Escola Superior de
Tecnologia e de Gestão
INSTITUTO POLITÉCNICO DE BRAGANÇA

Luís Filipe Teixeira Miranda

Web Crawling com Armazenamento em Dicionários Distribuídos

Engenharia Informática

Setembro 2008

Web Crawling
com Armazenamento em Dicionários Distribuídos

Relatório da Disciplina de Projecto
Engenharia Informática
Escola Superior de Tecnologia e de Gestão

Luís Miranda

10 de Setembro de 2008

A Escola Superior de Tecnologia e Gestão não se responsabiliza pelas opiniões expressas neste relatório.

Certifico que li este relatório e que na minha opinião, é adequado no seu conteúdo e forma como demonstrador do trabalho desenvolvido no âmbito da disciplina de Projecto.

_____

José Rufino Orientador

Certifico que li este relatório e que na minha opinião, é adequado no seu conteúdo e forma como demonstrador do trabalho desenvolvido no âmbito da disciplina de Projecto.

_____

José Exposto Co-Orientador

Certifico que li este relatório e que na minha opinião, é adequado no seu conteúdo e forma como demonstrador do trabalho desenvolvido no âmbito da disciplina de Projecto.

_____

Arguente

Aceite para avaliação da disciplina de Projecto

Prefácio

O projecto descrito neste documento tinha como objectivo último a criação de um *web crawler* paralelo/distribuído para ambiente de *cluster*.

O projecto foi divido em quatro fases: 1) familiarização com os conceitos de *web crawling*, 2) familiarização com a linguagem Python e implementação de uma versão centralizada, 3) familiarização com a plataforma Domus de Tabelas de Hash Distribuídas e implementação de uma versão distribuída, 4) experimentação e avaliação das diferentes versões desenvolvidas.

Agradecimentos

Agradeço aos meus orientadores, professor José Rufino e professor José Exposto, que sempre estiveram disponíveis para me atender e tirar dúvidas.

Aos meus pais e aos meus amigos que, à sua medida, contribuíram para a realização deste projecto.

Conteúdo

Lista de Gráficos

Lista de Figuras

Capítulo 1

1 Introdução

Nas últimas duas décadas, a Internet cresceu de uns milhares de páginas para biliões de páginas. Devido a esta explosão em tamanho, os motores de busca têm ganho uma importância cada vez maior como meio de localização da informação de que se necessita. Esses motores de busca assentam em enormes colecções de páginas web adquiridas por *web crawlers* e que são depois sujeitas a mecanismos de indexação de forma a permitir, posteriormente, a localização dos documentos originais, com base nos mais variados interfaces, dos quais os *frontends* dos motores de pesquisa (Google, Yahoo, etc.) são os mais conhecidos.

Todavia, a Internet é um sistema dinâmico, que muda constantemente, para além do volume de dados que encerra ser enorme. Estas características levam à necessidade de os motores de busca implementarem sistemas de *web crawling* altamente eficientes. No entanto, criar *web crawlers* altamente eficientes, capazes de descarregar e processar centenas ou milhares de páginas por segundo, não é tarefa simples.

1.1 Objectivos do Projecto

Neste projecto procurou-se desenvolver um *web crawler* assente na utilização de Tabelas de Hash Distribuídas para o armazenamento das estruturas de dados fundamentais. A proposta original do projecto pode ser encontrada no Apêndice A. De notar, desde logo, que o projecto envolve apenas a componente de *crawling* (identificação de hiperligações), não se tendo atacado a componente da indexação das páginas e documentos descarregados.

1.2 Organização do Relatório

O relatório está organizado em 6 capítulos:

- Capitulo 2 - Enquadramento: aqui são expostos alguns conceitos base, necessários a compreensão do trabalho realizado;

- Capitulo 3 - Tecnologias Base: neste capítulo são explicadas as tecnologias que permitiram o desenvolvimento deste projecto, sendo expostas algumas características das mesmas assim como a sua função no contexto deste projecto.

