uso de métodos numéricos (e.g. regressão linear) as informações armazenadas sobre o desempenho dos recursos são processadas para gerar estimativas do desempenho que os recursos podem oferecer em um determinado intervalo.

Considerando o benefício que uma arquitetura de monitoração, que forneça informações sobre os recursos do Grid, o Global Grid Fórum [196] mantém grupos de trabalho discutindo sobre questões relacionadas à definição de uma arquitetura de monitoração. Estas discussões são, em geral, baseadas na experiência obtida de investimentos já feitos por vários grupos, como NWS [389] e *Autopilot* [309],

dentre outros trabalhos [375, 336, 361]. A arquitetura proposta é baseada na idéia de *produtor/consumidor* de eventos disparados pelos sensores que monitoram os recursos [376]. Apesar da evolução conceitual que a padronização pode introduzir na área de previsão de performance para sistemas distribuídos, muito ainda tem que ser feito para se ter uma infraestrutura amplamente disponível.

A seguir descreveremos algumas situações onde um serviço de previsão de desempenho é extremamente útil, bem como estratégias eficientes de escalonamento de aplicação que não dependem de informação sobre os recursos do Grid nem da aplicação.

Aplicações Fortemente Acopladas

Jacobi AppLeS [89] é um exemplo interessante, primeiro por ter introduzido a idéia de escalonamento de aplicações e também por fornecer uma solução de escalonamento para uma aplicação bem conhecida (i.e. Jacobi).

Jacobi é um método usado para resolver a aproximação por diferenças finitas da equação de Poisson e portanto é aplicável a problemas que envolvem fluxo de calor, eletrostática e gravitação. Além de ser interessante por si só, Jacobi pode ser visto como uma instância de uma importante classe de aplicações paralelas: aplicações fortemente acopladas de paralelismo em dados.

Jacobi AppLeS é um escalonador para Jacobi 2D. Em Jacobi 2D, o domínio do problema é discretizado em uma matriz bidimensional. Em cada iteração, cada elemento da matriz é atualizado com a média dos seus quatro vizinhos. Jacobi termina por convergência, isto é, quando uma iteração altera muito pouco os elementos da matriz.

Quando Jacobi é executado em um MPP (*Massive Parallel Processor*), a matriz bidimensional é tipicamente dividida em ambas as dimensões, gerando submatrizes de igual tamanho. Cada submatriz é então alocada a um processador. A cada iteração, portanto, é necessária comunicação entre processadores para troca das fronteiras das submatrizes. A Figura 13.12 mostra a distribuição de dados entre 4 processadores de um MPP alocados para executar Jacobi. Como, em um MPP, os processadores são idênticos e dedicados, esta simples estratégia de alocação de

trabalho balanceia a carga entre os processadores, garantindo bom desempenho.

Em um Grid, entretanto, processadores e canais de comunicação são heterogêneos. Além disso, outras aplicações estão concorrendo pelos mesmos recursos (processadores e canais de comunicação) enquanto Jacobi executa. Conseqüentemente, a estratégia descrita acima provavelmente vai produzir um desbalanço de carga, afetando o desempenho. Mais ainda, uma vez que as condições de carga do Grid variam dinamicamente, o que é uma boa divisão de carga vai variar a cada execução da aplicação. Finalmente, devido à existência de canais de comunicação mais lentos e compartilhados com outras aplicações, talvez não valha a pena utilizar todos os processadores disponíveis.

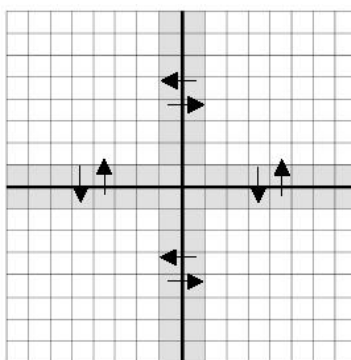


Figura 13.12: Jacobi executando em quatro processadores em um MPP

A solução oferecida por AppLes Jacobi se baseia em três elementos principais. Primeiro, o escalonamento em si é simplificado pela decisão de utilizar um particionamento unidimensional. Segundo, o escalonador se utiliza do NWS [389] para obter previsões de curto prazo da disponibilidade de cada processador e da latência e banda da comunicação entre quaisquer dois processadores. Terceiro, o escalonador dispõe de um modelo de performance da aplicação, que é usado para avaliar suas decisões. Este modelo é o seguinte:

$$T_i = A_i \times P_i + C_i \quad (13.1)$$

- T_i é o tempo para o processador i executar uma iteração.
- A_i é a área da submatriz alocada ao processador i .

- P_i é o tempo que o processador i leva para computar um elemento.
- C_i é o tempo que o processador i leva para comunicar suas fronteiras.

Note que P_i e C_i são estimados com base nos dados fornecidos pelo NWS.

O escalonamento propriamente dito começa ordenando os processadores por uma distância específica da aplicação (que cresce quadraticamente com a diferença de velocidade dos processadores e linearmente com a diferença de suas capacidades de comunicação). Uma vez os processadores ordenados, tenta-se iterativamente uma solução com os n primeiros processadores, até que a solução com $n - 1$ processadores se mostre mais rápida, ou até que não haja mais processadores. Naturalmente, o tempo de uma iteração é estimado como o maior T_i de todos os processadores. Fixados n processadores, a solução de escalonamento é obtida dividindo a matriz proporcionalmente a P_i .

Por exemplo, suponha que o Grid tem quatro processadores: P_0 , P_1 , P_2 e P_3 . Assuma ainda que P_0 e P_1 tem o dobro da velocidade de P_2 e P_3 , que P_1 tem uma outra aplicação rodando e só poderá dedicar 50% de seu poder computacional a aplicação, que P_3 está conectado a uma rede que vivencia intenso tráfego e que sua comunicação está ordens de grandeza mais lenta que entre os demais processadores. Uma vez que P_3 está se comunicando muito lentamente, o AppLeS não vai utilizá-lo para esta execução. Note que esta decisão não descarta a possibilidade que P_3 venha a ser usado em uma futura execução da aplicação, quando as condições da rede forem diferentes. Note também que, embora P_1 seja duas vezes mais rápido que P_2 , uma vez que só 50% de P_1 está disponível, P_1 e P_2 são idênticos para a aplicação (pelo menos nesta execução). A Figura 13.13 mostra o resultado que o AppLeS Jacobi produziria neste cenário.

Devemos salientar que um aspecto importante para o bom funcionamento do Jacobi AppLeS é o tempo relativamente curto de execução da aplicação. O tempo de execução dos experimentos descritos em [89] é da ordem de segundos, casando perfeitamente com as previsões de curto prazo do NWS. Para aplicações que executam por mais tempo (horas, digamos), seria necessário prever a disponibilidade de recursos do Grid por prazos mais longos. Uma alternativa interessante seria construir um escalonador que, além do escalonamento inicial, continuasse funcionando para reescalonar a aplicação caso as condições do Grid mudassem consideravelmente [327]. Neste caso, naturalmente, a aplicação teria que ser es-

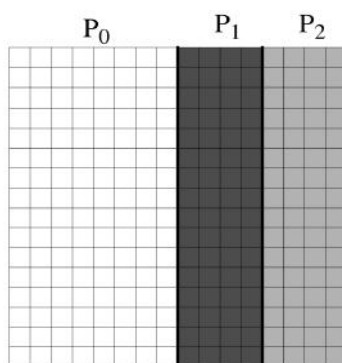


Figura 13.13: Escalonamento feito pelo Jacobi AppLes

crita para permitir tal reescalonamento, suportando a redistribuição de trabalho durante a execução.

Note que as previsões de performance fornecidas pelo NWS são fundamentais para o funcionamento do Jacobi AppLeS. De fato, a vasta maioria dos escalonadores de aplicação descritos na literatura [89, 111, 327, 380, 379, 167] utiliza alguma forma de previsão de performance. Infelizmente, há problemas em aplicar em larga escala os sistemas de monitoração e previsão existentes (como NWS [389] e Remos [377]), especialmente quando se trata de prever o comportamento dos canais de comunicação, que crescem quadraticamente com o número de máquinas do Grid. Em Francis, et al. [189] é apresentada uma discussão sobre a dificuldade e se construir um sistema escalável para monitoramento de canais de comunicação.

Aplicações *Bag-of-Tasks*

Apesar de ser possível efetuar escalonamentos eficientes usando informação sobre o desempenho dos recursos, muitas aplicações podem ser escalonadas de forma eficiente sem o uso dessa informação. Isso é possível devido a algumas características da aplicação. Em particular, aplicações *Bag-of-Tasks* são aplicações que podem ser escalonadas sem o uso de informação dinâmica sobre o Grid (e.g. carga de CPU, largura de banda).

Como parte do projeto OurGrid [36], foram desenvolvidas duas heurísticas de escalonamento, o Work Queue with Replication (WQR) [297], um escalonador ca-

paz de obter boa performance para a aplicação no ambiente muito dinâmico que é o Grid sem depender de previsões de performance dos componentes do Grid. Isto é possível devido a dois fatores fundamentais. Primeiro, WQR usa replicação de tarefas para garantir a boa performance da aplicação. A idéia é utilizar alguns ciclos extra para compensar pela falta de informação sobre o ambiente. Segundo, WQR escala aplicações relativamente simples.

A idéia aplicada na heurística WQR é bastante simples e ao mesmo tempo poderosa. Uma fila de tarefas é criada na submissão da aplicação. Sempre que há um processador disponível, uma tarefa é enviada para este processador. Quando não há mais tarefas para enviar (i.e. a fila de tarefas está vazia), uma das tarefas em execução é replicada. Quando uma das replicas termina, as demais réplicas são abortadas pelo escalonador. Para evitar o desperdício de poder computacional, é estabelecido o máximo de replicas que uma tarefa pode ter. Nossos experimentos mostraram um resultado bastante animador, eles indicam que grande parte do ganho de desempenho obtido pelo WQR se manifesta com um grau de replicação 2.

Considere a Figura 13.14, que mostra o desempenho do WQR em comparação com o *Workqueue*, o Sufferage [226] (um bom escalonador baseado em informações sobre o Grid e sobre as tarefas), e com o Dynamic FPLTF (Dynamic Fastest Processor to Largest Task First, outro bom escalonador que utiliza informações sobre o Grid e sobre as tarefas). Este resultado apresenta o tempo médio obtido pelos quatro algoritmos de escalonamento em função da heterogeneidade das tarefas (quanto maior, mais heterogêneo). Note que WQR foi executado três vezes, com replicação máxima de 2, 3 e 4 processadores. Observe também que Sufferage e Dynamic FPLTF tiveram informação perfeita sobre o desempenho dos recursos do Grid, bem como das tarefas que formam a aplicação, algo inatingível na prática. Portanto, é um excelente resultado o fato de WQR obter desempenho comparável com Sufferage e Dynamic FPLTF baseados em informação perfeita.

Obviamente, WQR consome mais ciclos que os algoritmos de escalonamento tradicionais. A Figura 10 mostra o consumo adicional de CPU em função da heterogeneidade das tarefas. Em situações que a heterogeneidade de tarefas é pequena, este consumo não é significativo, não ultrapassando 15%. Por outro lado, quando tarefas são muito heterogêneas, WQR desperdiça uma quantidade considerável de ciclos. De fato, o desperdício pode chegar próximo a 100%. Entretanto, tal

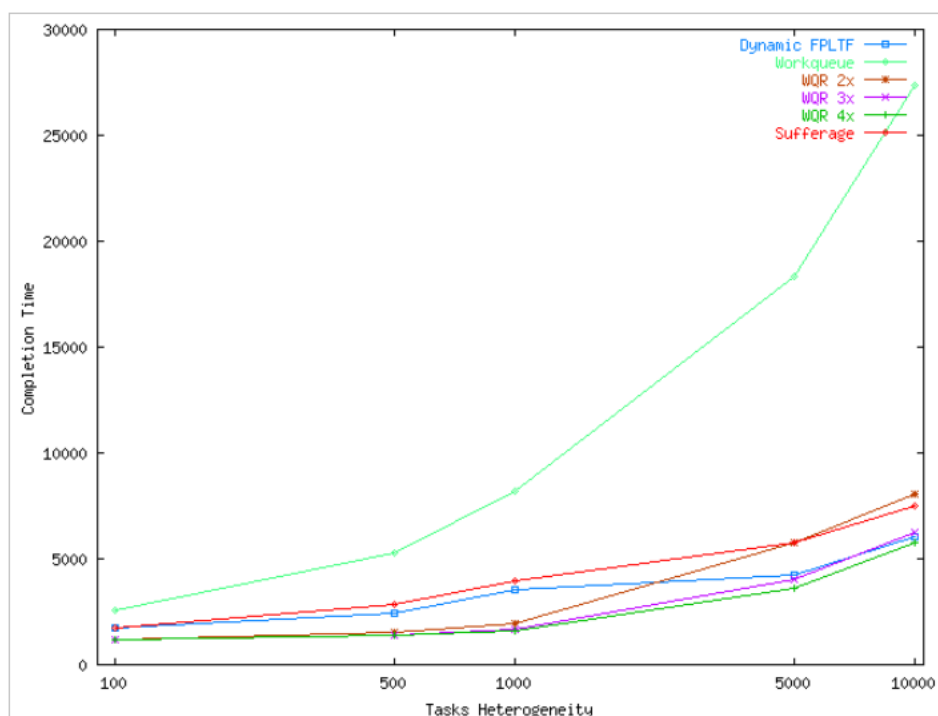


Figura 13.14: Desempenho do WQR

problema pode ser controlado pela limitação do número máximo de réplicas de uma tarefa. Quando limitamos WQR a usar 2 réplicas (WQR 2x), temos que o desperdício de CPU fica sempre abaixo de 40%.

Aplicações que Processam Grandes Quantidades de Dados

Apesar de WQR ser uma heurística eficiente, ela apresenta uma limitação, o bom desempenho é alcançado apenas para aplicações *cpu-intensive*. Ou seja, caso a aplicação necessite processar grandes quantidades de dados, o que implicará em muitas transferências, a replicação pode não ter o mesmo efeito.

Uma grande parte das aplicações Bag-of-Tasks necessitam processar grandes quantidades de dados. Por exemplo, bioinformática [54], Seti@Home [59], renderização de imagens [139, 254, 320, 207]. Sendo assim, uma heurística de escalonamento que melhore o desempenho dessas aplicações é bastante relevante.

Felizmente, algumas aplicações apresentam características que podem ser exploradas por heurísticas de escalonamento para obter escalonamentos eficientes.

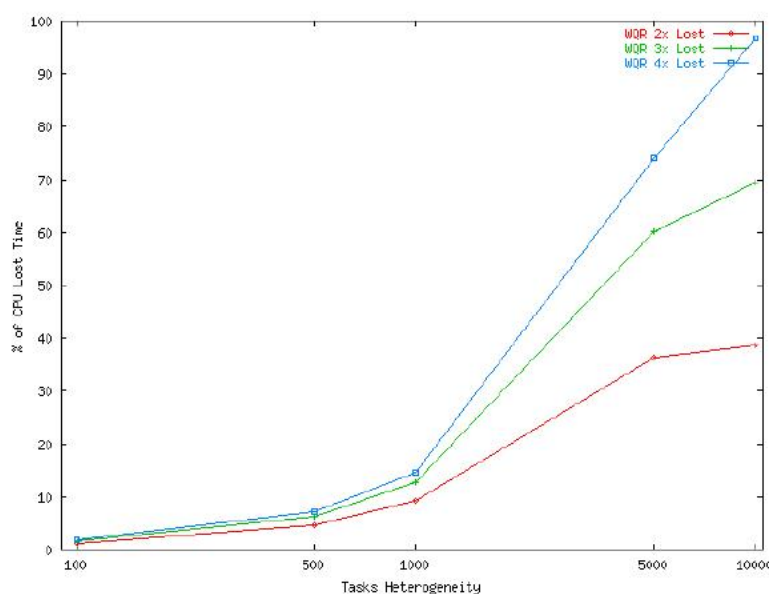


Figura 13.15: Desperdício de ciclos com a replicação

Pensando nisso, foi desenvolvida a heurística Storage Affinity [319], que explora padrões de reutilização de dados e aplica replicação de forma semelhante a WQR.

Os padrões de reutilização foram classificados em dois tipos básicos, *inter-job* e *inter-task*. O padrão *inter-job* determina que há reutilização de dados entre execuções sucessivas das aplicações. Vale salientar que isso é bastante comum em aplicações do tipo *Parameter Sweep* [113]. Outra situação de reutilização é capturada pela definição do padrão *inter-task*, aplicações que apresentam esse padrão possuem reutilização de dados durante uma execução. Por exemplo, aplicações de busca de seqüências de DNA, como o Blast [54], podem apresentar esse padrão, considerando que várias seqüências podem ser pesquisadas em paralelo usando o mesmo banco de dados de seqüências catalogadas.

Portanto, a idéia é explorar ambos os padrões de reutilização de dados para melhorar o desempenho das aplicações. Assim, Storage Affinity efetua o escalonamento baseado *afinidade* que as tarefas têm com os sites que formam o Grid. O valor da *afinidade* determina *quão próximo* do site esta tarefa está. A semântica do termo *próximo* está associada à quantidade de bytes da entrada da tarefa que já está armazenada remotamente em um dado site, ou seja, quanto mais bytes da entrada da tarefa já estiver armazenado no site, mais próximo a tarefa estará do site, pois poderá iniciar sua execução mais rapidamente. Assim, o valor da *afinidade* de uma tarefa com um site é o número de bytes pertencentes à entrada da tarefa,

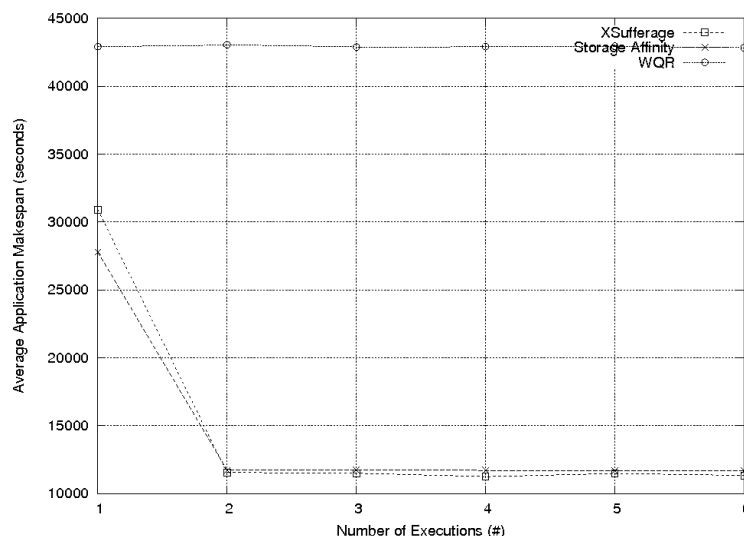


Figura 13.16: Sumário do desempenho de Storage Affinity comparado com outras heurísticas

que já estão armazenados no site.

Note que *Storage Affinity* utiliza tão somente as informações sobre o tamanho e a localização dos arquivos de entrada. É importante dizer que tais informações podem estar disponíveis no instante do escalonamento sem dificuldade e perda de precisão. Por exemplo, as informações relacionadas aos dados de entrada de uma tarefa podem ser obtidas através de requisições aos recursos de armazenamento de dados. Mesmo que estes recursos não sejam capazes de responder às requisições sobre *quais elementos* de dados eles armazenam e *qual o tamanho* de cada elemento de dado, uma implementação alternativa da heurística *Storage Affinity* poderia facilmente manter um histórico das informações sobre as transferências efetuadas para cada tarefa e assim possuir tais informações.

Além de aproveitar a reutilização dos dados, *Storage Affinity* também trata a dificuldade na obtenção de informações dinâmicas sobre o Grid, bem como informações sobre o tempo de execução das tarefas da aplicação. Para resolver este problema, *Storage Affinity* efetua *replicação de tarefas*.

Storage Affinity executa em duas fases. Na primeira fase, cada tarefa é associada a um processador. A ordem de escalonamento é determinada através do cálculo do valor da *afinidade* das tarefas. Após determinar as afinidades, a tarefa que apresentar o maior valor é escalonada em um processador do site com o qual a tarefa apresentou maior afinidade. A segunda fase consiste da replicação de tarefas.

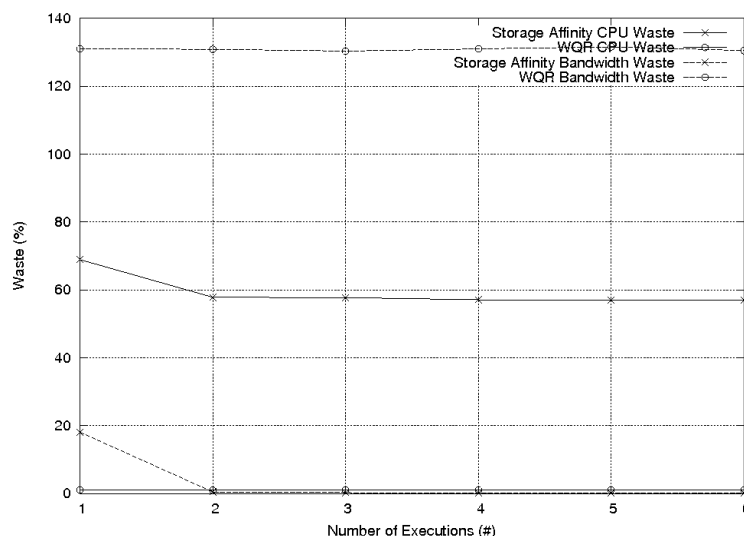


Figura 13.17: Sumário do desperdício de recursos por Storage Affinity comparado com outras heurísticas

Esta fase inicia quando não há mais tarefas aguardando para executar e pelo menos um processador está disponível. Uma réplica *pode* ser criada para qualquer tarefa em execução. Contudo, ao contrário de WQR, há um critério e uma ordem de prioridade para criação de réplicas. Considerando que o grau de replicação de uma tarefa é o número de réplicas criadas para esta tarefa, então ao iniciar a fase de replicação, os seguintes critérios são verificados na escolha de qual tarefa deve ser replicada: *i*) a tarefa deve estar executando e ter um valor de afinidade positivo, ou seja, alguma porção de sua entrada já deve estar presente no site de algum dos processadores disponíveis no momento; *ii*) o grau de replicação corrente da tarefa deve ser o menor entre as tarefas que atendem o critério (*i*); e *iii*) a tarefa deve apresentar o maior valor de afinidade entre as tarefas que atendem o critério (*ii*).

Quando uma tarefa completa sua execução as outras réplicas da tarefa são interrompidas. O algoritmo finaliza quando todas as tarefas que estão executando completam. Até isto ocorrer, a replicação de tarefas continua.

Na Figura 13.16 é apresentado uma comparação de desempenho entre três heurísticas: WQR, *XSufferage* e *Storage Affinity*. Esses resultados foram obtido através da investigação de mais de 3.000 cenários, onde vários aspectos foram considerados (i.e. heterogeneidade do Grid e da aplicação, tipo da aplicação e granularidade da aplicação) [319]. É possível ver que *Storage Affinity* consegue melhor desem-

penho que heurísticas que usam informação sobre o ambiente (XSufferage).

Um detalhe importante, mostrado na Figura 13.17 é a grande diferença de desperdício de recurso entre Storage Affinity e WQR. Esse efeito é produzido devido ao uso de estratégias de replicação diferentes pelas heurísticas. O fato de WQR não evitar transferências reduz o desperdício de CPU, por outro lado eleva bastante o desperdício de largura de banda da rede. Para Storage Affinity ocorre exatamente o contrário.

13.3.4 Imagem do Sistema

Ao usamos um computador, dependemos das abstrações criadas pelo sistema operacional, tais como arquivos, diretórios, permissões e processos, para lidarmos com o sistema. São tais abstrações que nos permitem expressar o queremos fazer. Elas também nos permitem nomear os dados persistentes que temos armazenados no sistema. Através destas abstrações básicas fornecidas pelo sistema operacional, o usuário tem uma imagem do sistema, formada pelo conjunto de objetos que ele pode manipular e pelas regras de manipulação destes objetos.

Plataformas de execução de aplicações paralelas que tem uma única instância do sistema operacional (SMPs) automaticamente fornecem a seus usuários uma imagem única do sistema. Já em plataformas que contém várias instâncias do sistema operacional (MPPs, NOWs e Grids), é necessário construir uma imagem consistente do sistema. Uma imagem consistente do sistema cria a ilusão (ainda que imperfeita) que os objetos que o usuário pode manipular são acessíveis da mesma forma de qualquer processador que compõe a plataforma.