- Capitulo 4 - Implementação: aqui é explicada a organização do código e a interacção entre os diversos módulos que compõe o programa e ainda a diferença entre as versões desenvolvidas.

- Capitulo 5 - Benchmarking/Experimentação: aqui são expostos o resultados obtidos após a condução de testes ao *web crawler* e ainda a análise crítica dos mesmos.

- Capitulo 6 - Conclusão: conclusões sobre o trabalho realizado, descrição da contribuição pessoal e perspectivas futuras deste projecto.

Capítulo 2

2 Enquadramento

Neste capítulo serão expostos alguns conceitos base fundamentais a compreensão do trabalho desenvolvido.

2.1 Web Crawling

Um *web crawler* (também conhecido como *web robot* ou *web spider*), é um programa que explora a Internet continuamente e de modo automático, sendo o principal método de obtenção de informação dos motores de busca [j].

A abordagem que nos serviu de referência, foi o Mercator [h]. O Mercator é um *web crawler* escrito em Java, sendo modular, de forma a permitir a inclusão de módulos de terceiros.

2.1.1 Algoritmo Base

O algoritmo base de um *web crawler* é relativamente simples. Resumidamente, o processo consiste em [h]:

- descarregar uma lista de páginas web iniciais;
- identificar as hiperligações ou urls nas páginas descarregadas;
- adicionar os urls identificados a uma lista de urls a visitar;
- repetir o processo a partir do passo 1., com base em urls da lista de urls a visitar.

O algoritmo base requer um conjunto de componentes de software fundamentais [h]:

- um componente para armazenar a lista de urls a visitar;
- um componente para descarregar páginas web;
- um componente para extrair os urls das páginas web;

- um componente para determinar se um url já foi visitado antes.

2.1.2 Web Crawling de Alto Desempenho

Se o algoritmo base de um *web crawler* é relativamente simples, já o projecto de um *web crawler* de alto desempenho compreende muitos desafios. Tal *web crawler* deve ser:

- Flexível e configurável: adaptável a diferentes estratégias de *crawling*;
- Eficiente: capaz de descarregar e processar um elevado número de páginas por segundo e de manter estruturas de dados enormes mas de acesso eficiente;
- Económico: modesto nos recursos computacionais necessários,
- Robusto: resistente a terminações anormais e comportamentos estranhos dos servidores web
- Respeitador de regras de etiqueta no acesso aos servidores, de forma a não sobrecarregá-los e a recolher apenas a informação autorizada;
- Facilmente controlável através de interfaces adequados
- Dotado de instrumentação capaz de recolher estatísticas importantes sobre a sua actividade

Os pontos em cima descritos, não são todos respeitados pelo crawler ou então um ponto em partícula não é respeitado na sua totalidade.

Em relação a flexibilidade, pode-se considerar como nada flexível, uma vez que a sua estratégia de crawling apenas depende de factores como a velocidade de ligação a internet ou a velocidade da máquina, como de resto será explicado na secção 4.5.

No que toca a eficiência, o crawler não é capaz de processar um elevado numero de páginas por segundo, mas pode-se considerar capaz de manter estruturas de dados enormes sem comprometer o acesso eficiente as mesmas. Este ponto será discutido na secção dos benchmarks após análise dos resultados.

Na economia, não foi feito nenhum estudo sobre os recursos utilizados.

Pode-se considerar o crawler robusto, na medida em que não é afectado por downloads não concluídos ou a qualquer outro comportamento estranho por parte de um servidor. O bom funcionamento do crawler não está dependente das páginas a que tenta aceder, em caso de erro em um determinado url, passa ao url seguinte, voltando a inserir o primeiro no *Url Frontier* para nova tentativa futura.

O *web crawler* cumpre as regras de etiqueta, existem mecanismos que serão descritos nas próximas páginas que implementam algumas regras para não permitir o acesso abusivo aos servidores das páginas.

É bastante fácil controlar o crawler, as operações sobre o mesmo são poucas e bem descritas na ajuda dos comandos. Este projecto apenas contemplou o interface em linha de comandos apenas.

Quanto as estatísticas, é algo limitado, apenas permite recolher as estatísticas que foram implementadas directamente no código, não havendo uma base de dados com todos os parâmetros de funcionamento do crawler no qual se possa recolher os valores necessários a estatística que se pretende efectuar.

2.1.3 Web Crawling Centralizado e Paralelo/Distribuído

Antes de falar sobre as vantagens e desvantagens de uma abordagem centralizada sobre uma abordagem distribuída na criação de um *web crawler*, convém entender alguns conceitos base. Assim, um sistema de processamento centralizado trata um determinado processo sequencialmente, resolvendo todos os sub-problemas um após o outro. Por seu turno, "Um sistema de processamento distribuído ou paralelo é um sistema que interliga vários nós de processamento (computadores individuais, não necessariamente homogéneos) de maneira que um processo de grande consumo seja executado no nó "mais disponível", ou mesmo subdividido por vários nós, [...] conseguindo-se, portanto, ganhos óbvios nestas soluções: uma tarefa qualquer, se divisível em várias sub-tarefas pode ser realizada em paralelo."[a]

Um web *crawler* centralizado tem a vantagem de ser de simples implementação: baseando-se numa só máquina que faz todo o trabalho, o processo desenrola-se de acordo com o algoritmo básico de um *web crawler*, sendo as tarefas executadas sequencialmente. Mas isto tem óbvias desvantagens. Por exemplo, enquanto se descarrega uma página, o resto dos módulos do *web crawler* estão inactivos, quando poderia estar a executar outras funções, como o processamento das páginas já descarregadas. Uma possibilidade de aumentar o trabalho realizado será implementar o web *crawler* com vários processos ou fios-de-execução; todavia, para além de ser difícil, por vezes, garantir a correcção de uma implementação concorrente, esta abordagem depara-se ainda com os limites de processamento de uma só máquina executando o web *crawler*.

Uma outra possibilidade é a execução de várias instâncias de um web *crawler* em paralelo, em diferentes nós computacionais. Mas também isto levanta outros problemas, como a necessidade de partilhar dados entre as várias instâncias e de garantir a correcção do acesso concorrente aos mesmos, bem como de coordenar a execução (incluindo arranque e paragem) das várias instâncias.

2.2 Tabelas de Hash Distribuídas

Tabelas de Hash Distribuídas (DHTs) são uma classe de sistemas ou estruturas de dados distribuídas que providenciam métodos de pesquisa semelhantes aos de um dicionário: armazenam registos do tipo <chave, valor>, indexados pela componente <chave> e acessíveis através de operações básicas de inserção, remoção e pesquisa. [c]

Em abordagens mais evoluídas, o mapeamento entre as chaves e os nós que guardam os registos encontra-se distribuído pelos nós que suportam a DHT. Desta maneira, é suportada a localização rápida dos registos, mesmo perante a adição, subtracção ou falha de nós.

As tabelas de hash distribuídas formam uma infra-estrutura que pode ser usada para a construção de sistemas complexos como sistemas de ficheiros distribuídos, partilha de ficheiros *peer-to-peer* ou *web caching* cooperativo. Por norma, estes sistemas são [g]:

- Descentralizados: o sistema distribuído é formado colectivamente por todos os nós sem haver uma coordenação centralizada;

- Escaláveis: o sistema funciona eficientemente mesmo com milhões de nós;

- Tolerantes a falhas: o sistema deve-se manter em funcionamento, mesmo com constante adições, subtracções ou falhas de nós;

No contexto deste projecto utilizaram-se DHT's como forma de armazenamento de Dicionários Distribuídos, num ambiente paralelo/distribuído do tipo *cluster*, diferente dos ambientes descentralizados de larga escala onde é mais frequente o recurso a DHT's como mecanismo de localização distribuída. [g]

Capitulo 3

3 Tecnologias de Base

Neste capítulo aborda-se a tecnologias de base usadas para o desenvolvimento do web *crawler* assim como as tecnologias que possibilitam o seu funcionamento.