MPPs e NOWs contam com boa conectividade e administração centralizada. Isso permite a configuração dos processadores que compõem a plataforma para compartilhar o mesmo cadastro de usuários e os sistemas de arquivo mais importante (o /home, por exemplo), criando assim uma imagem razoavelmente consistente do sistema. Grids, por outro lado, são amplamente dispersos e muitas vezes sob controle de diversas entidades administrativas distintas. Não é factível, por exemplo, simplesmente montar o mesmo /home em todos os processadores que compõem o Grid, pois, além de problemas de desempenho, há também questões administrativas. Por outro lado, não queremos deixar que o usuário tenha que

lidar com várias imagens totalmente distintas do sistema.

As soluções para este problema dividem-se em dois grandes grupos, aquelas que evitam os problemas administrativos trabalhando à nível de usuário [258, 368] e aquelas que introduzem novas abstrações para que o usuário possa lidar com o Grid [120, 119]. Em princípio, soluções à nível de usuário são mais simples de usar pois suportam abstrações já conhecidas pelo usuário (e.g. arquivos). Entretanto, elas podem apresentar sérios problemas de performance dependendo da aplicação e da conectividade do Grid. Novas abstrações, ao contrário, requerem o aprendizado de novos conceitos, mas podem vir a oferecer uma forma mais eficiente de usar o Grid.

13.3.5 Segurança

Um dos desafios impostos pela introdução de um serviço de execução remota (ver Seção 13.3.2) em um Grid é relacionado a segurança. Ou seja, os problemas de segurança podem afetar não apenas o proprietário do recurso, como também o usuário da aplicação. Ou seja, o recurso estará exposto ao permitir a execução de uma aplicação de um usuário de origem, a princípio, desconhecida. Por outro lado, o usuário não gostaria que sua aplicação fosse *sabotada* durante a execução gerando resultados não confiáveis.

Portanto, segurança é um tópico que deve se considerado para o desenvolvimento dos Grids Computacionais. Nesta seção iremos abordar algumas questões de segurança que surgem ao se pensar em uma infraestrutura de computação na escala de um Grid Computacional.

Proteção dos recursos

Uma vez que o Grid é formado por vários domínios administrativos distintos, naturalmente é possível que a execução de uma aplicação seja iniciada de uma domínio X e utilize recursos dos domínios Y e Z . Porém, como saber se a aplicação iniciada por um usuário do domínio X não contém código malicioso que podem prejudicar o funcionamento dos recursos utilizados?

Uma primeira, e óbvia, medida é implementar mecanismos de autenticação e autorização para permitir que apenas usuários do domínio X , que sejam autenticados pelos outros domínios, possam acessar os recursos autorizados por estes domínios. Ainda assim, não se pode garantir que a *credencial de acesso* do usuário não está sendo usada de forma incorreta, seja por corrupção (e.g. uso da máquina para disparar ataques de interrupção de serviço) ou acidentalmente (e.g. uma falha na aplicação pode criar uma vulnerabilidade) [29].

É justamente por não ter garantias a esse respeito que mecanismos de proteção para os recursos têm sido desenvolvidos. A idéia básica é criar um ambiente isolado, que no caso de ser comprometido não prejudique o funcionamento do recurso. Uma abordagem comum é a utilização de virtualização [241, 324, 81].

Uma das abordagens é implementada pelo Jail [241, 324], que surgiu para prover maior flexibilidade no gerenciamento de sistemas FreeBSD. A idéia é que o esquema de autorização clássico do Unix é pouco sofisticado para situações onde há uma demanda por uma granularidade mais fina nos níveis de permissões sobre o recursos. Essa estratégia funciona através do confinamento de um processo e seus descendentes em uma área (i.e. *jail*), nesta área o processo só pode acessar outros processos, sistemas de arquivo e recursos de rede que se encontram na mesma *partição*.

Uma partição é criada através da invocação à chamada de sistema *jail()*. Além disso, o escopo do sistema de arquivos visível pelo processo confinado é utilizando a chamada de sistema *chroot()*. Esta chamada foi modificada para corrigir vulnerabilidades que não permitiam a redução da visibilidade do processo com relação aos sistema de arquivos [241].

Note que não estamos tratando de partição necessariamente no que se refere ao sistema de arquivos. A partição (ou jail) na qual o processo deverá estar confinado é composta de um ambiente com sistema de arquivos, processos e rede isolados de outros processos.

Uma das vantagens do Jail é que um usuário pode interagir com o recurso, enquanto algum processo remoto executa em uma partição isolada do resto do sistema. Porém, isso pode causar inconvenientes para o usuário interativo, que na maioria dos casos não está disposto a contribuir com seu recurso para o Grid, caso isso implique em um consumo exagerado do recurso por parte do processo

estrangeiro. Sendo assim, outras alternativas fornecem mecanismos de *detecção de ociosidade* que prepara o ambiente para receber processos estrangeiros e executá-los em uma partição independente. Assim, espera-se que o usuário local não seja prejudicado por um processo que, por exemplo, consome muita memória, uma vez que o recurso estaria ocioso.

Outras soluções têm o objetivo de não apenas garantir que o sistema estará a salvo de qualquer tentativa de danificação, como de início de ataques a partir do recurso, como protegerá o usuário de inconvenientes causados por aplicações que utilizam muito da capacidade do recurso. Pensando nisso, o Swan é uma solução de *sandboxing* baseado na máquina virtual Xen [81]. O Swan é dividido em dois módulos: *i) Mode Switcher; ii) Virtual Machine*.

O primeiro módulo é responsável por monitorar o recurso, no intuito de detectar sua ociosidade. Ao detectar a ociosidade o recurso muda do sistema operacional no qual o usuário normalmente trabalha, para um sistema operacional modificado que garante o isolamento da aplicação remota do resto do sistema local. Isso inclui um sistema de arquivos completamente diferente (i.e. uma partição do disco diferente), a impossibilidade de enviar sinais para processos fora da máquina virtual e a restrição no acesso aos recursos de rede da máquina na qual está executando. A vantagem dessa abordagem é a exploração de ociosidade, porém há o overhead gerado pela reinicialização em outro sistema operacional.

No estado atual, não encontramos um padrão definido pelos comitês da área de Grid Computing, porém vários projetos apresentam soluções semelhantes para a proteção de recursos que formam o Grid.

Proteção da aplicação

Um outro lado da questão da segurança é a proteção da aplicação que executa no Grid. Ou seja, garantir que não haverá sabotagem na execução do serviço requisitado por um cliente. Por exemplo, suponha um serviço que fornece renderização de imagens. O serviço supostamente estaria disponível para responder a requisições de renderização de clientes espalhados por vários domínios administrativos. Porém, por algum motivo, esse serviço prioriza as requisições dos clientes locais e retorna sempre a mesma imagem para qualquer requisição de clientes exter-

nos. Com isso o provedor do serviço pretende economizar recursos que seriam destinados à computações estrangeiras.

Certamente, essa é uma situação simples de contornar, porém ainda assim pode causar prejuízos para o usuário da aplicação cliente que requisitou a renderização da imagem. Portanto, é necessário munir o cliente de mecanismos e estratégias de proteção contra este tipo de sabotagem.

Várias estratégias de tolerância à sabotagem já foram propostas para ambientes de computação voluntária, onde essa prática parece ter um maior impacto, já que os voluntários, em geral, não são confiáveis [322]. Um caso clássico foi o do Seti@home, onde o que motivava a sabotagem era apenas o aspecto *fama* [59]. Voluntários que mais fornecessem o serviço de execução para estas infraestruturas figuravam em um ranking. Assim, através de uma modificação no serviço de execução, se tornava possível *enganar* o sistema retornando um resultado inválido que era contabilizado como trabalho útil e melhorava a posição daquele voluntário no ranking.

Assim, uma estratégia simples é usar o que se chama de *majority voting* [322], que replica a execução de uma unidade de trabalho entre vários voluntários do sistema e espera até que um número m de resultados finais coincidentes sejam retornados. Porém, esse esquema tem suas limitações. Por exemplo, suponha um ambiente com um número grande de voluntários que retornam resultados inválidos. A replicação crescerá bastante em função deste número para tolerar essa quantidade de falhas. Isso, portanto, acarretará um nível alto de desperdício de recursos.

Apesar da estratégia *majority voting* ter a vantagem de ser bastante simples de implementar, ela não tolera situações onde o número de resultados inválidos é muito alto. Desta forma, outras abordagens são propostas. Uma forma de contornar a limitação do *majority voting* é usar a política denominada *spot-checking* [322]. Nesse esquema, os voluntários são verificados de forma aleatória através da solicitação de execução de um *benchmark*. A intenção é verificar se é possível confiar nos resultados gerados por este voluntário. Caso o *benchmark* detecte alguma falha na computação efetuada, ou seja, os resultados retornados pelo voluntário não conferem com o resultado esperado do teste, os resultados anteriores retornados por este voluntário são descartados e colocados na lista de trabalho pendente.

Uma vez que *spot-checking* é baseada na verificação aleatória da confiabilidade dos voluntários. Assim, ao detectar que um voluntário não é confiável, todos os resultados anteriores deste voluntário são descartados e o trabalho é reenviado a outros voluntários. Nesse estratégia, há uma menor repetição de trabalho, quando comparamos à estratégia *majority voting*. Existem duas variações da estratégia *spot-checking*: i) *spot-checking with blacklist*; ii) *spot-checking without blacklist*. A diferença entre as duas é o uso de uma lista com os voluntários maliciosos. Essa lista indica os voluntários que não devem ser mais considerados. Para uma maior discussão sobre as diferenças e implicações de cada abordagem sugerimos ao leitor o trabalho de Sarmenta et al. [322].

Devido ao fato de nem sempre ser possível manter uma lista de voluntários maliciosos de forma eficiente. Por exemplo, usar o IP para identificar unicamente os voluntários maliciosos pode não ser uma boa idéia, pois é bastante comum o fato dos hosts obterem IP dinâmicos todas as vezes que se conectam. Sendo assim, para resolver essa limitação surge uma nova abordagem baseada na definição de *credibilidade* [322]. A idéia é marcar vários objetos do sistema com um valor que descreve sua credibilidade. Então, é possível que voluntários, resultados e grupos de resultados tenham valores de credibilidade dependendo de vários fatores. Por exemplo, novos voluntários que entram no sistema têm menos credibilidade que antigos voluntários, onde seus resultados passaram com sucesso por vários *spot-checking*. Naturalmente, a credibilidade dos resultados do voluntário terá bastante relação com sua própria credibilidade, que pode evoluir ao passo que sua computação vai sendo verificada. Note que uma combinação de *majority voting* e *spot-checking* é uma alternativa possível para determinação da credibilidade dos voluntários.

13.4 Estudos de Caso

13.4.1 Globus

Globus consiste de um conjunto de serviços que facilitam a construção de infraestruturas para Computação em Grid [181]. Os serviços Globus podem ser usados para submissão e controle de aplicações, descoberta de recursos, movimentação de dados e segurança no Grid.

Apesar desses serviços fornecerem uma parte importante para a construção de Grids Computacionais, existem outros serviços além desse núcleo. Estes serviços, igualmente importantes, podem ser construídos sobre essas camadas definidas como a base de serviços para a infraestrutura. Discutiremos nas seções seguintes os aspectos mais importantes dessa infraestrutura de serviços.

Autenticação

Um aspecto que complica o uso de Grids na prática é a autenticação de usuários em diferentes domínios administrativos. Em princípio, o usuário tem que ser autenticado para cada domínio administrativo de uma forma determinada pelo administrador do domínio (que tipicamente envolve fornecer uma identificação de usuário e uma senha). Este esquema coloca uma grande carga no usuário (quem usa vários sites Web que exigem login, tem uma idéia bem concreta destes problemas).

No contexto de Computação em Grid, os problemas causados pela múltipla autenticação são agravados, pois queremos serviços que possam efetuar ações automaticamente, porém essas ações exigem autenticação (e.g. submeter uma tarefa para execução em um site remoto, solicitar o armazenamento ou acesso a um determinado arquivo). Além disso, como o objetivo do Globus é permitir a criação de organizações virtuais, através da agregação de recursos e serviços distribuídos por vários domínios administrativos diferentes, certamente questões relacionadas a delegação de credencial estão envolvidas no processo de autenticação.

GSI (Globus Security Infrastructure) é o serviço Globus que ataca estes problemas. GSI viabiliza o login único no Grid. GSI utiliza criptografia de chave pública, certificados X.509 e comunicação SSL (Secure Sockets Layer) para estabelecer a identidade Globus do usuário. Por exemplo, `C=US,O=University of California San Diego, OU=Grid Computing Lab, CN=Walfredo Cirne` era uma identidade em Gusto (o primeiro Grid montado com Globus). Depois do usuário ter se identificado junto ao GSI, todos os demais serviços Globus saberão, de forma segura, que o usuário é de fato quem diz ser. Uma vez que um serviço sabe a identidade Globus do usuário, resta estabelecer quais operações tal usuário pode realizar. Isto é feito mapeando a identidade Globus para um usuário local. Por exemplo, o serviço GRAM (veja

Seção 13.4.1) instalado em *thing1.ucsd.edu* mapeava C=US, O=University of California San Diego, OU=Grid Computing Lab, CN=Walfredo Cirne para walfredo. Já o GRAM que executava em *bluehorizon.sdsc.edu* mapeava C=US, O=University of California San Diego, OU=Grid Computing Lab, CN=Walfredo Cirne para ul5595.

Com a introdução da especificação OGSI, a partir do Globus 3.0, novas questões de segurança tiveram que ser abordadas, principalmente pela convergência com Web Services. Ainda assim, vários padrões e especificações que definem formatos para descrição de políticas de segurança, formatos para delegação de credencial e autenticação e estabelecimento de comunicação segura, puderam ser aproveitados no intuito de prover uma infraestrutura de segurança para computação em Grids baseada em componentes interoperáveis [381].

O objetivo principal do conjunto de serviços de segurança Globus, ilustrado na Figura 13.21 como a camada *GT3 Security Services*, é prover transparência para os serviços das camadas de mais alto nível com relação à infraestrutura de segurança do Grid.

Descoberta e Alocação de Recursos

Como discutimos na Seção 13.3.3, Grids não têm um escalonador que controla todo o sistema. Assim sendo, quando um usuário submete uma aplicação para execução no Grid, o usuário utiliza um escalonador de aplicação que escolhe os recursos a utilizar, particiona o trabalho entre tais recursos, e envia tarefas para os escalonadores dos recursos, como ilustrado pela Figura 13.11.

GRAM (*Globus Resource Allocation Manager*) é o serviço da arquitetura Globus que fornece uma interface uniforme para submissão e controle de tarefas [133], escondendo a multiplicidade de escalonadores de recursos dos demais serviços de Grid (do escalonador de aplicações, por exemplo). Além disso, GRAM provê informações sobre o status do recurso ao MDS (o serviço Globus que fornece informação sobre o Grid).

A Figura 13.18 permite um exame da arquitetura do GRAM, que esclarece bastante sobre seus objetivos e funcionamento, através da identificação dos três compo-

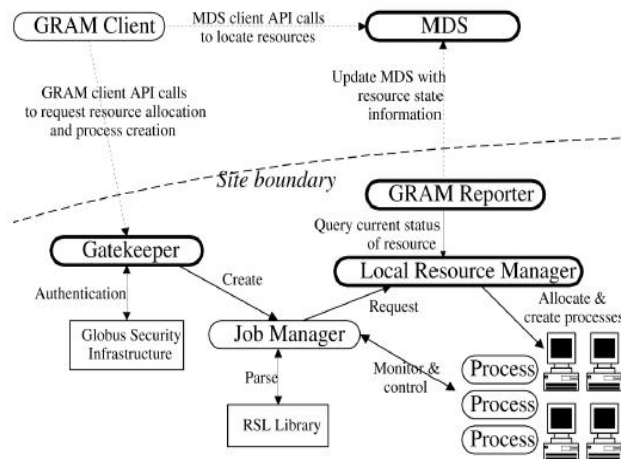


Figura 13.18: Arquitetura do GRAM [133]

nentes do GRAM (*Gatekeeper*, *Job Manager* e *GRAM Reporter*), bem como componentes externos que interagem com o GRAM. O cliente GRAM é aquele que o utiliza para submeter e controlar a execução de tarefas. Note que o cliente GRAM pode ser um escalonador de aplicação ou até o próprio usuário. Para o cliente, a grande vantagem de usar GRAM é a manipulação uniforme de tarefas, i.e. a submissão e controle de tarefas não importando qual é o escalonador de recurso (*Local Resource Manager*, na Figura 13.18) usado para controlar a máquina. Isto é possível porque as requisições enviadas ao GRAM são sempre escritas em RSL (*Resource Specification Language*), independentemente de qual escalonador de recurso esteja sendo utilizado. O *Job Manager* é o responsável por converter a requisição em RSL em um formato que o escalonador de recurso em questão entenda. Ao que sabemos, há versões do *Job Manager* para Condor [258], LoadLeveler [242], PBS, Unix e Windows, entre outras plataformas.

Uma idéia bastante interessante em Globus é que escalonadores de aplicação podem usar os serviços de outros escalonadores de aplicação. O escalonador que recebe a solicitação do cliente lida com a especificação em mais alto nível. Ele refina tal especificação e, para implementá-la, submete novas solicitações a escalonadores de recurso (que de fato executam solicitações) e/ou escalonadores de aplicação (que utilizam outros escalonadores para executar solicitações).

Globus suporta bem esta hierarquia de escalonadores através da linguagem RSL. RSL é capaz de expressar tanto solicitação de alto nível (como a que o usuário envia a seu escalonador de aplicações), como também solicitações concretas (que

são enviadas para GRAMs, que as traduzem para escalonadores de recurso locais). Portanto, o trabalho de um escalonador de aplicação em Globus pode ser descrito como sendo o de refinar solicitações RSL. A Figura 13.19 ilustra este processo no Globus 2.0. Note que Globus usa o termo *broker* para o que chamamos de escalonador de aplicação.

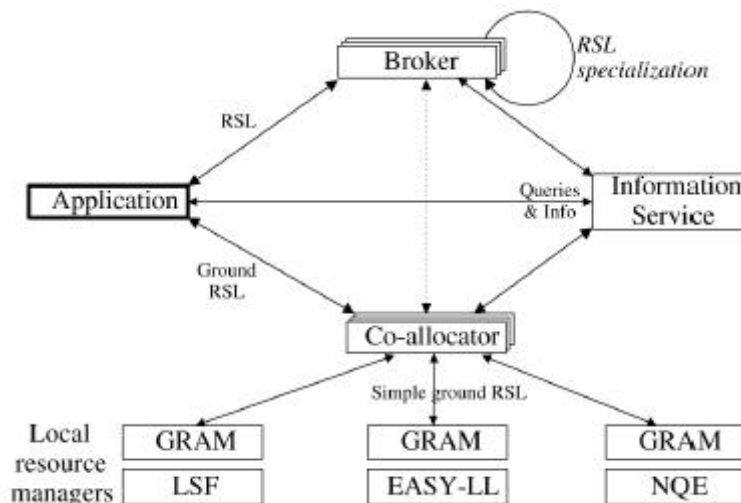


Figura 13.19: Delegação entre escalonadores de aplicação [133]

Um componente importante para a execução de aplicações fortemente acopladas é o co-alocador (*Co-allocator*). O co-alocador é um escalonador de aplicação especializado em garantir que tarefas localizadas em máquinas distintas executem simultaneamente. Em aplicações fortemente acopladas, as tarefas precisam se comunicar para que a aplicação faça progresso. Portanto, todas as tarefas da aplicação têm que ser executadas simultaneamente. É importante ressaltar que uma boa implementação de co-alocação depende da implementação, por parte dos escalonadores de recurso, do serviço de reservas prévias (*advance reservation*). Reservas prévias permitem a escalonadores de aplicação obter garantias de escalonadores de recurso que determinados recursos (e.g. processadores) estarão disponíveis para aplicação em um intervalo de tempo preestabelecido [338].

A Figura 13.20 apresenta um exemplo da submissão de uma aplicação em um Grid Globus. Veja que um usuário envia uma solicitação de executar uma simulação interativa envolvendo 100.000 entidades para um escalonador de aplicação especializado em simulação interativa distribuída. Tal escalonador converte a so-

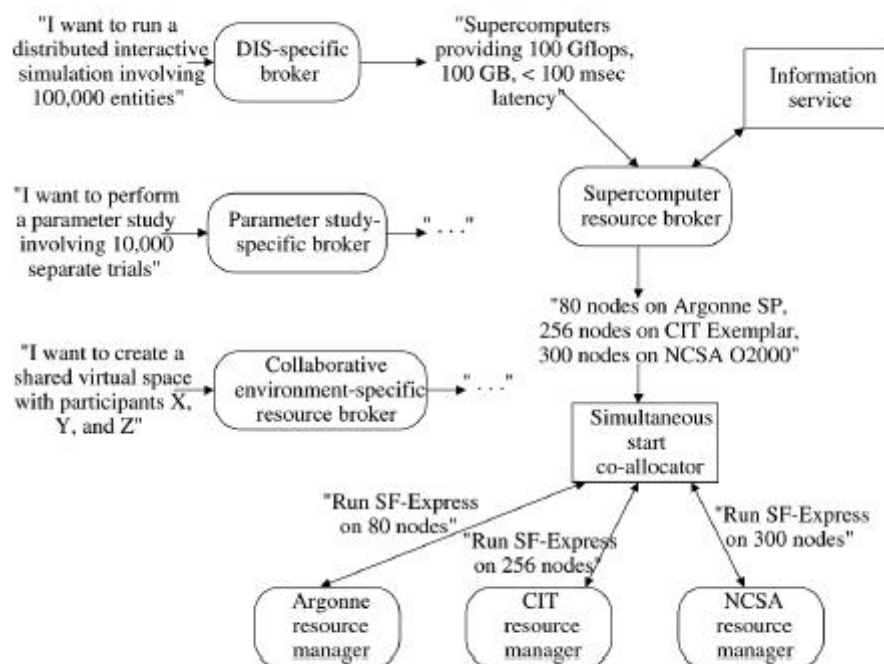


Figura 13.20: Exemplo do uso de escalonadores no Globus [133]

licitação original em outra mais específica, que descreve a necessidade do usuário em termos de ciclos, memória e latência de comunicação. Esta nova solicitação é então enviada a um escalonador de aplicação especializado em MPPs. Este escalonador consulta o MDS para descobrir quais MPPs (dentro daqueles aos quais o usuário tem acesso) são os melhores para utilizar no momento. Além disso, o escalonador especializado em MPPs faz a partição do trabalho entre os MPPs escolhidos e envia a solicitação mais refinada para o co-alocador. O co-alocador garante que as tarefas submetidas aos distintos MPPs comecem a executar simultaneamente. Note também que outros escalonadores de aplicações podem participar do sistema. A Figura 13.20, por exemplo, exemplifica ainda escalonadores para varredura de parâmetros e para ambientes de colaboração virtual.

Comunicação

O problema de comunicação no Grid pode ser visto como uma instância do eterno conflito entre generalidade e performance. Caso utilizemos um mecanismo de comunicação genérico (e.g. TCP) que viabilize a comunicação fim-a-fim entre quaisquer duas tarefas no Grid, perdemos performance em casos especiais (e.g.

se ambas tarefas rodam em uma máquina de memória compartilhada, elas poderiam se comunicar muito mais rapidamente pela memória). Por outro lado, gostaríamos de usar um mecanismo genérico para não ter que programar para cada uma das várias tecnologias de comunicação existentes.

Globus ataca este problema com o Nexus [185]. Nexus fornece uma interface de baixo nível, mas uma implementação adaptável que escolhe, dentre as tecnologias de comunicação disponíveis, a que vai oferecer melhor performance. Por exemplo, se ambas tarefas estão em uma máquina de memória compartilhada, Nexus utilizará a memória para efetuar a comunicação. Caso as tarefas estejam em um MPP, Nexus utilizará o switch de alta velocidade para comunicação. Caso as tarefas estejam em máquinas geograficamente distantes, Nexus utilizará TCP/IP.

Nexus fornece uma interface de relativo baixo nível: invocação remota de procedimento, mas sem retorno de resultado. Portanto, programar diretamente em Nexus não é das tarefas mais agradáveis. Entretanto, a idéia da equipe Globus é que Nexus seja usado por desenvolvedores de ferramentas e mecanismos de comunicação, não diretamente pelo desenvolvedor de aplicações. MPI-G é o exemplo perfeito desta abordagem. MPI-G implementa o popular padrão MPI (Message Passing Interface) sobre Nexus. Assim, o desenvolvedor de aplicações escrever em MPI e link-editar sua aplicação com MPI-G para automaticamente ter acesso a melhor tecnologia de comunicação disponível (selecionada pelo Nexus).