3.1 Python

O Python é uma linguagem interpretada, multi-paradigma, de alto nível e de aplicação em vários domínios. A sua filosofia é reduzir o esforço do programador face ao esforço da máquina. Entre as organizações que utilizam Python na produção do seu software, incluem-se nomes como o Google ou a NASA [b].

Entre as maiores vantagens do Python, encontra-se 1) a quantidade de módulos existentes para além dos módulos padrão, que possibilitam o uso do Python para quase tudo e 2) a rapidez com que se pode escrever um programa devido a ser uma linguagem simples.

No entanto o Python também apresenta algumas desvantagens, como o menor desempenho do código produzido, face a linguagens compiladas. Mas este facto é grandemente ultrapassado pela vantagem da rapidez com que se produz código. Além disso, o Python permite uma fácil integração com outras linguagens, pode-se assim escrever os módulos em que a rapidez de execução seja crucial numa outra linguagem mais adequada a problema em causa [b].

O tratamento flexível das excepções é também uma mais-valia do Python. No nosso caso, em que o contexto é o de uma aplicação que usa a Internet como campo de pesquisa, essa propriedade é importante, dado que as ligações aos servidores nem sempre são bem sucedidas, sendo necessário tratar esses erros para que não se termine anormalmente.

Refira-se também que a criação e utilização de tipos de dados estruturados em Python é bastante simples. Por exemplo, listas e dicionários são tipos nativos, de utilização fácil.

3.2 Ssh

SSH ou Secure Shell é um protocolo de rede que permite trocar dados de maneira segura entre dois computadores numa rede. Pode ser usado para estabelecer sessões remotas para a execução de comandos noutra máquina, ou para copiar dados entre sistemas numa rede.

No contexto do projecto, o protocolo ssh foi o método de comunicação entre o *frontend* e as restantes máquinas do cluster. Os comandos de operação sobre o crawler são enviados com recurso a este protocolo.

3.3 Rocks

Rocks é uma distribuição de GNU/Linux destinada à criação de clusters de alto desempenho (HPC) [i]. Inclui ferramentas destinadas a transformar um grupo de computadores num cluster HPC, de uma maneira fácil em termos administrativos.

A distribuição Rocks é o sistema operativo em funcionamento no cluster no qual o webcrawler foi testado.

3.4 Domus

Domus é uma arquitectura e uma plataforma para Tabelas de Hash Distribuídas baseadas em *cluster*, com a finalidade de providenciar um sistema de armazenamento distribuído de dicionários [c].

Do ponto de vista do programador, a plataforma Domus apresenta uma biblioteca com interfaces em Python e em C, e fornece um conjunto de ferramentas que permitem um fácil desenvolvimento de aplicações cliente que façam uso das capacidades da plataforma.

A API do Domus dá acesso às mais variadas operações sobre as DHTs como criar, afinar, destruir, desligar ou reiniciar uma DHT, para além das operações usuais sobre dicionários (inserção, remoção e pesquisa de registos <chave, valor>) entre outras. Para além do controlo sobre as DHTs, a API também proporciona interfaces para o controlo do conjunto de serviços que instanciam a arquitectura, designado por cluster Domus. Assim, é possível criar, afinar, destruir, desligar ou reiniciar clusters Domus.

No contexto do projecto, a plataforma Domus providenciou um sistema de armazenamento distribuído, de forma, que todas as instâncias do crawler partilhem os mesmos dados.

Capitulo 4

4 Implementação

Neste capítulo é feita a descrição da implementação e da arquitectura do software. No total, a ultima versão (versão com DHT's), contabiliza certa de duas mil quinhentas e sessenta linhas de código, sem contabilizar os módulos não usados.