Transferência de Dados

A necessidade de acesso remoto e transferência de dados é uma constante na Computação em Grid. Na verdade, várias das aplicações aptas a executar no Grid necessitam de paralelismo exatamente porque processam enormes quantidades de dados. Ciente deste fato, Globus logo disponibilizou GASS (Global Access to Secondary Storage) [92], um serviço para acesso remoto a arquivos sob a tutela de um servidor GASS. O cliente GASS é uma biblioteca C que é link-editada à aplicação usuária do serviço. Com o intuito de fornecer boa performance, o serviço GASS implementa as otimizações típicas de acesso remoto como caching e pre-fetching.

Apesar de ser um bom serviço, o GASS encontrou problemas de implantação. A dificuldade encontrada foi de interoperabilidade. A maioria das fontes de dados onde se instalaria um servidor GASS já executa algum serviço de transferência e/ou acesso remoto a arquivos. Os administradores de sistema se questionavam então porque não se poderia usar os serviços existentes. Essa realidade motivou a introdução do GridFTP [51] por parte da equipe Globus. GridFTP estende o popular protocolo FTP para torná-lo mais adequado para as necessidades da Computação em Grid. Mais precisamente, GridFTP introduz suporte a:

- Autenticação GSI e Kerberos
- Transferência em paralelo (várias conexões TCP entre fonte e destino)
- Transferência striped (conexões TCP entre várias fontes e um destino, ou vice-versa)
- Controle manual dos buffers TCP (usado para afinamento de performance)
- Instrumentação embutida

Uma vez que GridFTP é uma extensão do FTP, o problema de interoperabilidade fica resolvido, pois FTP é amplamente suportado pelos servidores de dados. Obviamente, se as extensões GridFTP não estiverem implementadas em um dado servidor, os benefícios adicionais do protocolo não estarão disponíveis. Mas o cliente GridFTP ainda será capaz de obter os dados desejados. Ou seja, o sistema será penalizado apenas no desempenho, porém continuará funcionando.

Avaliação do Globus

Um aspecto importante para grande aceitação do Globus é que os serviços oferecidos são razoavelmente independentes, possibilitando que se utilize apenas parte dos serviços Globus em uma dada solução. Essa possibilidade do uso parcial de Globus ajuda sobremaneira na adaptação de aplicações paralelas existentes para o Grid. Pode-se começar usando serviços mais básicos e ir, aos poucos, incorporando funcionalidades mais avançadas. O design oposto à abordagem conjunto de serviços independentes do Globus é exemplificado pelo Legion [204]. Legion fornece um modelo orientado a objetos poderoso e flexível. Entretanto, o

usuário tem que utilizar a solução Legion de forma integral, sem estágios intermediários. Esta diferença de abordagem talvez tenha contribuído para prevalência do Globus como padrão *de facto* de infraestrutura para Computação em Grid. É interessante notar que a decisão de estruturar Globus como um conjunto de serviços independentes deixa claro que Globus não é uma solução pronta e completa (*plug-and-play*) para construção de Grids. Globus certamente fornece serviços úteis para Computação em Grids. Mas, desenvolvedores, administradores e usuários precisam despendar certo esforço para finalizar seu Grid. Por exemplo, administradores precisam decidir quais usuários terão acesso a quais recursos que compõem o Grid e em quais condições este acesso se dará (veja Seção 13.4.1). Em outro exemplo, freqüentemente é necessário desenvolver escalonadores de aplicação (veja Seção 13.3.3) que tenham conhecimento sobre as aplicações que serão executadas e algumas vezes também sobre a estrutura do Grid a ser usado.

Computação em Grid é simplesmente muito complexa para possibilitar soluções *plug-and-play*. Portanto, o fato do Globus não ser uma solução pronta e completa não é nenhum demérito. Entretanto, algumas pessoas têm a idéia de que Globus é a solução, completa e perfeita. Esta falsa concepção, sim, é um problema pois gera falsas expectativas e obscurece discussões técnicas com alegações de marketing.

Vale ressaltar que a discussão apresentada nas seções anteriores é válida para o Globus 2.0. Ainda se torna pertinente apresentar as características do Globus 2.0, pois muitos grupos interessados em computação de alto desempenho utilizam infraestruturas baseadas no GT2 (Globus Toolkit 2.0).

A introdução do padrão OGSA criou um alinhamento com tecnologias e padrões Web Services, assim vários desses aspectos discutidos anteriormente se modificaram em busca da implementações de padrões que promovem maior interoperabilidade.

A arquitetura do Globus Toolkit 3 é ilustrada pela Figura 13.21. Essa versão é uma implementação da especificação OGSF (*Open Grid Services Infrastructure*) [366], os serviços implementados na camada *GT3 Core Services* representam os serviços especificados pela OGSF. O objetivo é especificar mecanismos para criação, gerenciamento e troca de dados entre Grid Services.

Como discutimos nas Seções 13.2.7 e 13.2.7, há uma tendência forte de convergência entre as tecnologias de Grids Computacionais e Web Services. Isso fica

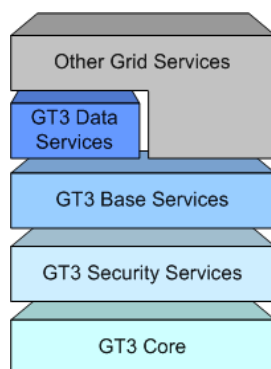


Figura 13.21: Arquitetura do Globus [343]

claro com a introdução da especificação WSRF, que posiciona as tecnologias de Grids junto aos padrões para Web Services.

13.4.2 MyGrid

A motivação para a construção do MyGrid surgiu do fato que, embora bastante pesquisa tenha sido realizada para o desenvolvimento dos Grids Computacionais, poucos usuários executavam suas aplicações paralelas sobre essa infraestrutura. Assim, foi concebido projeto MyGrid, com o intuito de alterar esta situação. Para tanto, foram atacadas apenas aplicações Bag-of-Tasks, ou seja, aquelas aplicações cujas tarefas são independentes, podendo ser executadas em qualquer ordem. Aplicações Bag-of-Tasks são um alvo interessante porque (i) se adequam melhor a ampla distribuição, heterogeneidade e dinamicidade do Grid, e (ii) resolvem vários problemas importantes, tais como mineração de dados, processamento genômico, pesquisa massiva (como quebra de chaves criptográficas), varredura de parâmetros, simulações Monte Carlo (muito utilizado, por exemplo, pela indústria farmacêutica [215]), computação de fractais (como Mandelbrot), e manipulação de imagem (como tomografia).

Estabelecido o escopo do MyGrid, nosso objetivo é construir um sistema simples, completo e seguro. Por simples queremos dizer que o esforço para utilização do MyGrid deve ser mínimo. Em particular, queremos chegar o mais próximo possível de uma solução pronta (plug-and-play). Por completo denotamos a necessidade de cobrir todo o ciclo de uso de um sistema computacional, do desenvolvimento à execução, passando por instalação e atualização e incluindo também

a manipulação de arquivos. Por seguro expressamos a necessidade de não introduzir vulnerabilidades ao ambiente computacional do usuário. Ou seja, não queremos que falhas de segurança em qualquer uma das máquinas que o usuário possa utilizar sejam propagadas para sua máquina base (i.e. o computador usado pelo usuário).

MyGrid diferencia entre máquina base e máquina do Grid. Em um MyGrid, a máquina base é aquela que controla a execução da aplicação. Ela tipicamente contém os dados de entrada e coleta os resultados da computação. A máquina base é normalmente usada pelo usuário diretamente no seu dia-a-dia, muitas vezes sendo o próprio computador desktop do usuário. Esperamos, portanto, que o usuário tenha excelente acesso à máquina base e que tenha customizado um ambiente de trabalho confortável nela.

Todas as máquinas usadas via MyGrid para executar tarefas são chamadas de máquinas de grid. Ao contrário da máquina base, não assumimos que o usuário customizou cada máquina do Grid para criar-lhe um ambiente de trabalho familiar. Além disso, todas as máquinas do Grid tipicamente não compartilham um mesmo sistema de arquivo ou têm os mesmos softwares instalados. A imagem do sistema pode variar de uma máquina do Grid para outra. Portanto, para mantermos a simplicidade de uso do sistema, precisamos evitar que o usuário tenha que lidar diretamente com as máquinas do Grid. Por exemplo, queremos evitar que o usuário tenha que instalar software em cada máquina de seu Grid. A idéia é que máquinas do Grid sejam manipuladas através das abstrações criadas por MyGrid (descritas a seguir).

Um aspecto importante de MyGrid é dar ao usuário a possibilidade de usar quaisquer recursos que ele tenha acesso. Este não é um objetivo trivial porque ele implica que temos que assumir muito pouco a respeito de uma máquina do Grid, de forma a não impedir algum usuário de usar uma máquina que não suporta nossas hipóteses. Em particular, não podemos assumir que tal recurso tenha software MyGrid instalado. MyGrid define *Grid Machine Interface* como sendo o conjunto mínimo de serviços que precisam estar disponíveis para que uma dada máquina possa ser adicionada ao Grid do usuário. Tais serviços são:

Como ilustrado pela Figura 13.22, há várias formas de implementar os serviços oferecidos através da *Grid Machine Interface*. Uma forma é fornecer ao sistema scripts que implementam os serviços listados na Tabela 13.2. Neste caso, MyGrid

Serviços
Execução remota
Transferência de Arquivo (Máquina Base $\leftrightarrow$ Grid Machine)
Transferência de Arquivo (Grid Machine $\leftrightarrow$ Máquina Base)
Interrupção de uma Execução múltipla

Tabela 13.2: Grid Machine Interface

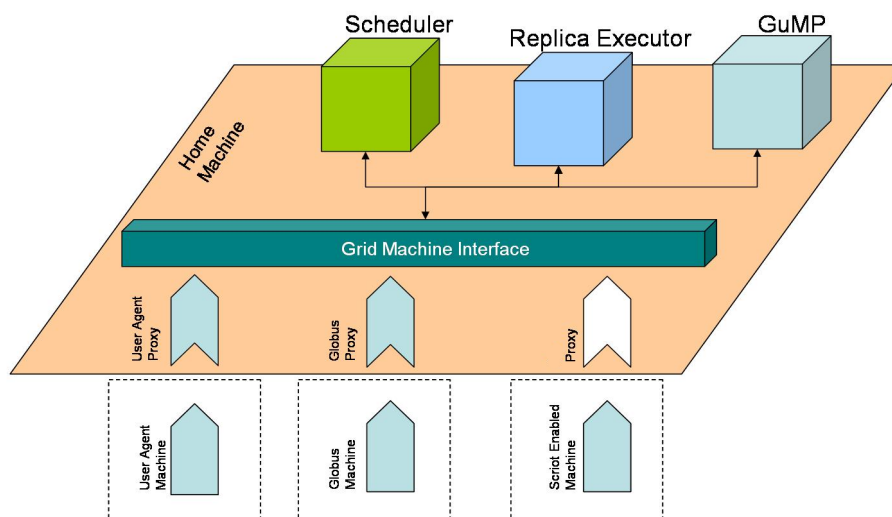


Figura 13.22: Arquitetura do MyGrid

utiliza o módulo Grid Script para acessar a máquina em questão. Note que Grid Script possibilita que, qualquer que seja a máquina que o usuário tem acesso, ele possa informar como este acesso se dá através da escrita de três scripts. Alternativamente, há casos em que a forma de acessar uma determinada máquina do Grid já é do conhecimento do MyGrid. Por exemplo, suponha que a máquina em questão pode ser acessada via serviços Globus (GSI, GRAM e GridFTP). Neste caso, o usuário não precisa fornecer os scripts, indicando apenas que o acesso à máquina já é conhecido de MyGrid. Finalmente, MyGrid também provê um mecanismo de acesso a máquinas do Grid, chamado de User Agent. O User Agent provê serviços simples. É interessante notar que, pela terminologia adotada por Foster, et al. [184], *Grid Machine Interface* é uma virtualização para os serviços de acesso a uma máquina do Grid.

Outro componente fundamental a arquitetura MyGrid é o *Scheduler*. O Scheduler recebe do usuário a descrição das tarefas a executar, escolhe qual processador usar para cada tarefa, e finalmente submete e monitora a execução da tarefa. O

MyGrid possui atualmente duas heurística de escalonamento: Workqueue with Replication (WQR) [297] e Storage Affinity [319]. Ambas, conseguem obter uma boa performance mesmo sem utilizar informações sobre o estado do Grid ou o tamanho de cada tarefa (ver Seção 13.3.3 e Seção 13.3.3). O WQR foi definido para aplicações *cpu-intensive*, enquanto Storage Affinity foi desenvolvido para melhorar o desempenho de aplicações que processam grandes quantidades de dados.

Note que ter um algoritmo de escalonamento que funciona bem sem depender de muita informação é importante, pois simplifica a definição da *Grid Machine Interface*. Caso o algoritmo de escalonamento do MyGrid necessitasse de informações sobre as máquinas do Grid, *Grid Machine Interface* teria que ser mais rica e, portanto, mais difícil de virtualizar. Por exemplo, o Nimrod-G [102] define uma interface de abstração para os recursos, que contempla métodos de fornecimento de informação sobre o recurso. Certamente, a informação obtida por esses métodos é valiosa para o escalonamento eficiente das aplicações. Porém, isso limita o tipo de recurso que pode ser utilizado, pois torna o *middleware* dependente de um recurso que forneça uma implementação para os métodos dessa interface.

Uma aplicação (ou Job) MyGrid é composta de tarefas independentes. Estas tarefas são compostas por três partes ou sub-tarefas: *init*, *remote* e *final*. As sub-tarefas são executadas sequencialmente ($init \mapsto remote \mapsto final$). As sub-tarefas *init* e *final* são usadas para efetuar as transferências de arquivo de entrada e saída da tarefa respectivamente. Sendo assim, tanto a sub-tarefa inicial quando a final são executadas na máquina base. Enquanto, a sub-tarefa *remote*, como o próprio nome sugere, é executada em uma máquina do Grid e realiza a computação propriamente dita da tarefa.

Para definir as sub-tarefas inicial e final, o usuário tem disponível algumas abstrações providas pelo MyGrid para lidar com a máquina do Grid sem necessitar de conhecer sua imagem do sistema. As abstrações providas pelo MyGrid são *storage* e *playpen*. O serviço de *storage* possui a semântica de uma área de armazenamento persistente à execução da tarefa. Ou seja, é útil usar *storage* para distribuição de arquivos que provavelmente serão reutilizados no Grid (ex.: executáveis da aplicação). Assim sendo, qualquer arquivo transferido para o *storage* é automaticamente incluído no PATH da máquina do Grid. Por outro lado, o *playpen* provê uma área de armazenamento temporária de maneira independente

das convenções locais do sistema de arquivo de uma dada máquina do Grid. Essa área de armazenamento temporária é criada automaticamente e é o diretório base de execução da sub-tarefa *remote*. Note que o *playpen* foi concebido para possibilitar o armazenamento de dados temporários e resultados de tarefas. Também no sentido de simplificar a escrita das sub-tarefas, as variáveis de ambiente \$STORAGE, \$PLAYPEN, \$PROC e \$TASK são automaticamente definidas pelo MyGrid e contêm, respectivamente, o caminho completo para área de *storage*, o caminho completo para o *playpen* de uma dada tarefa, o nome da máquina do Grid escolhida para executar a tarefa e um identificador da tarefa.

13.4.3 OurGrid

Ao contrário do Globus, a solução OurGrid tem um escopo diferente, porém complementar. O objetivo é prover uma solução efetiva para a execução de aplicações Bag-of-Tasks em Grids Computacionais. Sendo assim, as decisões de projeto estão centradas no uso da solução em ambientes de produção. Portanto, a idéia básica é abdicar da generalidade, em alguns casos, no intuito de se obter uma solução, apesar de simples, eficiente e que possa ser facilmente usada em produção.

A arquitetura do OurGrid, brevemente comentada na Seção 13.2.6 e ilustrada na Figura 13.5, é formada por três componentes: MyGrid Broker (ver Seção 13.4.2), OurGrid Peer e uma solução de sandboxing baseada na máquina virtual Xen [81]. Nas seções seguintes descreveremos como os componentes do OurGrid abordam alguns aspectos importantes da Computação em Grid.

Autenticação

Na arquitetura OurGrid existem basicamente dois níveis de autenticação. Esses níveis dependem de como o usuário obteve o recurso. Primeiramente, o usuário pode ter acesso direto a alguns recursos (i.e. *Grid Machines* - GUMs - em sua rede local), neste caso o usuário usa o esquema de autenticação tradicional, em geral, isso implica na utilização da infraestrutura de autenticação do sistema operacional do recurso, ou seja, nome de usuário (*login*) e uma senha.

Contudo, além das GUMs que o usuário tem acesso direto, OurGrid permite (e

promove) a obtenção de acesso a GUMs de outros sites, isso ocorre através de um OurGrid Peer local ao site do usuário. Este peer deve estar conectado à rede de favores (ver Figura 13.5). Assim, para as GUMs obtidas da comunidade, há uma autenticação em duas vias baseada em certificados digitais no formato X.509. A primeira parte da autenticação deve garantir que o usuário tem permissão para solicitar serviços às GUMs (i.e. que a GuM conhece o usuário que está requisitando seus serviços). A segunda parte deve garantir que o usuário não está solicitando serviços a uma falsa GUM. Ou seja, tanto o usuário, através do broker, quanto os peers da comunidade possuem uma lista de certificados que são usadas para validar a tentativa de acesso.

Isso impede que usuários não autorizados pelo peer, obtenham acesso aos serviços de descoberta de novas Grid Machines, transferência de arquivos, execução e gerenciamento do ciclo de vida de replicas fornecido pelas GUMs controladas por um dado peer.

Outro aspecto importante é que através da utilização de certificados, a comunicação entre o MyGrid Broker, o peer e as Grid Machines será segura, evitando que os dados sejam interceptados e manipulados durante a comunicação. A segurança na comunicação é fornecida através do uso de RMI baseado em SSL (*Secure Socket Layer*), que garante comunicação criptografada.

Descoberta e Alocação de Recursos

Para executar uma aplicação usando a OurGrid o usuário deve descrever sua aplicação e o conjunto de recursos que o usuário tem acesso. Note que esse conjunto de recursos pode ser apenas a indicação de um OurGrid Peer, que tem a função de obter recursos para o usuário.

A descrição da aplicação é basicamente: um conjunto tarefas, seus arquivos de entrada, arquivos de saída e seus requisitos (e.g. sistema operacional necessário, mínimo de memória, arquitetura do processador). Em seguida, o usuário o submete sua aplicação para execução no Grid através do MyGrid Broker. O componente interno do MyGrid Broker que recebe a submissão é o *Scheduler*. Por sua vez, o Scheduler requisita aos provedores de GUMs recursos para executar a aplicação submetida pelo usuário. Esses provedores podem responder com recursos

locais ou recursos obtidos na rede de favores. Para que o Scheduler receba uma resposta dos provedores é necessário que tenha sido encontrada uma GUM que preenche os requisitos determinados na descrição da aplicação.

Portanto, após ter sido descoberto um recurso que possui atributos compatíveis com os requisitos da aplicação, o recurso é alocado e repassado para o Scheduler que o solicitou. Certamente, isso só irá ocorrer caso o recurso esteja disponível. Porém, caso o recurso tenha sido descoberto através da rede de favores, o recurso pode ser tomado de volta (i.e. preemptado) pelo peer que o forneceu, seguindo a dinâmica da rede de favores. A preempção é um evento natural e previsto pela arquitetura do OurGrid, uma vez que os recursos só são cedidos caso esteja ocioso. Ou seja, uma solicitação local no site, ao qual o recurso pertence, pode ocasionar a preempção.

Vale também ressaltar que a alocação do recurso é feita no nível do MyGrid Broker, ou seja, isso não significa que o recurso estará dedicado exclusivamente ao MyGrid Broker. Portanto, não há impedimento para que outras aplicações que não usam a infraestrutura do OurGrid estejam executando concorrentemente com a aplicação submetida pelo usuário.

Comunicação

Uma vez que o foco da solução OurGrid está nas aplicações Bag-of-Tasks, não faz parte do escopo da solução OurGrid prover mecanismos de comunicação para aplicações fortemente acopladas. Mesmo assim, é possível usar a infraestrutura OurGrid para executar aplicações deste tipo, desde que a execução seja interna a um site. Por exemplo, uma aplicação que usa MPI, quando descrita pelo usuário, pode ter especificado em seus requisitos que necessita de uma GUM (Grid Machine), que na verdade é o *front end* de uma coleção de vários processadores (i.e. um cluster). Essa demanda será atendida se existir uma GuM, não alocada, que possua um atributo compatível com o requisito especificado pela aplicação. Portanto, apesar de não ter uma arquitetura que provê comunicação entre as tarefas que estão sendo executadas nas GuMs, a solução OurGrid provê meios de agregar ao Grid GuMs que permitem a execução de aplicações fortemente acopladas.

Transferência de Dados

A solução OurGrid para transferência de dados é baseada no tratamento de arquivos. Desta forma, o usuário ao descrever sua aplicação tem a sua disposição o uso de três operações de transferência arquivos, `put`, `store` e `get`, que podem ser usadas para preparar o ambiente para execução da aplicação (colocando os arquivos nos sites onde a aplicação irá executar), como também coletar os dados resultantes do processamento.

Tanto `put` quanto `store` são operações que permitem a transferir arquivos *para* a GuM. A diferença entre as duas operações consiste apenas do fato que `store` evita a transferência do arquivo caso o arquivo já se encontre armazenado no lado remoto. Isso é útil, por exemplo, para executáveis da aplicação e dados que são reutilizados entre execuções sucessivas da aplicação. A terceira operação, `get`, fornece um método de coletar arquivos resultantes da execução das aplicações.

A infraestrutura de comunicação usada para a transferência é a mesma usada para a solicitação de serviços de execução, ou seja, RMI. Uma vez que é possível ter segurança na comunicação com as GuMs de RMI sobre SSL, as operações de transferências de dados também gozam da segurança fornecida pela camada de comunicação baseada em certificados.

Avaliação do OurGrid

A característica mais importante do OurGrid é conseguir prover uma solução útil e eficiente para uma comunidade de usuários em produção, apesar de se basear em soluções simples e de escopo limitado (i.e. apenas aplicações do tipo Bag-of-Tasks).

É importante notar que o objetivo do OurGrid contrasta com o objetivo do Globus, que fornece um conjunto de serviços para a construção da infraestrutura do Grid. Portanto, OurGrid é uma solução que complementa o Globus provendo um broker (i.e. MyGrid) e abstrações que permitem o usuário usar recursos Globus e não-Globus. Por outro lado, Globus complementa o OurGrid ao fornecer a infraestrutura de serviços para execução de aplicações em larga escala.

OurGrid persegue um objetivo diferente do que seria prover uma solução genérica para computação em Grid. Com o foco em aplicações BoT foi possível produzir uma solução efetiva para uma comunidade de usuários em produção. Não se quer dizer com isso que não se pretende introduzir novas funcionalidades, que aumentem o escopo da solução. Ao contrário, a idéia é gradativamente permitir que mais aplicações possam utilizar a solução. Por exemplo, suporte a *workflow*, suporte a *data-mining*, etc.

Atualmente na versão 3.0.2, OurGrid já é usado em produção por vários grupos de pesquisa no Brasil [117]. As próximas versões prometem incorporar melhorias no escalonamento de aplicações, solução de sandboxing, bem como maior tolerância para aplicações de longa duração (*high-throughput computing*) através do uso de *checkpoint*. O software OurGrid está disponível para download em <http://www.ourgrid.org>.

13.4.4 Condor

De forma semelhante ao OurGrid, Condor é uma sistema que possui um escopo bem definido e menos genérico que outras soluções para computação de alto desempenho em Grids Computacionais. Condor objetiva fornecer grande quantidade de poder computacional a médio e longo prazo (dias a semanas) utilizando recursos ociosos na rede [258]. Os autores do sistema salientam insistentemente que Condor objetiva alta vazão (*high throughput*) e não alto desempenho (*high performance*) [85, 165, 191, 258]. Entenda-se disto que Condor visa fornecer desempenho sustentável a médio e longo prazos, mesmo que o desempenho instantâneo do sistema possa variar consideravelmente.

Condor foi inicialmente concebido para funcionar em NOWs (*Network of Workstations*). Uma NOW que executa Condor denomina-se *Condor Pool*. O elemento arquitetural mais importante de um Condor Pool é o *Matchmaker*. O Matchmaker aloca tarefas a máquinas pertencentes ao Pool. Tal alocação é baseada nas necessidades de cada tarefa e nas restrições de uso de cada máquina. As necessidades de uma tarefa são especificadas pelo usuário quando de sua submissão. Por exemplo, uma tarefa pode precisar de uma máquina Sun Sparc, rodando Solaris, com pelo menos 256MB de memória. Já as restrições de uso de uma dada máquina, estas são especificadas por seu dono quando da inclusão da máquina no Pool. Por

exemplo, o dono pode preferir que sua máquina execute as aplicações de João, seguindo das aplicações do grupo de sistemas operacionais, e que nunca execute as aplicações de Pedro. Ou seja, as restrições permitem ao dono determinar como sua máquina será usada no Condor Pool. Tipicamente, o dono estabelece também que sua máquina só é usada quando estiver ociosa e que, quando ele voltar a utilizar a máquina, qualquer aplicação Condor em execução seja suspensa imediatamente.

Um aspecto interessante do Condor é que ambos usuários e donos de máquinas são representados no sistema por agentes de software. O *Customer Agent* (Agente do Usuário) envia as necessidades da tarefa para o Matchmaker. De forma similar, *Resource Owner Agent* envia as restrições de uso do recurso ao Matchmaker. Ao efetuar o casamento de padrões entre tarefa os requisitos da tarefa e os atributos da máquina, o Matchmaker notifica ambos Agentes. A partir daí, os Agentes interagem diretamente para realizar a execução da tarefa. A Figura 13.23 ilustra este protocolo.

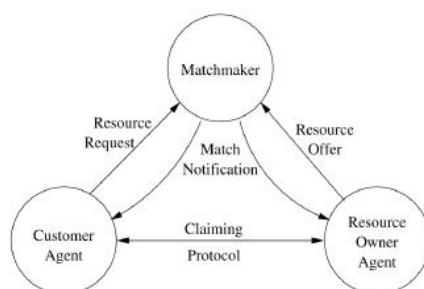


Figura 13.23: Condor protocol [85]

Outro aspecto atraente do Condor é seu mecanismo de *checkpointing*. Uma vez que o dono normalmente especifica que sua máquina seja desocupada pela tarefa Condor assim que ele retornar a usá-la, garantir progresso das tarefas torna-se um problema não-trivial. Condor aborda esta questão fazendo o checkpoint da tarefa (i.e. salvando transparentemente o estado de sua execução). Isto permite que a tarefa seja reexecutada em outra máquina a partir do ponto em que parou. Vale salientar que o mecanismo de checkpoint do Condor é implementado através da substituição da biblioteca do sistema por uma biblioteca Condor.

Condor foi posteriormente estendido para execução em Grids [165, 191]. É interessante notar que Condor dispõe de duas formas de funcionamento em Grids: Flock of Condors [165] e Condor-G [191].

Flock of Condors é um trabalho que antecede Condor-G. Um Flock of Condors é um Grid formado por vários Condor Pools [165]. A construção do Flock é bastante elegante do ponto de vista de sistemas distribuídos pois não acrescenta nenhuma centralização a arquitetura Condor original. A base para criação de um Flock é um acordo de cooperação (de troca de recursos) entre dois Condors Pools. Portanto, por maior que seja o Flock, suas ligações são sempre dois-a-dois, sem envolver nenhuma entidade centralizadora. Mais que isso, o Flock of Condors não chega a alterar o software Condor original. Toda a funcionalidade do Flock of Condors é implementada por uma máquina especial, chamada Gateway. Ambos os Pools que firmam um acordo de cooperação instalam cada qual um Gateway. Os dois Gateways mantêm comunicação constante para troca de tarefas entre os Pools. Para o Pool local, o Gateway é uma máquina qualquer. Entretanto, ao invés de oferecer seus próprios recursos, o Gateway simplesmente representa os recursos do Pool remoto, republicando as restrições estabelecidas pelos donos das máquinas remotas. Quando uma tarefa é recebida pelo Gateway, esta é repassada para o Gateway remoto que então a encaminha para uma máquina do pool remoto.

Talvez por ser mais recente, o Condor-G adota uma visão mais heterogênea de Grid. Além de Condor Pools, Condor-G também utiliza recursos via Globus. Devido à necessidade de suportar mais heterogeneidade, Condor-G usa uma arquitetura mais centralizada que o Flock of Condors. O Condor-G Scheduler controla a execução da aplicação, submetendo tarefas tanto a Condor Pools quanto a recursos acessíveis via Globus (como MPPs). No caso de recursos Globus, Condor-G utiliza os serviços GRAM, GASS, MDS e GSI para viabilizar a execução das tarefas.

13.5 Tendências em Grids Computacionais

Neste capítulo foi apresentando os principais aspectos dos Grids Computacionais, desde a apresentação de um conceito do que classifica uma infraestrutura de computação distribuída como um Grid Computacional, até o estado atual do

desenvolvimento de tecnologia para a construção dessas infraestruturas.

Vimos que Grids Computacionais levantam questões em várias áreas de pesquisa, porém seus benefícios vão além de uma simples plataforma para execução de aplicações em larga escala. A idéia é facilitar a colaboração de grupos de pesquisa distribuídos geograficamente.

Mostramos então que os Grids Computacionais como uma tecnologia que nasceu na comunidade de alto desempenho e ganhou espaço na indústria através da transformação da computação distribuída pelo uso de serviços sob demanda. Assim, a convergência de tecnologias que culminou com a definição do conceito de *Grid Service* foi um processo natural de evolução dos Grids.

Tendo em vista a nova perspectiva que os *Grid Services* trouxeram para o desenvolvimento dos Grids, a tendência é que Grids se tornem uma infraestrutura global de computação orientada a serviços, atendendo tanto a comunidade de computação científica de alto desempenho, quanto a comunidade de computação comercial.

Finalmente, a discussão sobre os padrões emergentes para o desenvolvimento de infraestruturas de Grids Computacionais, mostra que os esforços têm amadurecido, fomentado a pesquisa e o desenvolvimento de ferramentas, que contribuem para a concretização de um ambiente amplamente distribuído onde será possível consumir serviços computacionais de forma transparente.

Capítulo 14

Virtualização de recursos

Virtualização é o modo de apresentação ou agrupamento de um subconjunto lógico de recursos computacionais de modo que possam ser alcançados resultados e benefícios como se o sistema estivesse executando sobre a configuração nativa. A virtualização de recursos geralmente incluem o armazenamento de dados e poder de processamento. Deste modo, a virtualização dos recursos não é restrita somente à execução, posição geográfica ou pela configuração física de recursos.

Uma tendência nova na virtualização é o conceito de um “motor de virtualização” que dê uma visão holística de toda a infraestrutura de rede usando a técnica de agregação. Um outro tipo popular de virtualização e atualmente muito utilizado é a virtualização de hardware para rodar mais de um sistema operacional ao mesmo tempo, através de microkernels ou de camadas de abstração de hardware, como por exemplo o Xen.

14.1 Principais tipos de virtualização

Virtualização é um termo muito amplo que leva à abstração de recursos em diferentes aspectos da computação. O conceito “virtualização” foi concebido pela área de TI para se referir a tudo que quisessem dizer sobre máquinas virtuais e a softwares da gerência de sistemas, o que acaba tornando o sentido do termo muito amplo.

14.1.1 Virtualização por software

Uma máquina virtual é um ambiente que se apresenta como um sistema operacional “convidado” no hardware mas é simulado em um ambiente de software dentro do sistema hospedeiro¹. A simulação dos drivers do hardware deve ser muito robusta para o sistema hospedeiro funcionar corretamente. A criação e a gerência de máquinas virtuais são freqüentemente consultadas pelo software de virtualização também chamado de **servidor de virtualização**. Há diversas soluções, considerando:

- **Emulação**, a máquina virtual simula todo o hardware, permitindo que um Sistema Operacional sem modificações rode em um processador central completamente diferente do hardware nativo. (Bochs, PearPC, versões do virtual PC para PPC, Qemu sem aceleração)
- **Virtualização nativa** ou “virtualização cheia”, a máquina virtual somente simula parcialmente o hardware para permitir que um Sistema Operacional sem modificações funcione isoladamente no hardware, mas o Sistema Operacional convidado deve ser projetado para o tipo de processador central. (VMware, Parallels Desktop, Adeos, Mac-on-Linux, Xen)
- **Paravirtualização**, a máquina virtual não simula o hardware mas oferece preferencialmente um API especial que requer modificações do kernel do Sistema Operacional hospede. As chamadas de sistema ao hypervisor é conhecida como paravirtualização no Xen.
- **Virtualização no nível do sistema operacional**, Virtualiza um servidor no nível do sistema operacional, permitindo o múltiplos isolamentos de modo seguro aos servidores virtualizados, em um único servidor físico. Os ambientes dos Sistemas Operacionais hospedes são os mesmos que o do Sistema hospedeiro, já que o mesmo kernel do hardware é usado para executar os ambientes no hospedeiro. (Linux-VServer, Virtuozzo e OpenVZ, Solaris Containers, User Mode Linux e FreeBSD Jails)

A Virtualização de aplicação envolve o uso de uma aplicação desktop ou server localmente, usando recursos locais, sem ser instalado (comparado com os sis-

¹O hardware propriamente dito, ou seja, o sistema base que irá receber os outros sistemas operacionais.

temas de instalação e terminal services). A aplicação virtualizada roda em um pequeno ambiente virtual que contém as entradas de registro, arquivos e outros componentes necessários para execução. Este ambiente virtual age como uma camada entre a aplicação e o sistema operacional e elimina conflitos entre a aplicação e as aplicações do sistema operacional. (Softtricity, Thinstall, Appstream, Ardence, Trigence, Neoware)

14.2 XEN - Xen virtual machine monitor

Xen é um Monitor de Máquinas Virtuais (VMM) que provê uma camada de abstração entre o hardware e o sistema operacional virtualizado. Todo o código do micro-kernel e das aplicações da VMM do Xen está sob GPL.

Xen provê paravirtualização de Sistemas Operacionais com alterações no kernel para a arquitetura xen em hardwares x86/x86_64 e *full virtualization* em hardwares x86/x86_64 com suporte a virtualização assistida sem necessidade de modificações do sistema operacional hospede.

O Xen é mantido pela Universidade de Cambridge e conta com apoio de empresas globais da área de tecnologia da informação tais como IBM, HP, Intel, AMD entre outras. Para maiores informações acesse o site do projeto na Universidade de Cambridge <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/> ou o site Xen Sources <http://www.xensource.com>.

Alguns ports do Xen estão disponíveis para outros sistemas operacionais como NetBSD, FreeBSD e Solaris.

A dependência do Sistema Operacional de um hardware exclusivo parece estar se tornando coisa do passado. No futuro se pretende ter um Sistema Operacional que não dependa exclusivamente do hardware e de sua localização geográfica.

O Xen nasceu do projeto NetOS (Networks and Operating Systems), criado pelo "Computer Laboratory's Systems Research Group" e pretende, como o próprio nome do projeto pai sugere, criar uma camada de abstração onde o Sistema Operacional "navegue" nos recursos dos servidores por uma rede TCP/IP.

Trazendo estes conceitos para o nosso presente, imagine o seguinte cenário: poderemos estar acessando um bit de uma determinada aplicação em um dado momento do tempo rodando em um servidor físico no Brasil e em outro momento este mesmo bit estar localizado em outro continente sem que ao menos o usuário que acessa este bit perceba a mudança geográfica do Sistema Operacional.

Aliando-se sistemas de balanceamento de carga, alta disponibilidade, consolidação de recursos computacionais e sistemas de arquivos em cluster, esta tarefa parece estar se materializando. Neste horizonte, o que torna o Xen a aposta correta é o fato dele já cumprir com grande parte das tarefas projetadas.

14.2.1 Comparação

Ao contrário do VMWare, o Xen é executado diretamente sobre o hardware, no *ring 0*, e possui uma máquina virtual chamada de *Dom0*. Essa máquina tem acesso privilegiado ao *hypervisor*, permitindo a ela as operações necessárias que as demais máquinas virtuais não podem executar. A máquina virtual *Dom0* é então responsável pelo gerenciamento de toda a estrutura de gerenciamento de virtualização fazendo uso de aplicações que tem acesso ao *hypervisor*. É nesta máquina virtual que se parametriza a virtualização do *hardware* e a fatia entregue para cada máquina virtual que não tenha acesso direto ao *hardware* e ao *hypervisor*.

No site de notícias da Sun <http://www.sun.com/emrkt/innercircle/newsletter/brazil/0706vass.html>, é feita algumas ponderações sobre tecnologias de virtualização:

- Vmware:

Embora atraente, a abordagem de máquina virtual do VMware pode ser relativamente cara em termos de desempenho. Geralmente, o *hypervisor* consome entre 5 e 15% da potência total da CPU, enquanto cada sistema operacional aumenta a carga. No final, as empresas podem consumir uma grande quantidade de recursos da CPU simplesmente para comportar a infra-estrutura de máquina virtual.

- Xen:

Recentemente, o software de código aberto, Xen, surgiu como alternativa

ao VMware. Como o VMware, o Xen suporta a execução de vários sistemas operacionais no mesmo *hardware*. O Xen é uma forma de virtualização de nível mais baixo, com a qual os administradores podem virtualizar várias partes de um sistema, incluindo a memória e a CPU. Como ele reside em um nível tão baixo, oferece isolamento de falhas e recursos consideráveis.

Há vários motivos para se considerar o Xen:

- é um programa de código aberto,
- é relativamente leve, portanto, não consome uma quantidade absurda de recursos da CPU.
- atinge um alto grau de isolamento entre as tecnologias de máquina virtual.
- Como qualquer outra tecnologia de máquina virtual, suporta combinações variadas de sistemas operacionais e versões, além de permitir que os administradores iniciem e executem dinamicamente uma instância do sistema operacional sem afetar o serviço.

14.2.2 Sistema Operacional nativo versus virtualização com Xen

Algumas justificativas são válidas para a utilização de virtualização. Ainda no site de notícias da Sun <http://www.sun.com/emrkt/innercircle/newsletter/brazil/0106feature.html>:

“A popularidade da virtualização tem a ver com seu princípio filosófico - a convicção de que os data centers estão abarrotados de servidores subutilizados. Ela parece solucionar o problema criado pelo paradigma predominante do ‘um servidor para um aplicativo’ que resulta do superprovisionamento visando à máxima performance. As taxas de utilização de servidores podem oscilar entre 5% e 15%. Por fim, a promessa de servidores baseados em commodities resultou em data centers excessivamente caros de gerenciar, alimentar e refrigerar.”

Esta afirmação faz cair por terra a tese de que é necessário o uso de “um servidor por aplicação”, considerando que servidor neste caso é subentendido por *hardware*. As taxas de utilização do processador do *hardware* podem ser melhor aproveitadas utilizando sistemas de virtualização, aumentando o uso dos processadores (o Xen provê virtualização dos processadores ou mesmo balanceamento de carga entre eles), reduzindo o espaço físico do *Data Center* e em contra partida reduzindo o consumo de energia elétrica tanto para a alimentação dos servidores quanto para outros tens como condicionador de ar.

Ainda, com Xen é possível manter um SLA muito interessante. A possibilidade de dar manutenção física nos servidores sem necessidade de parada dos serviços é um diferencial que todo administrador deseja ter para não sofrer no momento da parada de um hardware. Basta para isso ter um outro servidor configurado e migrar em tempo real (50ms) o sistema operacional de um domínio para outro.

14.2.3 Paravirtualização no Xen

O Xen usa uma técnica completamente diferente do que conceitualmente é utilizada em outros *hypervisors*. Na paravirtualização, o Sistema Operacional hospede é portado para uma camada de *hardware* (ring 1) que virtualiza todas as relações do Sistema Operacional com o *hardware*. Quando o Sistema Operacional atualiza estruturas de dados do *hardware*, tais como a tabela de paginação ou da início a uma operação do acesso direto da memória, o Sistema Operacional hospede faz chamadas na API que é oferecida pelo *hypervisor*.

Isto, por sua vez, permite que o *hypervisor* mantenha o controle de todas as mudanças feitas pelo Sistema Operacional, e decida como modificar as interrupções do *hardware*. O *hypervisor* é mapeado no endereço de cada Sistema Operacional hospede, minimizando o tempo do interrupção entre todo o *hardware* e o *hypervisor*. Finalmente, trabalhando cooperativamente com os Sistemas Operacionais hospedes, o *hypervisor* ganha a introspecção adicional do Sistema Operacional, e pode fazer com que ele fique ciente do fato que foi virtualizado. Isto pode ser uma grande vantagem para o sistema hospedeiro - por exemplo: o *hypervisor* pode informar ao hospede em *real-time* qual foi sua última atividade, permitindo um re-escalonamento bem mais eficiente dos sistemas hospedados.

A paravirtualização disponibiliza benefícios significantes em termos de drivers de dispositivos e interfaces. Essencialmente, drivers de dispositivos podem ser virtualizados utilizando o modelo de paravirtualização, e assim, garantindo recursos de baixo nível separados por domínios como memória, CPU e outros recursos. Além disso, o próprio *hypervisor* é protegido de eventuais erros e problemas com os drivers dos dispositivos e ainda pode-se empregar qualquer dispositivo disponível no mercado não precisando assim um *hardware* ou driver específico. E ainda, sistemas Operacionais virtualizados são muito mais portáteis quando virtualizados pelo *hardware*: eles são virtualizados em níveis baixos e, a gerência do *hardware* são módulos que funcionam sob o controle do *hypervisor*.

14.2.4 Virtualização nativa no Xen

Virtualização nativa, é também conhecida como virtualização acelerada ou híbrida, é uma combinação de virtualização nativa e virtualização de I/O (entrada e saída). Tipicamente, este método é iniciado com um VMM (Virtual Machine Monitor) com suporte a virtualização cheia, como o Xen por exemplo, e então, baseando-se na análise de desempenho, emprega as técnicas de aceleração. O uso do processador e também drivers de rede são os recursos mais comuns onde é empregada a virtualização nativa.

Uma técnica similar à virtualização nativa é usada em mainframes. Na virtualização de hardwares x86, a primeira implementação de virtualização nativa foi feita com o software Virtual Iron <http://www.virtualiron.com>.

Uma vantagem da virtualização nativa, é que esta reduz a maioria das despesas de manutenção da paravirtualização no que tange a quantidade de alterações necessárias no sistema operacional convidado, e também obtém também considerável ganho de desempenho comparando com paravirtualização.

Uma desvantagem da virtualização nativa é requerer que o convidado carregue módulos que podem afetar a sua sustentação. Ainda, existe uma certa limitação quanto ao número de sistemas operacionais convidados rodando eficientemente em uma VMM. É provável que, com as novas tecnologias de virtualização nativa de X86 e X86_64 da Intel (*Vander Pool*) e da AMD (*Pacifica*), o alcance de melhoras nestes quesitos possam estar sendo alcançados. Times de ambas as empresas tem

colaborado com o projeto Xen, o que pode trazer bons frutos para a ferramenta.

Parte IV

Apêndices

Apêndice A

Licença CC-GNU GPL



Figura A.1: Creative Commons

Licença Pública Geral do GNU (GPL) [General Public License]

Versão 2¹, Junho de 1991 Direitos Autorais Reservados (c) 1989, 1991 *Free Software Foundation, Inc.* 59 Temple Place, Suite [conjunto] 330, Boston, MA [Massachusetts] 02111-1307 USA [Estados Unidos da América]

É permitido a qualquer pessoa copiar e distribuir cópias sem alterações deste documento de licença, sendo vedada, entretanto, qualquer modificação.

Introdução

As licenças da maioria dos *softwares* são elaboradas para suprimir sua liberdade de compartilhá-los e modificá-los. A Licença Pública Geral do GNU, ao contrário,

¹ Disponível em <http://creativecommons.org/licenses/GPL/2.0/legalcode.pt>.

visa garantir sua liberdade de compartilhar e modificar *softwares* livres para assegurar que o *software* seja livre para todos os seus usuários. Esta Licença Pública Geral é aplicável à maioria dos *softwares* da Free Software Foundation [Fundação do *Software* livre] e a qualquer outro programa cujos autores se comprometerem a usá-la. (Em vez dela, alguns outros *softwares* da Free Software Foundation são cobertos pela Licença Pública Geral de Biblioteca do GNU). Você também poderá aplicá-la aos seus programas.

Quando falamos de *Software* Livre, estamos nos referindo à liberdade, não ao preço. Nossas Licenças Públicas Gerais visam garantir que você tenha a liberdade de distribuir cópias de *Software* Livre (e cobrar por isso se desejar), que receba código-fonte ou possa obtê-lo se desejar, que possa modificá-lo ou usar partes dele em novos programas livres; finalmente, que você tenha ciência de que pode fazer tudo isso.

Para proteger seus direitos, precisamos fazer restrições que proíbem que alguém negue esses direitos a você ou que solicite que você renuncie a eles. Essas restrições se traduzem em determinadas responsabilidades que você deverá assumir, se for distribuir cópias do *software* ou modificá-lo.

Por exemplo, se você distribuir cópias de algum desses programas, tanto gratuitamente como mediante uma taxa, você terá de conceder aos receptores todos os direitos que você possui. Você terá de garantir que, também eles, recebam ou possam obter o código-fonte. E você terá a obrigação de exibir a eles esses termos, para que eles conheçam seus direitos.

Protegemos seus direitos através de dois passos: (1) estabelecendo direitos autorais sobre o *software* e (2) concedendo a você esta licença, que dá permissão legal para copiar, distribuir e/ou modificar o software.

Além disso, para a proteção de cada autor e a nossa, queremos ter certeza de que todos entendam que não há nenhuma garantia para este *Software* Livre. Se o *software* for modificado por alguém e passado adiante, queremos que seus receptores saibam que o que receberam não é o original, de forma que quaisquer problemas introduzidos por terceiros não afetem as reputações dos autores originais.

Finalmente, qualquer programa livre é constantemente ameaçado por patentes de software. Queremos evitar o risco de que redistribuidores de um programa li-

vre obtenham individualmente licenças sob uma patente, tornando o programa, com efeito, proprietário. Para impedir isso, deixamos claro que qualquer patente deve ser licenciada para o uso livre por parte de qualquer pessoa ou, então, simplesmente não deve ser licenciada.

Os exatos termos e condições para cópia, distribuição e modificação seguem abaixo. TERMOS E CONDIÇÕES PARA CÓPIA, DISTRIBUIÇÃO E MODIFICAÇÃO.

1. Esta Licença se aplica a qualquer programa ou outra obra que contenha um aviso inserido pelo respectivo titular dos direitos autorais, informando que a referida obra pode ser distribuída em conformidade com os termos desta Licença Pública Geral. O termo “Programa”, utilizado abaixo, refere-se a qualquer programa ou obra, e o termo “obras baseadas no Programa” significa tanto o Programa, como qualquer obra derivada nos termos da legislação de direitos autorais: isto é, uma obra contendo o Programa ou uma parte dele, tanto de forma idêntica como com modificações, e/ou traduzida para outra linguagem. (Doravante, o termo “modificação” inclui também, sem reservas, a tradução). Cada licenciado, doravante, será denominado “você”.

Outras atividades que não a cópia, distribuição e modificação, não são cobertas por esta Licença; elas estão fora de seu escopo. O ato de executar o Programa não tem restrições e o resultado gerado a partir do Programa encontra-se coberto somente se seu conteúdo constituir uma obra baseada no Programa (independente de ter sido produzida pela execução do Programa). Na verdade, isto dependerá daquilo que o Programa faz.

2. Você poderá fazer cópias idênticas do código-fonte do Programa ao recebê-lo e distribuí-las, em qualquer mídia ou meio, desde que publique, de forma ostensiva e adequada, em cada cópia, um aviso de direitos autorais (ou copyright) apropriado e uma notificação sobre a exoneração de garantia; mantenha intactas as informações, avisos ou notificações referentes a esta Licença e à ausência de qualquer garantia; e forneça a quaisquer outros receptores do Programa uma cópia desta Licença junto com o Programa.

Você poderá cobrar um valor pelo ato físico de transferir uma cópia, e você pode oferecer, se quiser, a proteção de uma garantia em troca de um valor.

3. Você poderá modificar sua cópia ou cópias do Programa ou qualquer parte dele, formando, dessa forma, uma obra baseada no Programa, bem como copiar e distribuir essas modificações ou obra, de acordo com os termos da Cláusula 1 acima, desde que você também atenda a todas as seguintes condições:

- a. Você deve fazer com que os arquivos modificados contenham avisos, em destaque, informando que você modificou os arquivos, bem como a data de qualquer modificação.
- b. Você deve fazer com que qualquer obra que você distribuir ou publicar, que no todo ou em parte contenha o Programa ou seja dele derivada, ou derivada de qualquer parte dele, seja licenciada como um todo sem qualquer custo para todos terceiros nos termos desta licença.
- c. Se o programa modificado normalmente lê comandos interativamente quando executado, você deverá fazer com que ele, ao começar a ser executado para esse uso interativo em sua forma mais simples, imprima ou exiba um aviso incluindo o aviso de direitos autorais (ou copyright) apropriado, além de uma notificação de que não há garantia (ou, então, informando que você oferece garantia) e informando que os usuários poderão redistribuir o programa de acordo com essas condições, esclarecendo ao usuário como visualizar uma cópia desta Licença. (Exceção: se o Programa em si for interativo mas não imprimir normalmente avisos como esses, não é obrigatório que a sua obra baseada no Programa imprima um aviso).