4.1 Algoritmo Base

O algoritmo de *crawling* seguido no projecto é semelhante ao algoritmo base referido na secção 2.1.1. Em termos de pseudo-código, o algoritmo é como se segue:

```
Iniciar:

     UrlsAProcessar = ("www.ipb.pt", "www.estig.ipb.pt", …)
     UrlsProcessados = ()
     Processar = V

Enquanto Processar:

     url = UrlsAProcessar.getNext()
     se robots.podeprocessar( url ):
          ficheiro = Download( url )

     se não foiVisto( ficheiro ):
          ListaUrls = PesquisarPorUrls( ficheiro )
          PorCada urlNovo em ListaUrls:
               Se não UrlsProcessados.Contem( urlNovo )
               Então UrlsAProcessar.Inserir (urlNovo )

     UrlsAProcessar.Delete( url )
```

4.2 Versão Centralizada do Web Crawler

De forma a ganhar familiaridade com os conceitos básicos de web crawling e alguma prática de programação com a linguagem Python, desenvolveu-se em primeiro lugar uma versão centralizada do web *crawler*. Esta serviu não só de ponto de partida para as versões distribuídas, como também de termo de comparação com as mesmas.

4.2.1 Arquitectura do Software

A versão centralizada do *crawler* é constituída por oito módulos, inter-relacionados de acordo com a representação da Figura 1, ou por nove módulo de acordo com a Figura 2 dependendo da versão do URL Frontier que está a ser usada.

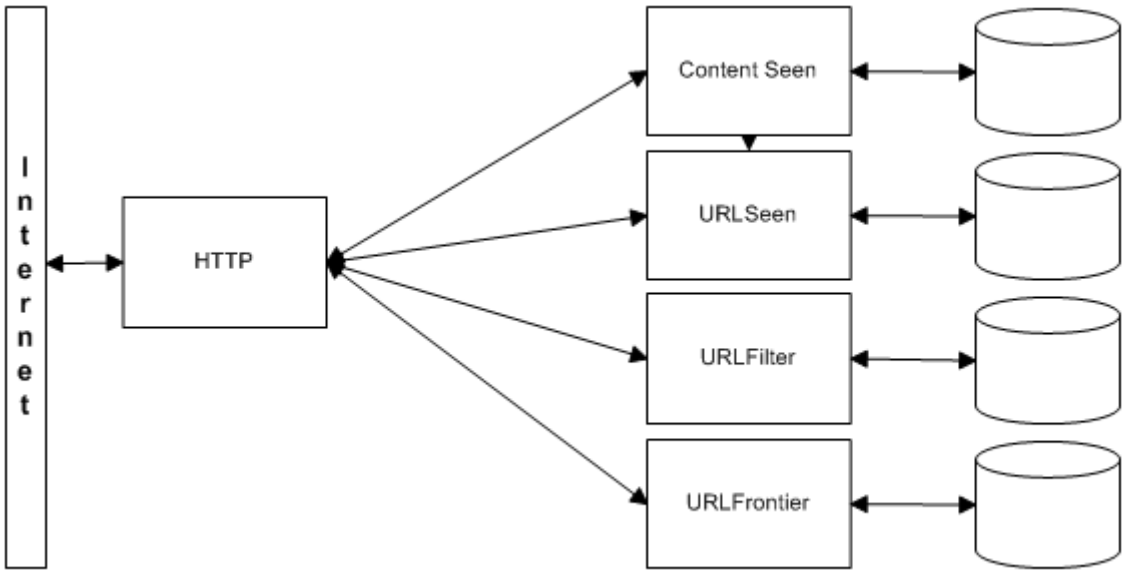


Figura . 1 - Arquitectura do crawler Centralizado

O papel de cada módulo é o seguinte:

- crawler.py : faz uso dos outros módulos; é o que implementa o algoritmo base;

- DNSResolver.py : implementa uma cache DNS e resolve *queries* DNS. O uso de um DNS resolver não é totalmente necessário. A sua implementação foi feita devido a necessidade de a terceira versão do URL Frontier fazer consultas DNS com bastante frequência;

- ContentSeen.py : verifica se uma determinada página já foi antes vista, mas sob outro endereço; isto é feito aplicando uma função de hash com o algoritmo md5 à página em causa e guardando o seu resultado para posterior comparação; estes valores estão armazenados num dicionário Python; usando uma função de hash para fazer o controlo, utiliza-se menos espaço de memória, pois em vez de se guardar um ficheiro completo para posterior comparação, apenas se guarda um número, que neste caso é um hexadecimal de trinta e dois bytes;