Essas exigências se aplicam à obra modificada como um todo. Se partes identificáveis dessa obra não forem derivadas do Programa e puderem ser consideradas razoavelmente como obras independentes e separadas por si próprias, nesse caso, esta Licença e seus termos não se aplicarão a essas partes quando você distribui-las como obras separadas. Todavia, quando você distribui-las como parte de um todo que constitui uma obra baseada no Programa, a distribuição deste todo terá de ser realizada em conformidade com esta Licença, cujas permissões para outros licenciados se estenderão à obra por completo e, conseqüentemente, a toda e qualquer parte, independentemente de quem a escreveu.

Portanto, esta cláusula não tem a intenção de afirmar direitos ou contestar os seus direitos sobre uma obra escrita inteiramente por você;

a intenção é, antes, de exercer o direito de controlar a distribuição de obras derivadas ou obras coletivas baseadas no Programa.

Além do mais, a simples agregação de outra obra que não seja baseada no Programa a ele (ou a uma obra baseada no Programa) em um volume de mídia ou meio de armazenamento ou distribuição, não inclui esta outra obra no âmbito desta Licença.

4. Você poderá copiar e distribuir o Programa (ou uma obra baseada nele, de acordo com a Cláusula 2) em código-objeto ou formato executável de acordo com os termos das Cláusulas 1 e 2 acima, desde que você também tome uma das providências seguintes:
 - a. Incluir o código-fonte correspondente completo, passível de leitura pela máquina, o qual terá de ser distribuído de acordo com as Cláusulas 1 e 2 acima, em um meio ou mídia habitualmente usado para intercâmbio de software; ou,
 - b. Incluir uma oferta por escrito, válida por pelo menos três anos, para fornecer a qualquer terceiro, por um custo que não seja superior ao seu custo de fisicamente realizar a distribuição da fonte, uma cópia completa passível de leitura pela máquina, do código-fonte correspondente, a ser distribuído de acordo com as Cláusulas 1 e 2 acima, em um meio ou mídia habitualmente usado para intercâmbio de software; ou,
 - c. Incluir as informações recebidas por você, quanto à oferta para distribuir o código-fonte correspondente. (Esta alternativa é permitida somente para distribuição não-comercial e apenas se você tiver recebido o programa em código-objeto ou formato executável com essa oferta, de acordo com a letra b, acima).

O código-fonte de uma obra significa o formato preferencial da obra para que sejam feitas modificações na mesma. Para uma obra executável, o código-fonte completo significa o código-fonte inteiro de todos os módulos que ela contiver, mais quaisquer arquivos de definição de interface associados, além dos *scripts* usados para controlar a compilação e instalação do executável. Entretanto, como uma exceção especial, o código-fonte distribuído não precisa incluir nada que não seja normalmente distribuído (tanto no formato fonte como no binário) com os componentes principais (compilador, kernel e assim por diante) do sistema operacional no qual o executável é executado, a menos que este componente em si acompanhe o executável.

Se a distribuição do executável ou código-objeto for feita mediante a permissão de acesso para copiar, a partir de um local designado, então, a permissão de acesso equivalente para copiar o código-fonte a partir do mesmo local será considerada como distribuição do código-fonte, mesmo que os terceiros não sejam levados a copiar a fonte junto com o código-objeto.

5. Você não poderá copiar, modificar, sublicenciar ou distribuir o Programa, exceto conforme expressamente estabelecido nesta Licença. Qualquer tentativa de, de outro modo, copiar, modificar, sublicenciar ou distribuir o Programa será inválida, e automaticamente rescindirá seus direitos sob esta Licença. Entretanto, terceiros que tiverem recebido cópias ou direitos de você de acordo esta Licença não terão suas licenças rescindidas, enquanto estes terceiros mantiverem o seu pleno cumprimento.
6. Você não é obrigado a aceitar esta Licença, uma vez que você não a assinou. Porém, nada mais concede a você permissão para modificar ou distribuir o Programa ou respectivas obras derivativas. Tais atos são proibidos por lei se você não aceitar esta Licença. Conseqüentemente, ao modificar ou distribuir o Programa (ou qualquer obra baseada no Programa), você estará manifestando sua aceitação desta Licença para fazê-lo, bem como de todos os seus termos e condições para copiar, distribuir ou modificar o Programa ou obras nele baseadas.
7. Cada vez que você redistribuir o Programa (ou obra baseada no Programa), o receptor receberá, automaticamente, uma licença do licenciante original, para copiar, distribuir ou modificar o Programa, sujeito a estes termos e condições. Você não poderá impor quaisquer restrições adicionais ao exercício, pelos receptores, dos direitos concedidos por este instrumento. Você não tem responsabilidade de promover o cumprimento por parte de terceiros desta licença.
8. Se, como resultado de uma sentença judicial ou alegação de violação de patente, ou por qualquer outro motivo (não restrito às questões de patentes), forem impostas a você condições (tanto através de mandado judicial, contrato ou qualquer outra forma) que contradigam as condições desta Licença, você não estará desobrigado quanto às condições desta Licença. Se você não puder atuar como distribuidor de modo a satisfazer simultaneamente suas obrigações sob esta licença e quaisquer outras obrigações pertinentes,

então, como consequência, você não poderá distribuir o Programa de nenhuma forma. Por exemplo, se uma licença sob uma patente não permite a redistribuição por parte de todos aqueles que tiverem recebido cópias, direta ou indiretamente de você, sem o pagamento de royalties, então, a única forma de cumprir tanto com esta exigência quanto com esta licença será deixar de distribuir, por completo, o Programa.

Se qualquer parte desta Cláusula for considerada inválida ou não executável, sob qualquer circunstância específica, o restante da cláusula deverá continuar a ser aplicado e a cláusula, como um todo, deverá ser aplicada em outras circunstâncias.

Esta cláusula não tem a finalidade de induzir você a infringir quaisquer patentes ou direitos de propriedade, nem de contestar a validade de quaisquer reivindicações deste tipo; a única finalidade desta cláusula é proteger a integridade do sistema de distribuição do *Software* Livre, o qual é implementado mediante práticas de licenças públicas. Muitas pessoas têm feito generosas contribuições à ampla gama de *software* distribuído através desse sistema, confiando na aplicação consistente deste sistema; cabe ao autor / doador decidir se deseja distribuir *software* através de qualquer outro sistema e um licenciado não pode impor esta escolha.

Esta cláusula visa deixar absolutamente claro o que se acredita ser uma consequência do restante desta Licença.

9. Se a distribuição e/ou uso do Programa for restrito em determinados países, tanto por patentes ou por interfaces protegidas por direito autoral, o titular original dos direitos autorais que colocar o Programa sob esta Licença poderá acrescentar uma limitação geográfica de distribuição explícita excluindo esses países, de modo que a distribuição seja permitida somente nos países ou entre os países que não foram excluídos dessa forma. Nesse caso, esta Licença passa a incorporar a limitação como se esta tivesse sido escrita no corpo desta Licença.
10. A Free *Software* Foundation poderá de tempos em tempos publicar novas versões e/ou versões revisadas da Licença Pública Geral. Essas novas versões serão semelhantes em espírito à presente versão, mas podem diferenciar-se, porém, em detalhe, para tratar de novos problemas ou preocupações.

Cada versão recebe um número de versão distinto. Se o Programa especificar um número de versão desta Licença que se aplique a ela e a “qualquer

versão posterior”, você terá a opção de seguir os termos e condições tanto daquela versão como de qualquer versão posterior publicada pela *Free Software Foundation*. Se o Programa não especificar um número de versão desta Licença, você poderá escolher qualquer versão já publicada pela *Free Software Foundation*.

11. Se você deseja incorporar partes do Programa em outros programas livres cujas condições de distribuição sejam diferentes, escreva ao autor solicitando a respectiva permissão. Para *software* cujos direitos autorais sejam da *Free Software Foundation*, escreva para ela; algumas vezes, abrimos exceções para isso. Nossa decisão será guiada pelos dois objetivos de preservar a condição livre de todos os derivados de nosso *Software Livre* e de promover o compartilhamento e reutilização de software, de modo geral.

EXCLUSÃO DE GARANTIA

11. COMO O PROGRAMA É LICENCIADO SEM CUSTO, NÃO HÁ NENHUMA GARANTIA PARA O PROGRAMA, NO LIMITE PERMITIDO PELA LEI APLICÁVEL. EXCETO QUANDO DE OUTRA FORMA ESTABELECIDO POR ESCRITO, OS TITULARES DOS DIREITOS AUTORAIS E/OU OUTRAS PARTES, FORNECEM O PROGRAMA “NO ESTADO EM QUE SE ENCONTRA”, SEM NENHUMA GARANTIA DE QUALQUER TIPO, TANTO EXPRESSA COMO IMPLÍCITA, INCLUINDO, DENTRE OUTRAS, AS GARANTIAS IMPLÍCITAS DE COMERCIALIZABILIDADE E ADEQUAÇÃO A UMA FINALIDADE ESPECÍFICA. O RISCO INTEGRAL QUANTO À QUALIDADE E DESEMPENHO DO PROGRAMA É ASSUMIDO POR VOCÊ. CASO O PROGRAMA CONTENHA DEFEITOS, VOCÊ ARCARÁ COM OS CUSTOS DE TODOS OS SERVIÇOS, REPAROS OU CORREÇÕES NECESSÁRIAS.
12. EM NENHUMA CIRCUNSTÂNCIA, A MENOS QUE EXIGIDO PELA LEI APLICÁVEL OU ACORDADO POR ESCRITO, QUALQUER TITULAR DE DIREITOS AUTORAIS OU QUALQUER OUTRA PARTE QUE POSSA MODIFICAR E/OU REDISTRIBUIR O PROGRAMA, CONFORME PERMITIDO ACIMA, SERÁ RESPONSÁVEL PARA COM VOCÊ POR DANOS, INCLUINDO ENTRE OUTROS, QUAISQUER DANOS GERAIS, ESPECIAIS, FORTUITOS OU EMERGENTES, ADVINDOS DO USO OU IMPOSSIBILIDADE DE USO DO PROGRAMA (INCLUINDO, ENTRE OUTROS,

PERDAS DE DADOS OU DADOS SENDO GERADOS DE FORMA IMPRECISA, PERDAS SOFRIDAS POR VOCÊ OU TERCEIROS OU A IMPOSSIBILIDADE DO PROGRAMA DE OPERAR COM QUAISQUER OUTROS PROGRAMAS), MESMO QUE ESSE TITULAR, OU OUTRA PARTE, TENHA SIDO ALERTADA SOBRE A POSSIBILIDADE DE OCORRÊNCIA DESSES DANOS.

FINAL DOS TERMOS E CONDIÇÕES

Como Aplicar Estes Termos para Seus Novos Programas.

Se você desenvolver um programa novo e quiser que ele seja da maior utilidade possível para o público, o melhor caminho para obter isto é fazer dele um *Software* Livre, o qual qualquer pessoa pode redistribuir e modificar sob os presentes termos.

Para fazer isto, anexe as notificações seguintes ao programa. É mais seguro anexá-las ao começo de cada arquivo-fonte, de modo a transmitir do modo mais eficiente a exclusão de garantia; e cada arquivo deve ter ao menos a linha de “direitos autorais reservados” e uma indicação de onde a notificação completa se encontra.

<uma linha para informar o nome do programa e uma breve idéia do que ele faz.> Direitos Autorais Reservados (c) <nome do autor> Este programa é *Software* Livre; você pode redistribuí-lo e/ou modificá-lo sob os termos da Licença Pública Geral GNU conforme publicada pela Free *Software* Foundation; tanto a versão 2 da Licença, como (a seu critério) qualquer versão posterior.

Este programa é distribuído na expectativa de que seja útil, porém, SEM NENHUMA GARANTIA; nem mesmo a garantia implícita de COMERCIALIZABILIDADE OU ADEQUAÇÃO A UMA FINALIDADE ESPECÍFICA. Consulte a Licença Pública Geral do GNU para mais detalhes.

Você deve ter recebido uma cópia da Licença Pública Geral do GNU junto com este programa; se não, escreva para a Free *Software* Foundation, Inc., no endereço 59 Temple Street, Suite 330, Boston, MA 02111-

1307 USA. Inclua também informações sobre como contatar você por correio eletrônico e por meio postal.

Se o programa for interativo, faça com que produza uma pequena notificação como esta, quando for iniciado em um modo interativo:

Versão 69 do Gnomovision, Direitos Autorais Reservados (c) ano
nome do autor. O Gnomovision NÃO POSSUI QUALQUER TIPO DE
GARANTIA; para detalhes, digite 'show w'. Este é um *Software Li-*
vre e você é bem-vindo para redistribuí-lo sob certas condições; digite
'show c' para detalhes.

Os comandos hipotéticos 'show w' e 'show c' devem mostrar as partes apropriadas da Licença Pública Geral. Naturalmente, os comandos que você utilizar poderão ter outras denominações que não 'show w' e 'show c'; eles poderão até ser cliques do *mouse* ou itens de um menu – o que for adequado ao seu programa.

Você também pode solicitar a seu empregador (se você for um programador) ou sua instituição acadêmica, se for o caso, para assinar uma “renúncia de direitos autorais” sobre o programa, se necessário. Segue um exemplo; altere os nomes:

A Yoyodyne Ltda., neste ato, renuncia a todos eventuais direitos autorais sobre o programa 'Gnomovision' (que realiza passagens em compiladores), escrito por James Hacker.

<Assinatura de Ty Coon> 1º de abril de 1989, Ty Coon, Presidente

Esta Licença Pública Geral não permite a incorporação do seu programa a programas proprietários. Se seu programa é uma biblioteca de sub-rotinas, você poderá considerar ser mais útil permitir a ligação de aplicações proprietárias à sua biblioteca. Se isso é o que você deseja fazer, utilize a Licença Pública Geral de Biblioteca do GNU, ao invés desta Licença.

Apêndice B

Marcas Registradas

Foram utilizadas marcas registradas neste Documento com o propósito de identificação. A equipe de elaboração do Guia de Cluster reconhece a propriedade dessas marcas registradas, conforme demonstrado na Tabela B.

Caso você acredite que sua marca foi utilizada sem a devida referência de propriedade, por favor, encaminhe uma mensagem para <guialivre@planejamento.gov.br>, para que a equipe de redação possa regularizar a situação nas próximas versões do Documento.

Tabela de Referência de Marcas Registradas	
Marcas Registradas	Proprietário
AT&T	AT&T
Adobe, Acrobat, Acrobat Reader, Photoshop, PageMaker, Framemaker	Adobe System Incorporated
Amiga	Amiga, Inc
Apple, Macintosh, Mac OS, AppleTalk	Apple Computer, Inc
BSD	University of California, Berkeley, USA
Citrix	Citrix Systems, Inc.
Chili!Soft	Chili!Soft Inc
Corel Draw, WordPerfect	Corel Corporation
CrossOver Office	CodeWeavers, Inc
Debian	Software in the Public Interest, Inc
Flash, Shockwave, Director	Macromedia, Inc

Marcas Registradas	Proprietário
Firebird	Firebird Project
FreeBSD	Walnut Creek CDROM, Inc
HP/UX	Hewlett-Packard Company
Hylafax Server	Silicon Graphics, Inc
IBM, Lotus, Lotus Notes, SmartSuite, MVS, Word Pro, AIX, AS/400, VM/CMS, Display Write, Lotus 123, AmiPro, DB2	IBM Corporation
Interbase, Kylix, Delphi	Borland Software Corporation
MaxDB, MySQL	MySQL AB
Windows, Windows 2000, Windows 3.x, Windows 95, Windows 98, Windows ME, Windows NT, Windows XP, Microsoft, Microsoft Excel, Microsoft Internet Explorer, Microsoft Office, Microsoft Visio, Microsoft Word, Microsoft Works, Outlook, Outlook Express, Outlook Web Access (OWA), ActiveX, DirectX, Active Directory, FrontPage, JScript, Visual Basic, Win32, Microsoft Access, ODBC, Microsoft Exchange, VBScript, SQL Server, PowerPoint, Paint	Microsoft Corporation
Netscape	Netscape Communications Corp.
NetBSD	NetBSD Foundation
Novell, Netware, NDS, Ximian, Ximian Evolution, Red Carpet, Red Carpet Enterprise	Novell, Inc
Adabas	Software/AG of North America, Inc
OSF/1	Hewlett-Packard Development Company, L.P.
Opera	Opera Software
Oracle	Oracle Corporation
PostgreSQL	PostgreSQL, Inc
Quark XPress	Quark, Inc
RealPlayer	RealNetworks, Inc
Red Hat	Red Hat, Inc
SuSE	SuSE AG

Marcas Registradas	Proprietário
Sendmail	SendMail, Inc
Sun, Solaris, Java, JDBC, StarOffice, JDK, Javascript	Sun Microsystems, Inc
Sybase	Sybase, Inc
Tarantella	Tarantella, Inc

Tabela B.1: Tabela de Referência de Marcas Registradas

Apêndice C

Lista de Abreviaturas

ATM	Assynchronous Transfer Mode
BSP	Bulk Synchronous Parallelism
CMOS	Complementary Metal Oxide Semiconductor
DRAM	Dynamic Random Access Memory
DSM	Distributed Shared Memory
ECL	Emitter Coupled Logic
FDDI	Fiber Distributed Data Interface
FFT	Fast Fourier Transformation
HIPPI	High-Performance Parallel Interface
HPF	High Performance Fortran
Mbps	Milhões de bits por segundo
MFLOPS	Milhões de Instruções de Ponto Flutuante Por Segundo
MIMD	Múltiplas seqüências de Instruções, Múltiplas seqüências de Dados
MIPS	Milhões de Instruções Por Segundo
MISD	Múltiplas Seqüências de Instruções, uma Seqüência de Dados
NFS	Network File System
NUMA	NonUniform Memory Access
OLTP	On Line Transaction Processing
PVM	Parallel Virtual Machine
SIMD	Uma Seqüência de Instruções, Múltiplas Seqüências de Dados
SISD	Uma Seqüência de Instruções, uma Seqüência de Dados
SONET	Synchronous Optical NETwork
SPMD	Simple Program Multiple Data
SR	Synchronizing Resources
SRAM	Static Random Access Memory
TCP/IP	Transmition Control Protocol/Internet Protocol

Apêndice D

Tecnologias

Tabela de referências de tecnologias que são abordadas neste Guia.

Tabela de referência de tecnologias			
Software	Licença	Versão	Site Oficial
HPC			
Beowulf			http://www.beowulf.org/
OpenSSI	GPL v2	1.9.2	http://www.openssi.org/
Kerrighed	GPL v2	1.0.2	http://www.kerrighed.org/
OpenMosix (Mosix)	GPL v2	openMosix- kernel- 2.4.26	http://openmosix.sourceforge.net/
Storage			
iSCSI	GPL	4.0.x	http://linux-iscsi.sourceforge.net/
gnbd	GPL	6.1	http://sourceware.org/cluster/gnbd/
drdb	GPL	0.7	http://www.drbd.org/
endb	GPL		http://www.it.uc3m.es/ptb/nbd/
gfs	GPL	6.1	http://www.redhat.com/software/rha/gfs/
ocfs2	GPL	1.2.3	http://oss.oracle.com/projects/ocfs2/
pvfs	GPL	1.6.3	http://www.parl.clemson.edu/pvfs/
lustre	GPL	1.4.7	http://www.lustre.org/
Cluster de Banco de Dados, Banco de Dados Distribuido			
mysql cluster VERSÃO 0.6	GPL e proprietária	5.0	http://www.mysql.com/products/database/cluster/

Software	Licença	Versão	Site Oficial
pgcluster	BSD	1.3	http://pgcluster.projects.postgresql.org/index.html
slony-l	BSD	1.1.5	http://gborg.postgresql.org/project/slony1/projdisplay.php
pgpool-I	BSD	3.1.1	http://pgpool.projects.postgresql.org/
pgpool-II	BSD	1.0.1	http://pgpool.projects.postgresql.org/pgpool-II/en/
sequoia	Apache License 2	2.10	http://sequoia.continuent.org/HomePage
pargres	GPL		http://pargres.nacad.ufrj.br/
Alta Disponibilidade/Balanceamento de Carga			
Heartbet	GPL	2.0.7	http://www.linux-ha.org/
Carp	BSD		http://www.openbsd.org/faq/pf/pt/carp.html
LVS	GPL	1.2.1	http://www.linuxvirtualserver.org/
Keepalive	GPL	1.1.12	http://www.keepalived.org/
Cluster de Aplicações			
Cluster ZOPE	ZPL	3.1	http://zope.delta.ncsu.edu/portal/delta/developers/projects/system_projects/zope_cluster
Cluster Tomcat	Apache License 2	5.0	http://jakarta.apache.org
cluster Apache	Apache License 2	2.0	http://httpd.apache.org
Ferramentas de Gerenciamento de Cluster			
ganglia	BSD	3.0	http://ganglia.sourceforge.net/
Etherboot	GPL v2		http://etherboot.sourceforge.net/

Tabela D.1: Tabela de referências de tecnologias

Apêndice E

Glossário

API (Application Programming Interface)	O método específico recomendado por um sistema operacional de computador, aplicativo ou ferramenta de terceiros, pelo qual um programador escrevendo um aplicativo pode fazer requisições do sistema operacional. Também conhecido por <i>Application Programmers Interface</i> .
APF - Administração Pública Federal	reúne órgãos da administração direta (serviços integrados na estrutura administrativa da Presidência da República e dos Ministérios) e indireta (Autarquias, Empresas Públicas, Sociedades de Economia Mista e Fundações Públicas) do Poder Executivo. https://www.planalto.gov.br/ccivil_03/decreto-lei/del0200.htm
Assíncrona	O que não ocorre ao mesmo tempo; sem relação regular de tempo, inesperado, imprevisível. Modo de transmissão no qual os dados são transmitidos sem periodicidade constante (no modo síncrono, os dados são transmitidos periodicamente); transmissão de sinais onde os intervalos de tempo entre os caracteres transmitidos podem ser diferentes, e a transmissão é controlada por elementos que indicam o início e o fim de cada caractere. Transmissão que envia um caractere de cada vez. Transmissão assíncrona é a transmissão de dados que não exige o uso de sinais externos para manter a sincronização entre emissor e receptor.

Business to Business (B2B)	Negócios feitos entre empresas, seus clientes, distribuidores e fornecedores, conectando seus sistemas de informação através da Internet.
Business to Consumer (B2C)	Negócios feitos das empresas com seus consumidores, ou seja, comércio eletrônico com o consumidor final.
CERN (Conseil Européen pour la Recherche Nucléaire / Conselho Europeu para a Pesquisa Nuclear)	Um dos mais importantes centros mundiais de pesquisas avançadas em Física Nuclear e de Partículas, localizado em Genebra/Suíça. Um de seus pesquisadores, Tim Berners Lee, foi o inventor, em 1989, do HTTP (Hypertext Transfer Protocol), o protocolo usado na WWW para transferir arquivos HTML.
CERT (Computer Emergency Response Team)	Organização criada em 1988 que oferece serviços de consulta para usuários da Internet e que entra em ação sempre que um novo vírus e outras ameaças aos computadores são descobertas.
CORBA (Common Object Request Broker Architecture)	É um padrão para comunicação entre objetos distribuídos. Provê diferentes formas para executar programas (objetos) desenvolvidos em diferentes linguagens de programação e em diferentes plataformas.
CRP (Continuous Replenishment Process)	É a prática de parceria entre os membros do canal de distribuição que altera o tradicional processo de reposição de mercadoria de geração de pedidos elaborados pelo distribuidor, baseado em quantidades economicamente convenientes, para a reposição de produtos baseada em previsão de demanda efetiva. Busca integrar, por meio de práticas distintas, o fluxo de informações e produtos.