- http.py : responsável por descarregar a página apontada por um determinado url e guardar o seu conteúdo num ficheiro temporário no disco ; este módulo retorna o tipo de página/documento descarregada(o) ou o erro ocorrido caso não tenha sido possível fazer a descarga ; note-se que apenas se suportam documentos do tipo text/html;

- LinksExtractor.py : faz o processamento dos documentos em html ; lê o ficheiro respectivo e retorna uma lista com os urls extraídos desse ficheiro;

- URLFilter.py : trata das exclusões impostas pelo protocolo de exclusões de *web crawling*; um site pode ter um ficheiro robots.txt na sua raiz, onde indica quais as partes do site que não devem ser acedidas por um *web crawler*; o módulo recebe um url e verifica se pode ser visitado pelo *crawler*; o módulo foi construído recorrendo ao processador de robots.txt da ferramenta "nikita the spider" [e], uma vez que o processador de robots.txt nativo do Python tem alguns problemas, como não aceitar ficheiros com caracteres não ASCII, entre outros [e]; o módulo tem também uma cache para guardar os ficheiros robots.txt já descarregados;

- URLSeen.py : contém todos os urls que já foram visitados; antes de se inserir um url no URLFrontier, o módulo permite verificar se esse endereço já foi visitado.

4.2.2 Versões do URLFrontier

A versão centralizada do *web crawler* suporta três versões do URLFrontier, pensadas inicialmente tendo em vista a comparação da eficiência das estruturas de dados nativas do Python com as DHTs.

A primeira versão é baseada numa lista simples, em que se adiciona um url ao fim da lista, ou se retira um aleatoriamente usando o método nativo do python "randint(a , b)", que retorna um inteiro *N* tal que $a <= N <= b$ tal que ,sendo *b* o número de itens na lista

A segunda versão é baseada num dicionário, em que se insere um par <chave, valor>, sendo a chave o url a adicionar e sendo o valor especificado como *None*. Para retirar um url do dicionário, usa-se o método nativo *popitem()*, que devolve e remove um item aleatório.

A terceira versão, é baseada num dicionário, que contém pares <chave, valor>, cuja chave é o endereço IP de uma máquina e o valor é uma fila (lista) onde serão guardados os urls relativos a essa máquina. Obtém-se assim uma estrutura URLFrontier onde os endereços guardados estão separados consoante a máquina para a qual apontam. Existe também uma lista que contém todos os endereços de máquinas contidos no dicionário. Quando se pretende retirar um endereço do dicionário, gera-se aleatoriamente um índice k desta lista. De seguida acede-se à posição k da lista e consulta-se o IP correspondente. Por fim, acede-se ao dicionário desse IP e retira-se a o primeiro url da fila associada a esse IP. O objectivo desta implementação seria verificar se uma maior organização dos urls dentro da estrutura de dados teria algum ganho de desempenho.

Para maximizar o desempenho, todos os IPs são guardados numa cache, de maneira que não seja necessário obter o IP de uma máquina que já tenha sido visitada sempre que se adiciona um url dessa máquina ao UrlFrontier. A Figura 2 ilustra o papel da cache DNS embutida no DNSResolver.