Customer Relationship Management (CRM)	Gerenciamento do relacionamento com cliente é a arte de integrar todos os aspectos da tecnologia da informação em benefício de um completo relacionamento com o cliente, desde atividades de marketing e vendas até contas a receber.
Data warehouse	Armazém de dados, sistema que guarda e organiza todas as informações espalhadas pelos vários sistemas dentro de uma empresa. Termo genérico para um tipo de banco de dados que consiste num sistema de armazenamento, recuperação e gerenciamento de grandes quantidades de quaisquer tipos de dados. Os softwares da espécie freqüentemente incluem sofisticadas técnicas, inclusive de compactação, para buscas mais rápidas de dados, assim como filtros avançados.
DNS - Domain Name System (Sistema de Nomes de Domínio)	forma como os nomes de domínio são encontrados e traduzidos no endereço de protocolo da Internet. Um nome de domínio é um recurso fácil de ser lembrado quando referenciado como um endereço na Internet.
EAI (Enterprise Application Integration)	Integração de Aplicações entre Empresas é um tipo de tecnologia que permite o movimento e troca de informações entre diferentes aplicações e processos de negócios entre e dentro de organizações.
EDI (Electronic Data Interchange)	Intercâmbio Eletrônico de Dados - tecnologia, que permite troca de informações (com modem e softwares adequados) diretamente de computadores para computadores, dispensando digitação e manipulação de dados. O sistema que a utiliza permite automatizar transações comuns de negócios como ordens de compras, faturas, notificações de embarques etc. Através do EDI, documentos são transmitidos e recebidos eletronicamente, independente de horários, distância e dos sistemas de computação utilizados. O resultado é um fluxo de informações rápido e preciso, no qual as mensagens vão e voltam sem qualquer interferência e com toda segurança, atendendo aos desafios de maior agilidade e eficiência na comunicação de negócios.

EJB (Enterprise JavaBeans)	É um padrão de programação Java que permite que códigos escritos nesta linguagem e que sigam este padrão, possam ser distribuídos e executados em servidores de aplicação de EJB (EJB Servers).
ERP (Enterprise Resource Planning)	Planejamento de recursos corporativos através de sistemas integrados de gestão implementados por softwares; um programa integrado de gestão empresarial, geralmente dividido em diversos módulos, como o de administração, o financeiro, de manufatura etc.
FTP - File Transfer Protocol (Protocolo de Transferência de Arquivo)	é um protocolo aplicativo que utiliza os protocolos TCP/IP da Internet, sendo a maneira mais simples de trocar arquivos entre computadores na Internet.
HTTP - Hyper Text Transfer Protocol (Protocolo de Transferência de Hipertexto)	conjunto de regras para permuta de arquivos (texto, imagens gráficas, som, vídeo e outros arquivos multimídia) na World Wide Web.
IEEE - Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Elétricos e Eletrônicos)	fomenta o desenvolvimento de padrões e normas que frequentemente se tornam nacionais e internacionais.
IETF - Internet Engineering Task Force (Força Tarefa de Engenharia da Internet)	entidade que define protocolos operacionais padrão da Internet, como o TCP/IP.
Internet	Rede mundial de computadores que utiliza a arquitetura de protocolos de comunicação TCP/IP. Originou-se de um sistema de telecomunicações descentralizado criado pelo Deptº de Defesa dos Estados Unidos durante a Guerra Fria. Durante os anos 70 e 80, cresceu entre os meios acadêmicos, quando sua principal aplicação era o correio eletrônico. Com a aparição da World Wid Web em 1993, a Internet se popularizou. Provê transferências de arquivos, login remoto, correio eletrônico, news, navegação na Web e outros serviços.

JDBC	<i>Java Database Connectivity</i> . Uma especificação de interface de programa aplicativo (<i>application program interface</i> - API) para conectar programas escritos em Java aos dados em bancos de dados populares. A interface de programa aplicativo permite que se codifiquem declarações de requisição de acesso em <i>Structured Query Language</i> (SQL), as quais são então passadas para o programa que gerencia o banco de dados. O resultado é retornado por uma interface similar.
Metadados	são informações adicionais necessárias para que os dados se tornem úteis. É informação essencial para que se possa fazer uso dos dados. Em suma, metadados são um conjunto de características sobre os dados que não estão normalmente incluídas nos dados propriamente ditos. http://www.isa.utl.pt/dm/sig/sig20002001/TemaMetadados/trabalho.htm
Middleware	é um termo geral que serve para mediar dois programas separados e normalmente já existentes. Aplicações diferentes podem comunicar-se através do serviço de Messaging, proporcionado por programas middleware.
NFS (Network File System)	É o protocolo de compartilhamento de arquivos remotos desenvolvido pela Sun Microsystems. Faz parte da família de protocolos TCP/IP.
NNTP (Network News Transfer Protocol)	Padrão usado para a troca de mensagens dos usuários da Usenet na Internet.
ORBs (Object Request Brokers)	É o componente responsável por atender requisições de objetos em um framework de objetos. Principal componente da arquitetura CORBA, ele recebe requisições de clientes e disponibiliza o acesso à objetos previamente publicados em um diretório de objetos.

Padrão aberto	todo o padrão tecnológico estabelecido por órgãos internacionais ou por consórcios de empresas do mercado que desenvolvem especificações que se encontram publicamente disponíveis. O PC (computador pessoal) foi lançado e é desenvolvido com padrão aberto. As especificações da Internet e seu desenvolvimento também. A grande maioria das linguagens de programação também.
RFC - Request for Comments (Solicitação de Comentários)	documento formal da IETF, resultante de modelos e revisões de partes interessadas. A versão final do RFC tornou-se um padrão em que nem comentários nem alterações são permitidos. As alterações podem ocorrer, porém, por meio de RFCs subsequentes que substituem ou elaboram em todas as partes dos RFCs anteriores. RFC também é a abreviação de Remote Function Call (chamada funcional remota).
RPCs (Remote Procedure Calls)	É o nome dado ao ato de chamar ou executar um procedimento ou programa que não se encontra na mesma máquina do programa chamador.
SOA - Service Oriented Architecture (Arquitetura Orientada a Serviços)	Arquitetura proposta para interoperabilidade de sistemas por meio de conjunto de interfaces de serviços fracamente acoplados (loosely coupled), onde os serviços não necessitam de detalhes técnicos da plataforma dos outros serviços para a troca de informações ser realizada.
SOAP - Simple Object Access Protocol (Protocolo Simples para Acesso a Objetos)	descreve um modelo para o empacotamento de perguntas e respostas XML. O envio de mensagens SOAP é utilizado para permitir o intercâmbio de uma variedade de informações XML. A norma de SOAP assume a tarefa de transmitir pedidos e respostas sobre serviços entre usuários e fornecedores de serviços.

Software Livre	programa de computador disponível através de seu código-fonte e com a permissão para qualquer um usá-lo, copiá-lo e distribuí-lo, seja na sua forma original ou com modificações, seja gratuitamente ou com custo. O software livre é necessariamente não proprietário, mas é importante não confundir software livre com software grátis.
UDDI - Universal Description Discovery and Integration (Descrição, Descoberta e Integração Universais)	é o repositório no qual os desenvolvedores registram os Web Services disponíveis que permitem aos clientes a descoberta e a utilização dos serviços alocados em Extranets e Intranets.
W3C - World Wide Web Consortium (Consórcio da Rede Mundial Web)	associação de indústrias que visa promover padrões para a evolução da web e interoperabilidade entre produtos para WWW produzindo softwares de especificação e referência.
Web Services	Aplicação lógica, programável que torna compatíveis entre si os mais diferentes aplicativos, independentemente do sistema operacional, permitindo a comunicação e intercâmbio de dados entre diferentes redes.
WSDL - Web Services Definition Language (Linguagem para definição de Serviços Web)	é um formato XML para descrição de serviços web e suas informações para acesso. Ela descreve as funcionalidades dos serviços oferecidos pelo provedor de serviços, bem como sua localização e forma de acesso.
WWW ou Web (World Wide Web)	Área da Internet que contém documentos em formato de hipermídia, uma combinação de hipertexto com multimídia. Os documentos hipermídia da WWW são chamados de páginas de Web e podem conter textos, imagens e arquivos de áudio e vídeo, além de ligações com outros documentos na rede. A característica multimídia da Web, tornou-a a porção mais importante da Internet.

XML - eXtensible Markup Language (Linguagem Markup Estensível)	maneira flexível para criar formatos de informações comuns e compartilhar ambos os formatos e os dados na World Wide Web, nas intranets e em qualquer lugar. O XML é extensível porque, diferentemente do HTML, os símbolos markup são ilimitados e se autodefinem.
XMPP - eXtensible Messaging and Presence Protocol (Protocolo de Mensageria em Tempo Real)	Protocolo aberto, baseado em XML para mensagens em tempo real.
XSL - eXtensible Stylesheet Language	linguagem de criação de planilhas que descreve como um dado é mandado por meio da web, usando o XML, e é apresentado ao usuário. O XSL é uma linguagem para formatar um documento XML.

Apêndice F

O Ambiente LabCluster

O LabCluster é o laboratório da SLTI/MP que prove infra-estrutura tecnológica e computacional, baseada em computação distribuída e padrões abertos de *hardware* e *software* para os projetos internos ou em parceria com a Secretaria de Logística e Tecnologia da Informação. O laboratório é um ambiente de testes, prospecção e análise de tecnologias, em especial de “Cluster e Grid”.

Alguns exemplos de ações práticas da SLTI com a aplicação de tecnologias de “Cluster e Grid” neste laboratório são:

- Tamanduá: projeto piloto de mineração da base de dados de compras governamentais. O processo de mineração de dados tem como principais objetivos descobrir relacionamentos, geralmente não triviais, entre dados armazenados, fornecer subsídios para que seja possível realizar previsão de tendências futuras com base nos registros acumulados, bem como estabelecer parâmetros para análise e auditoria das informações coletadas.
- Projeto Qualidade de Dados Sociais: foi realizada uma chamada pública em 2005 e posteriormente uma aquisição de solução baseada em tecnologia de Cluster para tratamento de qualidade de dados, que busca identificar inconsistências e fraudes no acervo de bases de dados sociais do governo. Foram utilizadas as bases: SISOBÍ, SIM, GFIP, CADUNICO e do Censo Previdenciário. O acervo de dados tratado neste projeto é da ordem de 300 milhões de registros, com possibilidade de expansão para até 1 bilhão de registros.

- Projeto de Integração, Inteligência e Informação de Governo ($I^3 - Gov$): hospedagem do sistema desenvolvido para integrar e oferecer aos gestores do governo federal uma visão voltada para o custeio da administração pública. Foi desenvolvida uma arquitetura referencial de integração de sistemas governamentais baseada em padrões abertos e Web Services.
- Guia de administração de ambientes em "Cluster e Grid": este documento possui um conjunto de justificativas para a adoção de tecnologias baseadas em computação distribuída pelo governo brasileiro e uma abordagem técnica e gerencial das tecnologias de Cluster de Processamento de Alto Desempenho, Cluster de Aplicação, Cluster e replicação de Banco de Dados, Cluster de Armazenamento, Grid de saco-de-tarefas, Virtualização de recursos computacionais.
- Testes, análises e prospecção tecnológica: foram realizados testes, análises e prospecções tecnológicas das tecnologias de Processamento de Alto Desempenho (MPI e PVM), Sistema de Arquivos Compartilhados (GFS, OCFS2, OCFS), Cluster de Banco de Dados (Oracle RAC, Sequoia, PgCluster, Paragres), Cluster de Aplicação (Linux Virtual Server, HeartBeat, CARP), Virtualização de Recursos (VMware e Xen).

F.1 Histórico do LabCluster

O Departamento de Integração de Sistemas de Informação (DSI), da Secretaria de Logística e Tecnologia de Informação possui como atribuição definir as regras e padrões de integração do Governo Federal.

No Departamento são desenvolvidos projetos relacionados com a integração dos sistemas estruturadores do Governo Federal, integração das bases de cadastros sociais, definição de padrões abertos para interoperabilidade¹, migração para software livre² e inovações tecnológicas baseadas em tecnologias emergentes e abertas.

Tais iniciativas objetivam a transparência nas relações tecnológicas internas na Administração Pública Federal, melhoria da qualidade do serviço de Governo

¹E-Ping - Padrões de Interoperabilidade de Governo Eletrônico.

²Guia de Referência de Migração para software livre do Governo Federal

Eletrônico aos cidadãos, racionalização do uso de recursos públicos em tecnologia da informação, independência tecnológica e inserção do uso de tecnologias inovadoras no Governo Federal.

Até 2004 a Secretaria não dispunha de um laboratório para a implementação de projetos piloto, provas de conceito e prospecção Tecnológica. Esta carência de laboratório muitas vezes dificultava a realização dos projetos desenvolvidos pelo Departamento de Integração de Sistemas de Informação uma vez que o referido Departamento via-se obrigado a depender de atores externos, que nem sempre possuem a possibilidade de atender as demandas dentro do prazo exeqüível.

O primeiro laboratório piloto foi implementado com estações de trabalho do próprio ministério para atender a demanda de otimização de compras governamentais através do uso de tecnologia baseada em *data minning* para pesquisar os melhores e piores padrões de compra. O software utilizado neste projeto foi o Tamanduá³ um software de mineração de dados em Cluster desenvolvido pela UFMG com recursos de financiamento do Governo Federal através da FINEP e disponibilizado como software livre.

Este laboratório piloto era composto por um conjunto de 8 máquinas desktop interligadas em um switch 100Mbps dedicado de 12 portas e configuradas como um Cluster de processamento baseado em tecnologia PVM⁴.

Os resultados deste projeto foram muito proveitosos e a Secretaria resolveu investir na criação de um Laboratório de Inovações Tecnológicas em Cluster e Grid, para tanto foi realizada a aquisição de 32 servidores dual processados totalizando uma capacidade de 64 processadores Xeon de 2.4Ghz, 80GB de memória ram e 7.36TB de armazenamento em disco rígido.

Este laboratório foi denominado LabCluster por conta do projeto de inovações tecnológicas em Cluster e Grid que busca construir alternativas economicamente viáveis, tecnologicamente sustentáveis e inovadoras ao uso de computadores de grande porte.

³<http://tamandua.speed.dcc.ufmg.br/>

⁴Parallel Virtual Machine - Máquina paralela virtual

F.2 Missão do LabCluster

A Missão do LabCluster é prover infra-estrutura tecnológica e computacional, baseada em computação distribuída e padrões abertos de hardware e software para os Projetos internos ou em parceria com a Secretaria de Logística e Tecnologia da Informação.

O LabCluster é um ambiente de testes, prospecção e análise, onde são feitas provas de conceito, projetos piloto e não deve ser tratado como um ambiente de produção. Para a disponibilização de aplicações em produção deverá ser utilizada a infra-estrutura das empresas de TI do Governo Federal, tais como: DATAPREV e SERPRO.

F.3 Descrição do Ambiente LabCluster

Em consonância com as diretrizes de Governo Eletrônico de racionalização de recursos e utilização de Software Livre:

- Todo o hardware utilizado no laboratório é baseado em tecnologia i386 e padrões abertos.
- Toda a infra-estrutura de software básico⁵ do laboratório é em Software Livre ou aberto, para não comprometer a adoção de tecnologias inovadoras pelo governo com os custos de aquisição de licenças de software.

Exceções:

- Poderão existir aplicações específicas de projetos, que não serão Software Livre ou aberto, mas a infra-estrutura de software base será totalmente livre ou aberta

⁵Entende-se por software básico, o sistema operacional e os softwares e aplicações necessários para o funcionamento, administração e gerenciamento do Cluster.

F.4 Infra-estrutura de Hardware

A tabela F.1 apresenta resumidamente as configurações dos hardwares disponíveis no LabCluster, enquanto as seções subseqüentes apresentam o detalhamento das configurações disponíveis:

Servidores SuperMicro 1U				
Quant.	CPU	Memória	HD	Rede
16	02 x 2.4Ghz Xeon HT	02 GB	01 x IDE 80GB	02 x Gigabit
Servidores SuperMicro 2U				
Quant.	CPU	Memória	HD	Rede
08	02 x 2.4Ghz Xeon HT	04 GB	01 x IDE 80GB	10 x Gigabit
Servidor SuperMicro Gabinete				
Quant.	CPU	Memória	HD	Rede
08	02 x 2.4Ghz Xeon HT	02 GB	04 x IDE 200GB	02 x Gigabit
Desktops Novadata				
Quant.	CPU	Memória	HD	Rede
12	01 x 2.4Ghz Pentium IV	256MB	01 x IDE 40GB	01 x 100Mbps
Servidor Dell PowerEdge 1850				
Quant.	CPU	Memória	HD	Rede
08	02 x 3.6Ghz	02 GB	01 x SCSI 73GB	02 x Gigabit

Tabela F.1: Tabela de Hardware

F.5 Política de Utilização do Ambiente LabCluster

A Política de Utilização do Ambiente LabCluster é um documento criado dentro da SLTI para conduzir e propiciar uma melhor relação de trabalho dos interessados com o laboratório. Ele possui os seguintes objetivos:

- Definir qual a missão do LabCluster,
- Descrever os procedimentos de uso do ambiente LabCluster,
- Especificar a Infra-estrutura física e lógica, de hardware e software do LabCluster,
- Definir as políticas e normas de utilização do LabCluster,

- Definir as políticas e normas do ambiente de “produção” do LabCluster.

Este documento pode ser obtido em: <http://guialivre.governoeletronico.gov.br/guiaonline/downloads/guias/politica.pdf>

Apêndice G

Outros Documentos Produzidos

Em paralelo ao trabalho neste Guia, vários outros documentos foram trabalhados pela equipe da SLTI, e podem ser obtidos no sitio do Guia de Cluster: <http://guialivre.governoeletronico.gov.br/guiacluster/>.

Os documentos se situam em vários tópicos e necessidades sendo divididos da seguinte forma:

- Documentos internos:
 - Política de Utilização do Ambiente LabCluster,
 - Mineração de Dados - Tamanduá
- Documentação de Tecnologias; que vem sendo escritas de forma colaborativa no wiki do seminário de cluster e grid <http://guialivre.governoeletronico.gov.br/mediawiki/index.php/DocumentacaoTecnologias>, como por exemplo:
 - DRBD v07 - Distributed Replicated Block Device,
 - DRBD v08 - OCFS2, LVS, Heartbeat e ldirector,
 - Virtualização de Recursos - Xen,
 - Implementação de Firewall Redundante com OpenBSD, Packet Filter, PFSYNC e CARP.
- Traduções:

- Guia do Usuário Sequoia,
- Manual OpenMosix,
- How-to PGcluster,

Referências Bibliográficas

- [1] Advanced mysql replication techniques. <http://www.onlamp.com/pub/a/onlamp/2006/04/20/advanced-mysql-replication.html>.
- [2] Arquitetura e-ping. <http://www.eping.e.gov.br>.
- [3] Blast webpage. <http://www.ncbi.nlm.nih.gov/BLAST>.
- [4] Compute against the cancer site. <http://www.computeagainstcancer.org>.
- [5] The dzero experiment. <http://www-d0.fnal.gov>.
- [6] Governo eletrônico - conheça o gov.br. <http://www.governoeletronico.gov.br>.
- [7] Introducing slony. <http://www.onlamp.com/pub/a/onlamp/2004/11/18/slony.html>.
- [8] Linux virtual server. <http://www.linuxvirtualserver.org>.
- [9] Live backups of mysql using replication. <http://www.onlamp.com/pub/a/onlamp/2005/06/16/MySQLian.html>.
- [10] Lvs-mini-howto. <http://www.austintek.com/LVS/LVS-HOWTO/mini-HOWTO/>.
- [11] Modifying slony clusters. http://www.onlamp.com/pub/a/onlamp/2005/03/17/slony_changes.html.
- [12] Mygrid site. <http://www.ourgrid.org/mygrid>.
- [13] Mysql - reference manual. <http://dev.mysql.com/doc/refman/5.1/en/index.html>.

- [14] OMG - object management group. <http://www.omg.org/corba>.
- [15] Pargres. <http://pargres.nacad.ufrj.br>.
- [16] Pargres: uma camada de processamento paralelo de consultas sobre o postgresql. http://pargres.nacad.ufrj.br/Documentos/id_9830_wsl2005_pargres.pdf.
- [17] Pgcluster. <http://pgcluster.projects.postgresql.org/>.
- [18] Pgpool. <http://pgpool.projects.postgresql.org/>.
- [19] Postgresql. <http://www.postgresql.org>.
- [20] Replication in mysql. <http://www.linux-mag.com/content/view/1599/0/1/0/>.
- [21] RMI - remote method invocation specification. <http://java.sun.com/products/jdk/rmi/index.jsp>.
- [22] Seti@home site. <http://setiathome.ssl.berkeley.edu>.
- [23] Simgrid site. <http://gcl.ucsd.edu/simgrid>.
- [24] United devices site. <http://www.ud.com>.
- [25] ISO New England: Electricity trading over the internet begins in six new england states. Business Wire - <http://industry.java.sun.com/javanews/stories/story2/0,1072,15093,00.html>, May 1999.
- [26] The evolution of UDDI: UDDI.org white paper. http://www.uddi.org/pubs/the_evolution_of_uddi_20020719.pdf, 2002.
- [27] UDDI: Universal description, discovery and integration of web services. <http://www.uddi.org>, 2002.
- [28] Business process execution language for web services version 1.1. <http://www-128.ibm.com/developerworks/library/specification/ws-bpel>, May 2003.
- [29] Seti@home client program remote buffer overflow vulnerability. <http://www.securityfocus.com/bid/7292/info/>, April 2003.
- [30] Entropia web page. <http://www.entropia.com>, 2004.

- [31] Mygrid online manual. <http://www.ourgrid.org>, 2004.
- [32] Gnutella. <http://www.gnutella.com>, 2005.
- [33] Grid physics network. <http://www.griphyn.org/>, 2005.
- [34] Growing animated film talent. <http://www.hpl.hp.com/SE3D/se3d-overview.html>, 2005.
- [35] Omg's corba website. <http://www.corba.org/>, 2005.
- [36] Ourgrid project. <http://www.ourgrid.org>, 2005.
- [37] SETI@home top users. <http://setiathome2.ssl.berkeley.edu/stats/users.html>, March 2005.
- [38] Teragrid. <http://www.teragrid.org/>, 2005.
- [39] Web services activity. <http://www.w3.org/2002/ws/>, 2005.
- [40] Ws - resource framework. <http://www.globus.org/wsrf/>, 2005.
- [41] Josh Aas. Understanding the linux 2.6.8.1 cpu scheduler. http://josh.trancesoftware.com/linux/linux_cpu_scheduler.pdf. Ultima Visita em 20.09.2005 12:12.
- [42] MySQL AB. Mysql manual. <http://www.mysql.com/doc/en/index.html>. Ultima Visita em 20.09.2004 12:20.
- [43] D. Abramson, J. Giddy, I. Foster, and L. Kotler. High performance parametric modeling with nimrod/G: Killer application for the global grid? In *IPDPS*, pages 520–528, 2000.
- [44] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th OSDI*, December 2002.
- [45] C. J. AGHA, G; CALLSEN. *ActorSpace: An Open Distributed Programming Paradigm*. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993.
- [46] G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. Mit Press, 1986.

- [47] G. Actons AGHA. *MIT Press, Cambridge, MA. A Model of Concurrent Computation in Distributed Systems*, 1986.
- [48] Marcos Aguilera, Ramaprabhu Janakiraman, and Lihao Xu. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN 2005)*, Yokohama, Japan, June-July 2005.
- [49] Hassan AIT-KACI. *The WAM: A (Real) Tutorial*. Paris: Digital Equipment Corporation, Research Laboratory, 1990.
- [50] et. al. Al Geist. *PVM: Parallel Virtual Machine: A Users; Guide and Tutorial for Network Parallel Computing (Scientific and Engineering Computation)*. The Mit Press, Cambridge, Massachussets.
- [51] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tueck. Gridftp protocol specification. GGF GridFTP Working Group Document, September 2002.
- [52] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, and C. Salisbury S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187-200, 2001.
- [53] Globus Alliance. Ogsa site. <http://www.globus.org/ogsa>, 2003.
- [54] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 1(215):403–410, 1990.
- [55] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.
- [56] Ahmed Amer and Amr El-Kadi. Beating bottlenecks in the design of distributed file systems. Dez 1996.
- [57] T. Bray and J. Paoli and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C, 1.1 edition, October 2000. <http://www.w3c.org/TR/REC-xml>.
- [58] Carl Anderson and John J. Bartholdi III. Centralized versus decentralized control in manufacturing: Lessons from social insects. In I. P. McCarthy

- and T. Rakotobe-Joel, editors, *Complexity and Complex Systems in Industry*, pages 92–105, September 2000.
- [59] D. Anderson, J. Cobb, and E. Korpela. SETI@home: An experiment in public-resource computing. *Communication of the ACM*, 45 (11):56–61, 2002.
- [60] N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray. Discouraging free-riding in a peer-to-peer grid. In *HPDC13, the Thirteenth IEEE International Symposium on High-Performance Distributed Computing*, June 2004.
- [61] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, 2003.
- [62] Nazareno Andrade, Walfredo Cirne, Francisco Brasileiro, and Paulo Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. *9th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003.
- [63] Gregory ANDREWS. Synchronizing resources. *ACM Transactions on Programming Languages and Systems*, 3(4):405–430, october 1981.
- [64] Gregory ANDREWS. The distributed programming language sr - mechanisms, design and implementation. *Software-Practice and Experience*, 12(8):719–753, august 1982.
- [65] Gregory R. Andrews. Synchronising resources. *ACM Transactions on Programming Languages Andsystems*, 3(4):405–430, oct 1981.
- [66] Ronald ANDREWS, Gregory.; OLSSON. The evolution of the sr language. *Distributed Computing*, 1(3):133–149, july 1986.
- [67] Ronald ANDREWS, Gregory; OLSSON. An overview of the sr language and implementation. *CM Transactions on Programming Languages and Systems*, 10(1):51–86, january 1988.
- [68] Ronald A. ANDREWS, Gregory R.; OLSSON. *The SR Programming Language*. The Benjamin/Cummings Publishing Company, 1992.
- [69] R. N. Anthony. Planing and control systems: A framework for analysis. Technical report, Havard University, Graduate Schoole of Business Administration, 1965.

- [70] Eliane Araújo, Walfredo Cirne, Gustavo Wagner, and Nigini Oliveira. The seghidro experience: Using the grid to empower a hydro meteorological scientific network.
- [71] Bradley BAKER, Louis; SMITH. *Parallel Programming*. McGraw-Hill, Inc, 1996.
- [72] K. R. Baker. Requirements planning. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Handbooks in Operations Research and Management Science - Logistics of Production and Inventory*, volume 4, chapter Chapter 11. North-Holland, 1993.
- [73] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. The Grid: International efforts in Grid Computing, 2000.
- [74] Jennifer G. BAL, Henri E.; STEINER. Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3):261–322, september 1989.
- [75] F. Banaei-Kashani, C. Chen, and C. Shahabi. Wspds: Web services peer-to-peer discovery dervice. In *Proc. of International Symposium on Web Services and Applications (ISWS'04)*, 2004.
- [76] A. Baratloo, P. Dasgupta, V. Karamcheti, and Zvi M. Kedem. Metacomputing with MILAN. In *Heterogeneous Computing Workshop*, pages 169–183, 1999.
- [77] A. Baratloo, P. Dasgupta, and Z. M. Kedem. CALYPSO: A novel software system for fault-tolerant parallel processing on distributed platforms. In *Proc. of the Fourth IEEE International Symp. on High Performance Distributed Computing (HPDC-4)*, pages 122–129, 1995.
- [78] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the web. In *Proc. of the 9th International Conference on Parallel and Distributed Computing Systems (PDCS-96)*, 1996.
- [79] A. C. Barbosa, J. Sauvé, W. Cirne, and M. Carelli. Independently auditing service level agreements in the grid. In *Proceedings of the 11th HP OpenView University Association Workshop (HPOVUA 2004)*, 2004.
- [80] Jorge L. V BARBOSA. *Princípios do Holoparadigma*. CPGCC da UFRGS, 1999.

- [81] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.
- [82] Alexander Barmouta and Rajkumar Buyya. GridBank: A Grid Accounting Services Architecture (GASA) for distributed systems sharing and integration, 2003 (submitted).
- [83] J. Bartholdi and D. Eisenstein. Bucket brigades. http://www.isye.gatech.edu/people/faculty/John_Bartholdi/bucket\discretionary{-}{ }{}brigades.html, 2002.
- [84] J. Basney, M. Livny, and P. Mazzanti. Harnessing the capacity of computational grids for high energy physics. In *Conference on Computing in High Energy and Nuclear Physics*, 2000.
- [85] Jim Basney and Miron Livny. *Deploying a High Throughput Computing Cluster*, volume 1, chapter High Performance Cluster Computing. Prentice Hall, may 1999.
- [86] O. Beaumont, L. Carter, J. Ferrante, and Y. Robert. Bandwidth-centric allocation of independent task on heterogeneous plataforms. In *Proceedings of the Internetonial Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, April 2002.
- [87] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.
- [88] F. Berman, A. Hey, and G. Fox, editors. *Grid Computing - Making the Global Infrastructure a Reality*. John Wiley and Sons, Ltd, 2003.
- [89] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing'96*, 1996.
- [90] Francine Berman and Richard Wolski. Scheduling from the perspective of the application. In *HPDC*, pages 100–111, 1996.
- [91] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

- [92] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. Gass: A data movement and access service for wide area computing systems. In *Sixth Workshop on I/O in Parallel and Distributed Systems*, May 1999.
- [93] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.
- [94] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pages 34–43, 2000.
- [95] G. Booch. *Object Oriented Design*. The Benjamin/Cummings Publishing Company, Inc, 1st edition, 1991.
- [96] M. BRIAT, J.; FAVRE. . In Computer Science, editor, *Parallel Architectures and Languages Europe*, volume 506, chapter Scheduling of OR-parallel Prolog on a scalable reconfigurable distributed memory multiprocessor. Springer-Verlang, 1991.
- [97] Rachid BRIOT, Jean-Pierre; GUERRAOUI. Concurrency and distribution in object-oriented programming. *ACM Computing Surveys*, 30(3):291–329, september 1998.
- [98] R. Buyya, D. Abramson, and J. Giddy. An economy driven resource management architecture for computational power grids. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2000.
- [99] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service-oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, 2001.
- [100] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, Maio 2002.
- [101] Rajkumar. BUYYA. *High Performance Cluster Computing, Architectures and Systems*. Prentice Hall, 1999.
- [102] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/g: An architecture of a resource management and scheduling system in a global

- computational grid. In *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China, 2000.
- [103] Rajkumar Buyya and Sudharshan Vazhkudai. Compute Power Market: Towards a Market-Oriented Grid. In *The First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Beijing, China, 2000. IEEE Computer Society Press.
- [104] Toni BUYYA, Rajkumar; CORTES. Single system image, special issue. In Sage Science Press, editor, *International Journal of High Performance*, volume Volume 15, chapter Pages: 124, 135. Sage Science Press, 2001.
- [105] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. Pvfs: A parallel virtual file system for linux clusters. <http://www.parl.clemson.edu/pvfs/el2000/extreme2000.ps>. Ultima Visita em 20.09.2005 12:12.
- [106] David CARRIERO, Nicholas; GELERNTER. Data paralelismo and linda. Technical report, International Workshop on Languages and Compilers for Parallel Computing, 1992.
- [107] N. Carriero and D. Gelernter. How to write parallel programs: a guide to the perplexed. *Communications of the ACM*, 21, 1989.
- [108] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane Australia, May 2001.
- [109] H. Casanova and F. Berman. *Grid Computing: making the Global Infrastructure a Reality*, chapter Parameter Sweep on the Grid with APST. John Wiley and Sons, 2003.
- [110] H. Casanova, J. Hayes, and Y. Yang. Algorithms and software to schedule and deploy independent tasks in grid environments. In *Proceedings of Workshop on Distributed Computing/Metacomputing and Resource Globalization*, 2002.
- [111] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 349–363, Cancun, Mexico, May 2000. IEEE Computer Society Press.

- [112] H. Casanova and L. Marchal. A network model for simulation of grid application. *Research Report N° 2002-40*, October 2002.
- [113] Henri Casanova, Graziano Obertelli, Francine Berman, and Rich wolski. The apples parameter sweep template: User-level middleware for the grid. In *Supercomputing Conference (SC'2000)*, 2000.
- [114] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel Distributed Computing*, 63:597–610, 2003.
- [115] W. Cirne. Grids computacionais: Arquiteturas, tecnologias e aplicações. In *Anais do Terceiro Workshop em Sistemas Computacionais de Alto Desempenho*, Vitória, Espírito Santo, Brasil, October 2002.
- [116] W. Cirne and F. Berman. Using moldability to improve the performance of supercomputer jobs. *Parallel and Distributed Computing*, 62(10):1571–1601, 2002.
- [117] W. Cirne, F. Brasileiro, L. Costa, D. Paranhos, E. Santos-Neto, N. Andrade, C. De Rose, T. Ferreto, M. Mowbray, R. Scheer, and J. Jornada. Scheduling in bag-of-task grids: The pauÁ case. In *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2004)*, October 2004.
- [118] W. Cirne and K. Marzullo. The computational coop: Gathering clusters into a metacomputer. In *PPS/SPDP'99 Symposium*, 1999.
- [119] W. Cirne and K. Marzullo. Open Grid: A user-centric approach for grid computing. In *Proceedings of the 13th Symposium on Computer Architecture and High Performance Computing*, 2001.
- [120] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauv  , F. Alves B. da Silva, C. O. Barros, and C. Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. In *Proceedings of the ICCP'2003 - International Conference on Parallel Processing*, October 2003.
- [121] L. et al Clarke. The mpi message passing interface standard. Technical report, Knoxville: University of Tennessee, 1994.
- [122] Inc. Cluster Resources. Maui cluster scheduler. <http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php>. Ultima Visita em 20.04.2006 12:20.

- [123] Inc. Cluster Resources. Torque overview. <http://www.clusterresources.com/pages/products/torque-resource-manager.php>. Ultima Visita em 20.04.2006 12:20.
- [124] Bram Cohen. Incentives build robustness in bittorrent. <http://www.bittorrent.com/bittorrentecon.pdf>, 2004.
- [125] M. C. Coint. *El Manual para el Clustering con OpenMosix*. miKeL a.k.a.mc2, GNU Free Documentation Licence, 2004.
- [126] P. W. A. Costa. Como surgiram os data warehouses? *Computerworld*, novembro:16, 1997.
- [127] F. P. Coyle. *Web Services, and the Data Revolution*. Addison-Wesley Information Technology Series, 2002.
- [128] D. E. CULLER and J.P. SINGH. *Parallel Computer Architecture: a hardware and software approach*. Morgan Kaufmann, 1999.
- [129] David et al. CULLER. *LogP: Toward a Realistic Model of Parallel Computation*. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993.
- [130] f. Culler. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, San Francisco, CA, 1998.
- [131] F. Curbera, Y. Golland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.0. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2002.
- [132] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maquire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution. Global Grid Forum Draft Recommendation, May 2004.
- [133] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.
- [134] Gustavo da Gama Torres. A empresa pública de informática e informação: Modelo de gestão e papel. *Revista IP*, 2(1), Maio 2000.

- [135] Programa Sociedade da Informação (SocInfo). Sociedade da informação no brasil - livro verde. http://ftp.mct.gov.br/Livro_Verde/Default3.htm. Última Visita em 11.09.2006 12:20.
- [136] Mario Dantas. *Computação Distribuída de Alto Desempenho*. Axcel Books, 2005.
- [137] Daniel Darlen. Utilização de ferramenta de mineração de dados em ambiente cluster. <http://guialivre.governoeletronico.gov.br/labcluster/tamandua.pdf>. Última Visita em 11.09.2006 12:20.
- [138] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, Reading, MA, 6 edition, 1995.
- [139] T. Davis, A. Chalmers, and H. W. Jensen. Practical parallel processing for realistic rendering. In *ACM SIGGRAPH*, 2000.
- [140] Escola Regional de Alto Desempenho. *Caderno dos Cursos Permanente*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2006.
- [141] Escola Regional de Alto Desempenho. *Primeira Escola Regional de Alto Desempenho*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2001.
- [142] Escola Regional de Alto Desempenho. *Segunda Escola Regional de Alto Desempenho - São Leopoldo - RS*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2002.
- [143] Escola Regional de Alto Desempenho. *Terceira Escola Regional de Alto Desempenho - Santa Maria - RS*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2003.
- [144] Escola Regional de Alto Desempenho. *Quarta Escola Regional de Alto Desempenho - Pelotas -RS*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2004.
- [145] Escola Regional de Alto Desempenho. *Quinta Escola Regional de Alto Desempenho - Canoas - RS*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2005.

- [146] Escola Regional de Alto Desempenho. *Sexta Escola Regional de Alto Desempenho - Ijuí - RS*. Comissão Regional de Alto Desempenho - Regional do Rio Grande do Sul, Sociedade brasileira de Computação, 2006.
- [147] C. DE ROSE, BLANCO, F. MAILLARD, N. SAIKOSKI, K. NOVAES, R. RICHARD, and O. RICHARD. The virtual cluster: a dynamic environment for exploitation of idle network resources. *14th symposium on Computer Architecture and High-Performance Computing (SBAC-PAD 2002)*. USA: IEEE Computer Society, pages p.141 – 148, 2002.
- [148] Doug DEGROOT. Restricted and-parallelism. Technical report, INTERNATIONAL CONFERENCE ON FIFTH GENERATION COMPUTER SYSTEMS, 1984.
- [149] Jay L. Devore. *Probability and Statistics for Engineering and The Sciences*, volume 1. John Wiley and Sons, Inc., 2000.
- [150] Peter Dibble, Michael Scott, and Carla Ellis. Bridge: A high-performance file system for parallel processors. http://www.cs.rochester.edu/u/scott/papers/1988_ICDCS_Bridge.pdf. Última Visita em 20.12.2005 12:12.
- [151] Ministério do Planejamento. Guia livre - referência de migração para software livre do governo federal. <http://www.governoeletronico.gov.br>. Última Visita em 11.09.2006 12:20.
- [152] Ministério do Planejamento. Política de utilização do labcluster. <http://guialivre.governoeletronico.gov.br/labcluster/politica.pdf>. Última Visita em 11.09.2006 12:20.
- [153] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of 11th International Parallel Processing Symposium (IPPS'97)*, April 1997.
- [154] R. Dragan. The meaning of moore's law. *PC Magazine Online*, Online on February 14th 2003. <http://www.pcmag.com/article2/0,4149,4092,00.asp>.
- [155] DRBD. Drbd. <http://www.drbd.org>. Última Visita em 20.04.2005 12:20.
- [156] R. Durbin, S. Eddy, A. Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

- [157] Andrea C. Dusseau, David E. Culler, Klaus E. Schauser, and Richard P. Martin. Fast parallel sorting under logp: Experience with the cm-5. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):791–805, 1996.
- [158] César A. F. De Rose e Philippe O. A. Navaux. *Arquiteturas Paralelas*. Instituto de Informática da UFRGS, série livros didáticos - número 15 edition.
- [159] Renato Silveira Eduardo Rodrigues Cerejo, Fabrício Correia Sales. Exemplo de arquitetura mimd - clusters. Technical report, Projeto final de curso do INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO - USP, 2005.
- [160] M. Eisler. Nfs version 2 and version 3 security issues and the nfs protocol's use of rpcsec gss and kerberos v5. <http://rfc-ref.org/RFC-TEXTS/2623/chapter2.html>. Ultima Visita em 20.12.2005 12:12.
- [161] Ted. EL-REWINI, Hesham; LEWIS. *Distributed and Parallel Computing*. Manning Publications Co, 1998.
- [162] W. R. Elwasif, J. S. Plank, and R. Wolski. Data staging effects in wide area task farming applications. In *IEEE International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, May 2001. IEEE Computer Society Press.
- [163] enbd. Ultima Visita em 20.04.2005 12:20.
- [164] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: load sharing among workstation clusters. *Journal on Future Generations of Computer Systems*, 12(1), 1996.
- [165] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12:53–65, 1996.
- [166] Geist et al. *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.
- [167] Huican Zhu et al. Adaptive load sharing for clustered digital library servers. In *Proceedings of 7th IEEE International Symposium on High Performance Distributed Computing*, July 1998.
- [168] W. Andrews et al. Predicts 2005: The impact of web services still grows. In *Gartner Research Note (G00123895)*, November 2004.

- [169] ExeCRaBLE. Prozessor history. <http://www.execrable.de/hardware/history.html>. Ultima Visita em 20.09.2005 12:12.
- [170] M. Faerman, R. Wolski A. Su, and Francine Berman. Adaptive performance prediction for distributed data-intensive applications. In *Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing*, Portland, OH, USA, 1999. ACM Press.
- [171] FarSite. <http://research.microsoft.com/sn/Farsite>, 2005.
- [172] D. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459, pages 1–24. Lecture Notes in Computer Science, Springer-Verlag, 1998.
- [173] D. G. Feitelson. Metric and workload effects on computer systems evaluation. *Computer*, 36(9):18–25, September 2003.
- [174] Dror Feitelson. Parallel workload archive.
- [175] Ciro Campos Christo Fernandes. A reforma administrativa no brasil: oito anos de implementação do plano diretor - 1995-2002, Out 2002.
- [176] A. B. H. Ferreira. *Novo Dicionário Aurélio da Língua Portuguesa*. Editora Nova Fronteira, 1986.
- [177] K.W. Flynn, M.J. e Rudd. Parallel architectures. *ACM Computing Surveys*, 28(1):67–70, 1996.
- [178] Message Passing Interface Forum. *MPI: A Message Passing Interface standard*. Message Passing Interface Forum, 1997.
- [179] I. Foster. *Designing and building parallel programs: Concepts and tools for parallel software engineering*. Addison-Wesley, 1995.
- [180] I. Foster. What is the grid? a three point checklist. *GRID today*, 1(6), July 2002.
- [181] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [182] I. Foster and C. Kesselman. The globus project: A status report. In *Proceedings of IPPS/SPDP Heterogeneous Computing Workshop*, pages 4–18, 1998.

- [183] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 1999.
- [184] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for distributed systems integration. Global Grid Forum - GGF, 2002.
- [185] I. Foster, C. Kesselman, and S. Tuecke. The nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.
- [186] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.
- [187] The Apache Software Foundation. Apache jakarta tomcat. <http://tomcat.apache.org/tomcat-5.0-doc>. Ultima Visita em 20.01.2006 12:20.
- [188] The Apache Software Foundation. Apache tomcat. <http://tomcat.apache.org/>. Ultima Visita em 20.01.2006 12:20.
- [189] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, and Y. Jim. An architecture for a global internet host distance estimation service. In *Proceedings of IEEE INFOCOM*, 1999.
- [190] FRESCO. Foundation for research on service composition. <http://www.servicecomposition.org/>, 2005.
- [191] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: a computation management agent for multi-institutional grids. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium*, pages 55 – 63, San Francisco, CA, USA, August 2001. IEEE Computer Society Press.
- [192] Doreen L. Galli. *Distributed Operating Systems*. Prentice Hall, 2000.
- [193] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Pub Co., 1995.
- [194] Elizabeth Garbett, Andrew Scheferman, and Albert Tse. Virtual disk - it's not just for mainframes anymore. <http://www.storagetek.com/>. Ultima Visita em 20.09.2005 12:12.

- [195] Cláudio F.R GEYER. Une contribution a l'étude du parallelisme ou en prolog sur des machines sans mémoire commune. Technical report, Grenoble: Université Joseph Fourier, 1991.
- [196] GGF. Global grid forum. <http://www.ggf.org>, November 2003.
- [197] Sanjay Ghemawat, Howard Gobio, and Shun-Tak Leung. The google file system. <http://www.cs.rochester.edu/sosp2003/papers/pl25-ghemawat.pdf>. Ultima Visita em 20.09.2005 12:12.
- [198] R. Gibbons. A historical application profiler for use by parallel schedulers. *Lecture Notes in Computer Science*, 1291:58–77, 1997.
- [199] Dan Gisolfi. Is web services the reincarnation of CORBA? <http://www-106.ibm.com/developerworks/webservices/library/ws-arc3/>.
- [200] J. et al GOGUEN. *An Introduction to OBJ3*. Springer-Verlag, 1995.
- [201] TI & Governo. O serpro inicia os testes para abandonar o mainframe e busca soluções em software livre. http://www.serpro.gov.br/noticiasSERPRO/20060829_02/. TI & Governo, nº 170, 29 de agosto de 2006.
- [202] Grid3. Grid3: An application grid laboratory for science. <http://www.ivdgl.org/grid2003/>, 2005.
- [203] Gridbus. The gridbus project. <http://www.gridbus.org/>, 2005.
- [204] Andrew S. Grimshaw, Wm. A. Wulf, and The Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.
- [205] W. Lusk Gropp. *Using MPI: Portable Parallel Programming with the Message Passing-Interface*. MIT Press, 1994.
- [206] Apache Group. The apache xml project. <http://xml.apache.org>. Ultima Visita em 20.01.2005 12:20.
- [207] GriPhyN Group. <http://www.GriPhyN.org>, 2002.
- [208] JBoss Group. Jboss group :: Professional open source. <http://www.jboss.org>. Ultima Visita em 20.09.2004 12:20.

- [209] Ibrahim F. Haddad. Pvfs: A parallel virtual file system for linux clusters. <http://www.linuxjournal.com/article/4354>. Ultima Visita em 20.09.2005 12:12.
- [210] Michael HANUS. The integration of functions into logic programming from theory to practice. *Journal of Logic Programming*, 19/20(1):583–628, may/july 1994.
- [211] S. Hastings, T. Kurc, S. Lamgella, U. Catalyurek, T. Pan, and J. Saltz. Image processing for the grid: A toolkit for building grid-enabled image processing applications. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003.
- [212] A. Hefez. *Curso de Álgebra*, volume 1. IMPA, 1993.
- [213] F HERMENEGILDO, M.; ROSSI. *Prolog and its Performance: Exploiting Independent And-Parallelism*. MIT Press, 1990.
- [214] hilip H. Carns, Robert B. Ross, Walter B. Ligon III, and Pete Wycko. Bmi: A network abstraction layer for parallel i/o. <http://www.osc.edu/~pw/papers/carns-bmi-ipdps05.pdf>. Ultima Visita em 20.12.2005 12:12.
- [215] Karen Epper Hoffman. Ending the grid lock. <http://www.technologyreview.com/>, March 2005.
- [216] T. Hogg and B. A. Huberman. Controlling chaos in distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, 21:1325–1332, 1991.
- [217] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. <http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/s11.pdf>. Ultima Visita em 20.09.2005 12:12.
- [218] HPF. High performance fortran. <http://www.crpc.rice.edu/HPFF/home.html>, December 2003.
- [219] J. HUDAK, P.; FASEL. *A Gentle Introduction to Haskell*. ACM SIGPLAN Notices, 1992.
- [220] M. Humphrey, G. Wasson, M. Morgan, and N. Beekwilder. An early evaluation of wsrp and ws-notification via wsrp.net. In *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004.

- [221] k. Hwang. *Advanced computer architecture: parallelism, scalability, programmability*. McGraw-hill, New York, NY, 1993.
- [222] Kai HWANG. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, Inc, 1993.
- [223] Miguel Catalán i Cort. *El Manual para el Clustering con OpenMosix*. TLDP, 2004.
- [224] Adriana Iamnitchi and Ian Foster. A peer-to-peer approach to resource location in grid environments. *Grid Resource Management*, 2003.
- [225] Adriana Iamnitchi, Matei Ripeanu, and Ian Foster. Small-world file-sharing communities, March 2004.
- [226] Oscar H. Ibarra and Chul E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.
- [227] IBM. System management guide: Communications and networks. http://publib16.boulder.ibm.com/pseries/en_US/aixbman/commadmn/nfs_netlock.htm. Ultima Visita em 20.09.2005 12:12.
- [228] IEC. International electrotechnical commission. <http://www.iec.ch/>. Ultima Visita em 20.04.2005 12:20.
- [229] Virgílio José Martins IGNACIO, Aníbal Alberto Vilcapona y FERREIRA FILHO. Mpi: uma ferramenta para implementação paralela. *Pesquisa Operacional*, 22(1):105–116, junho 2002.
- [230] Red Hat Inc. Gfs. <http://www.redhat.com>. Ultima Visita em 20.04.2005 12:20.
- [231] Sun Microsystems Inc. Rpc: Remote procedure call protocol specification version 2. <http://rfc-ref.org/RFC-TEXTS/1057/>. Ultima Visita em 20.09.2005 12:12.
- [232] Tech Insider. An interactive look at how ethernet has evolved. <http://www.networkworld.com/techinsider/2002/1014ethernettime.html>. Ultima Visita em 20.09.2005 12:12.
- [233] ISO. International organization for standardization. <http://www.iso.org>. Ultima Visita em 20.04.2005 12:20.

- [234] Lauro Ivo. Grid scheduling: Adapting space-shared resources to eager schedulers. In *HP-OurGrid-Technical Report*, 2004.
- [235] B. Kemme J. M. Milan-Franco. Adaptative middleware for data replication.
- [236] R. Jain. *The Art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling*, volume 1. John Wiley and Sons, Inc., 1991.
- [237] M.-C. Jih, Li-Chi Feng, and Ruei-Chuan Chang. The design and implementation of the pasda parallel file system. In *International Conference on Parallel and Distributed Systems*, chapter pages 142 - 147. 1994.
- [238] Paul JONES, Mark P.; HUDAK. Implicit and explicit parallel programming in haskell. nebulasystems.cs.yale.edu/pub/yale-fp/reports/RR-982.ps.Z. julho de 1999.
- [239] T. Jones, A. Koniges, and R. K. Yates. Performance of the ibm general parallel file system. <http://www.llnl.gov/icc/lc/siop/papers/GPFSPerformance.doc>. Ultima Visita em 20.09.2005 12:12.
- [240] Zhiwei Xu K. Hwang. *Scalable Parallel Computing*. McGraw-Hill, 2000.
- [241] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *Proceedings of 2nd International System Administration and Networking Conference*, May 2000.
- [242] Subramanian Kannan, Mark Roberts, Peter Mayes, Dave Brelsford, and Joseph F Skovira. Workload management with loadleveler. <http://www.redbooks.ibm.com/redbooks>, November 2001.
- [243] Z. M. Kedem, K. V. Palem, and P. G. Spirakis. Efficient robust parallel computations (extended abstract). In *ACM Symposium on Theory of Computing*, pages 138–148, 1990.
- [244] Philippe KERGOMMEAUX, Jacques Chassin; CODOGNET. Parallel logic programming systems. *ACM Computing Surveys*, 26(3):295–336, september 1994.
- [245] Fabio Kon. Distributed file systems past, present and future a distributed file system for 2006. <http://choices.cs.uiuc.edu/~f-kon/DFSPaper.ps.gz>. Ultima Visita em 20.09.2005 12:12.

- [246] Fabio Kon. Sistemas de arquivos distribuídos. <http://choices.cs.uiuc.edu/~f-kon/thesis/kon-master.ps.gz>. Última Visita em 20.09.2005 12:12.
- [247] Charles M. Kozierok. Hard disk drives. <http://www.pcguide.com/ref/hdd/>. Última Visita em 20.09.2005 12:12.
- [248] Charles M. Kozierok. The processor. <http://www.pcguide.com/ref/cpu/>. Última Visita em 20.09.2005 12:12.
- [249] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrant. Autonomous protocols for bandwidth-centric scheduling of independent-task applications. In *17th International Parallel and Distributed Processing Symposium*, Nice, France, April 2003.
- [250] Heather Kreger. Web services conceptual architecture. <http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, 2003.
- [251] J. Kubiawicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. IEEE Computer Society Press, November 2000.
- [252] The Olson Laboratory. Fight aids@home. <http://fightaidsathome.scripps.edu>, 2003.
- [253] U. et al. LECHNER. An object-oriented airport: Specification and refinement in maude. In Computer Science, editor, *Recent Trends in Data Type Specifications*, volume 906, chapter 351-367. Springer-Verlag, 1995.
- [254] C. Lee and M. Handi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.
- [255] F. Leymann. Web services flow language, version 1.0, 2001.
- [256] Especial Cosmo On Line. Declaração de ir pela internet bate recorde. <http://www.cosmo.com.br/especial/impostoderenda/integra.asp?id=149890>. Última Visita em 11.09.2006 12:20.

- [257] M. Litzkow, M. Livny, and M. Mutka. Condor: A hunter of idle workstations. In *Proceedings of 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.
- [258] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
- [259] V. Lo, J. Mache, and K. Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459, pages 25–46. Lecture Notes in Computer Science, Springer Verlag, 1998.
- [260] Olivier Lobry. Evaluation des systèmes de fichiers pvfs et nfsp. http://www-id.imag.fr/Laboratoire/Membres/Lobry_Olivier/PVFS_NFSP/PVFS_NFSP.html. Ultima Visita em 20.09.2005 12:12.
- [261] Pierre Lombard and Yves Denneulin. Nfsp: A distributed nfs server for cluster of workstations. <http://ka-tools.sourceforge.net/publications/nfsp-ipdps01.pdf>. Ultima Visita em 20.09.2005 12:12.
- [262] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, July 1998.
- [263] Peter Lyman, Hal R. Varian, James Dunn, Aleksey Strygin, and Kirsten Swearingen. How much information? <http://www.sims.berkeley.edu/research/projects/how-much-info-2003>, October 2003.
- [264] S. Machiraju, M. Seshadri, and D. Geels. Introspective prefetching for mobile users in oceanstore, 2002.
- [265] M. Maheswaran, S. Ali, H. J. Siegel, D. H., and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.
- [266] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *8th Heterogeneous Computing Workshop*, pages 30–45, San Juan, Puerto Rico, April 1999.

- [267] K. Marzullo, M. Ogg, A. Ricciardi and A. Amoroso, A. Calkins, and E. Rothfus. Nile: Wide-area computing for high energy physics. In *Proceedings 7th ACM European Operating Systems Principles Conference. System Support for Worldwide Applications*, pages 54–59, Connemara, Ireland, September 1996. ACM Press.
- [268] Marta Mattoso, Geraldo Zimbrão, Alexandre A. B. Lima, and Fernanda Baião. Pargres: Middleware para processamento paralelo de consultas olap em clusters de banco de dados.
- [269] Tom McNeal and Chuck Lever. Linux nfs faq. <http://nfs.sourceforge.net>. Última Visita em 20.09.2005 12:12.
- [270] Corinto Meffe. Software público é diferencial para o brasil. http://computerworld.uol.com.br/governo/corinto_meffe/idgcoluna.2006-06-09.2677471526/IDGColunaPrint_view. Última Visita em 11.09.2006 12:20.
- [271] Corinto Meffe, Carlos Castro, Anderson Peterle, Nazaré Bretas, and Rogério Santanna. Materialização do conceito de software público: Iniciativa cacic. http://guialivre.governoeletronico.gov.br/cacic/sisp2/info/software_publico.html. Última Visita em 11.09.2006 12:20.
- [272] Corinto Meffe, Elias O. P. Mussi, Leonardo Mello, and Rogério Santanna dos Santos. A tecnologia de cluster e grid na resolução de problemas de governo eletrônico. *The 18th International Symposium on Computer Architecture and High Performance Computing*, pages 1–8, Outubro 2006.
- [273] P MEHROTRA. Data parallel programming: The promises and limitations of high performance fortran. In Computer Science, editor, *Computer Science*, volume 734, chapter 1. Springer-Verlag, 1993.
- [274] Ethan L. Miller and Randy H. Katz. Rama: A file system for massively-parallel computers. <http://ssrc.cse.ucsc.edu/~elm/Papers/mss93.pdf>. Última Visita em 20.09.2005 12:12.
- [275] Ethan L. Miller and Randy H. Katz. Rama: An easy-to-use, high-performance parallel file system. <http://ssrc.cse.ucsc.edu/~elm/Papers/pc97.pdf>. Última Visita em 20.09.2005 12:12.
- [276] V. MILUINOVIC. Scanning the issue: Special issue on distributed shared memory systems. *Proceedings of the IEEE*, 87(3):1, March 1999.

- [277] Dan I MOLDOVAN. *Parallel Processing From Applications to Systems*. Morgan Kaufmann Publishers, 1993.
- [278] T. N. Moretti, C. O.; Bittencourt. Aspectos gerais da computação paralela e do sistema pvm. Technical report, Escola Politécnica da Universidade de São Paulo, Boletim Técnico, BT/PEF-9709, ISSN 0103-9822, 1997.
- [279] R. S. Morrison. *Cluster Computing Theory*. Richard S. Morrison, GNU General Public Licence, 2003.
- [280] H. Stephen MORSE. *Practical Parallel Computing*. Academic Press, Inc, 1994.
- [281] M. V MUTHUKUMAR, K.; HERMENEGILDO. Complete and efficient methods for supporting side-effects in independent/restricted and-parallelism. In *INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING*, 1989.
- [282] M. V MUTHUKUMAR, K.; HERMENEGILDO. Determination of variable dependence information through abstract interpretation. In *NORTH AMERICAN CONFERENCE ON LOGIC PROGRAMMING*, 1989.
- [283] Myricom. News & events archive myricom. <http://www.myricom.com/news/>. Ultima Visita em 20.09.2005 12:12.
- [284] Myricom. Performance measurements. <http://www.myricom.com/myrinet/performance/index.html>. Ultima Visita em 20.09.2005 12:12.
- [285] M. Nelson, B. Welch, and J. Ousterhout. Caching in the sprite network file system. 1987.
- [286] Zsolt Nemeth and Vaidy Sunderam. A formal framework for defining grid systems. In *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society Press, Maio 2002.
- [287] NeSC. National e-science centre. <http://www.nesc.ac.uk/>, 2005.
- [288] Tsuen-Wan 'Johnny' Ngan, Animesh Nandi, and Atul Singh. Fair bandwidth and storage sharing in peer-to-peer networks. In *Proceedings of*, Jan 2003.
- [289] N. Nieuwejaar, D. kootz, A. Purakayastha, C. S. Ellis, and M. L. Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, october 1996.

- [290] R. Novaes, P. Roisenberg, R. Scheer, C. Northfleet, J. Jornada, and W. Cirne. Non-dedicated distributed environment: A solution for safe and continuous exploitation of idle cycles. In *Proceedings of the AGridM 2003 - Workshop on Adaptive Grid Middleware*, 2003.
- [291] R. Oldfield. Summary of existing and developing data grids - draft. In *Grid Forum. Remote Data Access Working Group*. IEEE Computer Society Press, March 2001.
- [292] OpenMP. Simples, portable, scalable smp programming. <http://www.openmp.org>, December 2003.
- [293] C. Osthoff, P. Barros, C. Veronez, F. Agostini, W. Cirne, E. Santos-Neto, L. Costa, F. Silva, P. Pascutti, P. Bisch, and A. Silva. Utilização do software mygrid para adaptar uma aplicação de dinâmica molecular em um grid de 7 domínios. In *Technical Report LNCC*, LNCC, Rio de Janeiro, Brasil, 2002.
- [294] John Ousterhout. A brief retrospective on the sprite network operating system. [ABriefRetrospectiveontheSpriteNetworkOperatingSystem](http://www.cs.cmu.edu/~ouster/ABriefRetrospectiveontheSpriteNetworkOperatingSystem). Última Visita em 20.09.2003 12:12.
- [295] S.P. Pacheco. A user's guide to mpi. <http://nexus.cs.usfca.edu/mpi/>. Última Visita em 20.04.2005 12:20.
- [296] GIMPS Home Page. The great internet mersenne prime search. <http://www.merssene.org>, December 2003.
- [297] D. Paranhos, W. Cirne, and F. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing*, Klagenfurt, Austria, August 2003.
- [298] Simon PEYTON-JONES. *Implementing Functional Programming Languages*. International Series in Computer Science, Prentice-Hall, 1992.
- [299] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 2nd edition, New Jersey, USA, August 2001.
- [300] Jef Poskanzer. Bandwidth. <http://www.acme.com/buildapc/bandwidth.html>. Última Visita em 20.09.2003 12:12.
- [301] J. A. Pouwelse, P. Garbacki, D.H.J. Epema, and H. J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis.

- [302] Dhiraj K. Pradhan. *Fault Tolerant System Design*. Prentice Hall, 1996.
- [303] The Open MPI Project. Open mpi: Open source high performance computing. <http://www.open-mpi.org/>. Ultima Visita em 20.04.2005 12:20.
- [304] J. et al. PROTIC. *Distributed Shared Memory: Concepts and Systems*. IEEE Computer Society Press, 1998.
- [305] PVM. Pvm: Parallel virtual machine. <http://www.csm.ornl.gov/pvm/>. Ultima Visita em 20.04.2005 12:20.
- [306] Harish Ramachandran. Design and implementation of the system interface for pvfs2. <ftp://ftp.parl.clemson.edu/pub/techreports/2002/PARL-2002-008.ps>. Ultima Visita em 20.09.2005 12:12.
- [307] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *High Performance Distributed Computing, 2002. Proceedings. 11th IEEE International Symposium*, Edinburgh, Scotland, July 2002. IEEE Computer Society Press.
- [308] J. L. Reyes and J. Fernandez Haeger. Sequential co-operative load transport in the seed-harvesting ant messor barbarus. *Insectes Sociaux*, 46:119–125, 1999.
- [309] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive control of distributed applications. In *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*, July 1998.
- [310] C. Rolland. *Latex: Guide Pratique*. Addison-Wesley France, SA, 1995.
- [311] C. De Rose, F. Blanco, N. Maillard, K. Saikoski, R. Novaes, O. Richard, and B. Richard. The virtual cluster: A dynamic environment for exploitation of idle network resources. In *Proceedings of 14th Symposium on Computer Architectures and High Performance Computing*, 2002.
- [312] C. De Rose and P. Navaux. Fundamentos de processamento de alto desempenho. In *ERAD 2002 - 2a Escola Regional de Alto Desempenho*, Janeiro 2002.
- [313] Rob Ross, Walt Ligon, , and Phil Carns. Parallel virtual file system. <http://www.parl.clemson.edu/pvfs2/sc2002-whitepaper-pvfs.pdf>. Ultima Visita em 20.09.2005 12:12.

- [314] D. De Roure, N. R. Jennings, and N. R. Shadbolt. The semantic grid: Past, present and future. In *Proceedings of the IEEE*, volume 93, 2003.
- [315] David De Roure, Mark A. Baker, Nicholas R. Jennings, and Nigel R. Shadbolt. The evolution of the Grid. *Int. J. of Concurrency and Computation: Practice and Experience*, to appear.
- [316] Antony I. T. Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.
- [317] Alex Salkever. Trading cpu time like corn? http://yahoo.businessweek.com/technology/content/nov2004/tc2004119_3747_tc162.htm, November 2004.
- [318] E. Santos-Neto. A knowledge-free scheduling approach to improve the performance of data intensive grid applications. In *Proceedings of Research Colloquium on Third IFIP Conference*, September 2003.
- [319] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. *Lecture Notes in Computer Science*, 3277:210–232, 2005.
- [320] E. L. Santos-Neto, L. E. F. Tenório, E. J. S. Fonseca, S. B. Cavalcanti, and J. M. Hickmann. Parallel visualization of the optical pulse through a doped optical fiber. In *Proceedings of Annual Meeting of the Division of Computational Physics (abstract)*, June 2001.
- [321] Elizeu Santos-Neto. Comparative performance analysis between mygrid 2.1.3 and mygrid 3.0. In *HP-OurGrid-Technical Report*, 2004.
- [322] L. F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, 18(4):561–572, 2002.
- [323] L. F. G. Sarmenta and Satoshi Hirano. Bayanihan: building and studying Web-based volunteer computing systems using Java. *Future Generation Computer Systems*, 15(5-6):675–686, 1999.
- [324] E. Sarmiento. Inside jail. <http://www.daemonnews.org/200109/jailint.html>, 2001.

- [325] Mahadev Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Punnet Kumar, and Qi Lu. Experience with disconnected operation in a mobile computing environment. <http://www-2.cs.cmu.edu/afs/cs/project/coda/Web/docdir/mobile93.pdf>. Ultima Visita em 20.09.2005 12:12.
- [326] Michael F. Schwartz. A scalable, non-hierarchical resource discovery mechanism based on probabilistic protocols. *Technical Report*, CU-CS-474-90, June 1990.
- [327] G. Shao, R. Wolski, and F. Berman. Predicting the cost of redistribution in scheduling. In *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.
- [328] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network file system (nfs) version 4 protocol. <http://rfc-ref.org/RFC-TEXTS/3530/index.html>. Ultima Visita em 20.09.2005 12:12.
- [329] Ken Shirri. The sprite operating system. <http://www.cs.berkeley.edu/projects/sprite/sprite.html>. Ultima Visita em 20.09.2005 12:12.
- [330] David SKILLICORN. *Foundations of Parallel Programming*. Cambridge: University Press, 1994.
- [331] Domenico SKILLICORN, David B.; TALIA. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, june 1998.
- [332] Joseph D. Sloan. Ten tips for building your first high-performance cluster. http://www.linuxdevcenter.com/pub/a/linux/2004/12/29/lnxclstrs_10.html. Ultima Visita em 20.05.2006 12:20.
- [333] S. Smallen, H. Casanova, and F. Berman. Applying scheduling and tuning to on-line parallel tomography, November 2001.
- [334] S. Smallen, W. Cirne, and J. Frey et. al. Combining workstations and supercomputers to support grid applications: The parallel tomography experience, 2000.
- [335] J. Smith and S. K. Shrivastava. A system for fault-tolerant execution of data and compute intensive programs over a network of workstations. In *Lecture Notes in Computer Science*, volume 1123. IEEE Press, 1996.

- [336] W. Smith. A framework for control and observation in distributed environments. In *NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA. NAS-01-006*, June 2001.
- [337] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. *Lecture Notes in Computer Science*, 1459:122–142, 1998.
- [338] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of the IPDPS Conference*, May 2000.
- [339] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O’Keefe. The global file system. <http://www.diku.dk/undervisning/2003e/314/papers/soltis97global.pdf>. Ultima Visita em 20.09.2005 12:12.
- [340] I. Sommerville. *Software Engineering*. Pearson Education Deutschland GmbH, 6th edition, 2001.
- [341] S. Son and M. Livny. Recovering internet symmetry in distributed systems computing. In *Proceedings of GAN - Workshop on Grids and Advanced Networks*, 2003.
- [342] R. Sasic. Introspective computer systems. *Electrotechnical Review*, 59(5):292–298, December 1992.
- [343] B. Sotomayor. Globus toolkit 3 tutorial. <http://gdp.globus.org/gt3-tutorial/>, 2004.
- [344] E. STERLING, L; SHAPIRO. *The Art of Prolog*. 2a Ed. Cambridge: MIT Press, 1994.
- [345] J. Stiles, T. Bartol, E. Salpeter, and M. Salpeter. Monte carlo simulation of neuromuscular transmitter release using mcell, a general simulator of cellular physiological processes. *Computational Neuroscience*, 1998.
- [346] James Surowiecki. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Doubleday, May 2004.
- [347] D Talia. Parallelism in knowledge discovery techniques. *Springer-verlag Berlin*, 2367:127–136, 2002.
- [348] Siew-Joo Tan. Survey reveals web services are fulfilling their promises. In *Yakee Group Report*, September 2004.

- [349] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.
- [350] Albert S. TANENBAUM, Andrew S.; WOODHULL. *Sistemas operacionais: projeto e implementação*. Bookman, 1999.
- [351] Maarten Van. TANENBAUM, Andrew S.; STEEN. *Distributed systems: principles and paradigms*. Prentice Hall, 2002.
- [352] PVFS2 Development Team. Parallel virtual file system, version 2. <http://www.pvfs.org/pvfs2/pvfs2-guide.html>. Ultima Visita em 20.09.2005 12:12.
- [353] TechFest. Ethernet technical summary. <http://www.techfest.com/networking/lan/ethernet4.htm>. Ultima Visita em 20.09.2005 12:12.
- [354] Altair Grid Technologies. Openpbs technical overview. <http://www.openpbs.org/overview.html>. Ultima Visita em 20.04.2006 12:20.
- [355] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny. The kangaroo approach to data movement on the grid. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing*. IEEE Computer Society Press, May 2001.
- [356] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley and Sons Inc., December 2002.
- [357] Alok THAKUR, Rajeev; CHOUDHARY. Efficient algorithms for array redistribution. *IEEE Transactionson Parallel and Distributed Systems*, 6(7):587–594, june 1996.
- [358] Rajeev Thakur. Romio: A high-performance, portable mpi-io implementation. <http://www-unix.mcs.anl.gov/romio/>. Ultima Visita em 20.09.2005 12:12.
- [359] S. Thatte. XLANG: Web services for business process design, 2001.
- [360] Patrick Thibodeau. Sun to allow grid use sales on e-trading market. <http://computerworld.com/managementtopics/ebusiness/story/0,10801,99463,00.html>, February 2005.

- [361] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A monitoring sensor management system for grid environments. In *Proceedings of the IEEE High Performance Distributed Computing Conference (HPDC-9)*, August 2000.
- [362] TOP500. Top500.org. <http://www.top500.org/>. Última Visita em 20.04.2006 12:20.
- [363] H. Topcuoglu, S. Hairi, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.
- [364] Paulo R. Trezentos. Vitara: Network clustering como solução para grandes exigências de processamento - apresentação e contextualização. Technical report, Projeto final de curso de IGE, ADETTI / ISCTE, 1999.
- [365] W. W. Trigeiro, L. J. Thomas, and J. O. McClain. Capacitated lot sizing with setup times. *Management Science*, 35(3):353–366, March 1989.
- [366] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open grid services infrastructure (ogsi) version 1.0. Global Grid Forum Draft Recommendation, June 2003.
- [367] H. T UNG. The structure of parallel algorithms. *Advance in Computers*, 19(1):65–90, 1 1980.
- [368] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. Webos: Operating system services for wide area applications. In *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, 1998.
- [369] L. G VALIANTE. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, august 1990.
- [370] S. Vazhkudai, J. M. Schopf, and I. Foster. Predicting the performance of wide area data transfers. In *Proceedings of the 16th International Parallel and Distributed Process Symposium*. IEEE Computer Society Press, April 2002.
- [371] Bill von Hagen. Exploring the ext3 filesystem. what is journaling? <http://www.linuxplanet.com/linuxplanet/reports/4136/3/>. Última Visita em 20.09.2005 12:12.

- [372] Gregor von Laszewski. Grid Computing: Enabling a Vision for Collaborative Research. In Juha Fagerholm, Juha Haataja, Jari Järvinen, Mikko Lyly, Peter Raback, and Ville Savolainen, editors, *The Sixth International Conference on Applied Parallel Computing*, volume 2367 of *Lecture Notes in Computer Science*, pages 37–52, Espoo, Finland, 15-18 June 2002. Springer.
- [373] Gregor von Laszewski, Gail Pieper, and Patrick Wagstrom. *Performance Evaluation and Characterization of Parallel and Distributed Computing Tools*, chapter Gestalt of the Grid. Wiley Book Series on Parallel and Distributed Computing. to be published 2002.
- [374] W3C. World wide web consortium (w3c). <http://www.w3c.org>. Ultima Visita em 24.01.2005 12:20.
- [375] A. Waheed, W. Smith, J. George, and J. Yan. An infrastructure for monitoring and management in computational grids. In *Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems*, 2000.
- [376] A. Waheed, W. Smith, J. George, and J. Yan. An infrastructure for monitoring and management in computational grids. In *Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems*, 2000.
- [377] C. Waldspurger and W. Wehl. Stride scheduling: Deterministic proportional-share resource mangement. In *Technical Memorandum MIT/LCS/TM-528 (MIT Laboratory for Computer Science)*, June 1995.
- [378] David D.H WARREN. An abstract prolog instruction set. Technical report, Manchester: SRI International, 1983.
- [379] J. Weissman. Gallop: The benefits of wide-area computing for parallel processing. *Journal of Parallel and Distributed Computing*, 54(2), November 1998.
- [380] J. Weissman and Andrew Grimshaw. A framework for partitioning parallel computations in heterogeneous environments. *Concurrency: Practice and Experience*, 7(5), August 1995.
- [381] Von Welch, Frank Siebenlist, Ian Foster John Bresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, and Steven Tuecke. Security for grid services. In *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.
- [382] Wikipedia. 10 gigabit ethernet. http://www.wikipedia.org/wiki/10_Gigabit_Ethernet. Ultima Visita em 20.09.2005 12:12.

- [383] Wikipedia. Pio mode. http://en.wikipedia.org/wiki/Programmed_input/output. Ultima Visita em 20.09.2005 12:12.
- [384] Wikipedia. Scsi. <http://en.wikipedia.org/wiki/Scsi>. Ultima Visita em 20.09.2005 12:12.
- [385] Wikipedia. Serial ata. http://en.wikipedia.org/wiki/Serial_ata. Ultima Visita em 20.09.2005 12:12.
- [386] WikiPedia. Wikipedia, the free encyclopedia. <http://pt.wikipedia.org/wiki/Mainframes>. Ultima Visita em 20.04.2005 12:20.
- [387] WikiPedia. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Block_device. Ultima Visita em 20.04.2005 12:20.
- [388] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU availability of time-shared unix systems on the computational grid. In *Proceedings of 8th International Symposium on High Performance Distributed Computing (HPDC'99)*, August 1999.
- [389] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.
- [390] Adenauer Corrêa Yamin. *Um Estudo das Potencialidades e Limites na Exploração do Paralelismo*. 2006.
- [391] Yifeng Zhu, Hong Jiang, Xiao Qin, Dan Feng, and David R. Swanson. Improved read performance in ceft-pvfs: Cost efective, fault-tolerant parallel virtual file system. <http://www.cs.nmt.edu/~xqin/pubs/ccgrid03.pdf>. Ultima Visita em 20.09.2005 12:12.