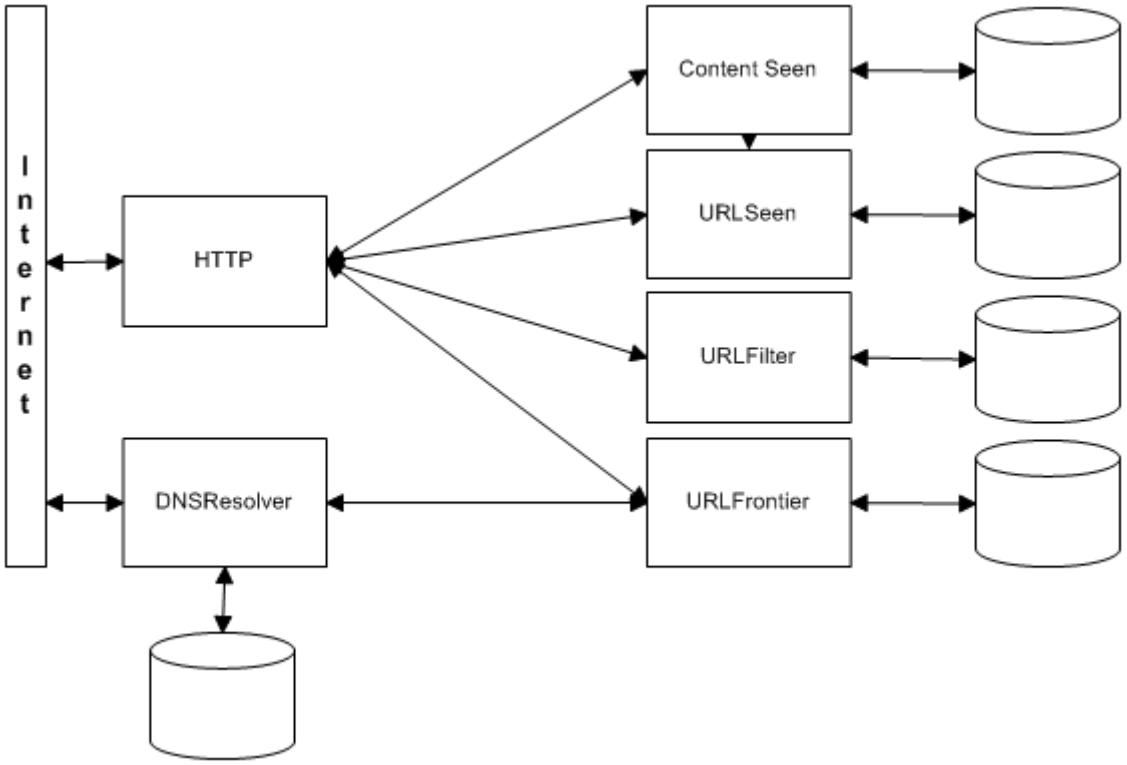


Figura . 2 - Web Crawler com DNS resolver

4.2.3 Suporte do Sistema de Ficheiros

Os ficheiros associados à actividade do *web crawler* centralizado encontram-se distribuídos pela hierarquia representada na Figura 3, assumindo que a execução do *crawler* é feita pelo utilizador user.

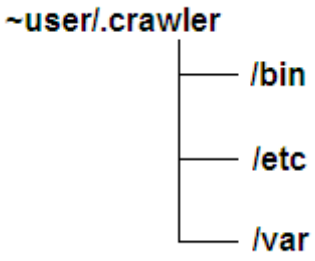


Figura . 3 - Estrutura de directorias

A raiz da hierarquia é a directoria "~user/*.crawler*" ; as subdirectorias procuram seguir, de alguma forma, a organização comum dos sistemas GNU/Linux: a subdirectoria "bin" destina-se a conter todos os ficheiros executáveis; a subdirectoria "etc" contém o ficheiro de configuração; a sub directoria "var" contém os ficheiros produzidos durante a execução do *crawler*. [d]

4.2.4 Ficheiro de Configuração

A Tabela 1 mostra o ficheiro de configuração utilizado, actualmente, pelo *crawler*.

Tabela . 1- Ficheiro de configuração do web crawler centralizado

```
#CRAWLER CONFIGURATION##############################
STATUS_SAVE_INTERVAL   700 #Intervalo de tempo no qual o  estado do crawler é salvo
SEQUENCIAL_ACCESS_LIMIT 4 #número de acessos sequenciais a um site
BLACK_LIST_TIME           4 #tempo em segundos que um site fica na lista negra
FEEDS
http://www.ipb.pt,http://www.estig.ipb.pt #lista de seeds
```

O ficheiro de configuração é um ficheiro de texto onde cada linha pode conter, para além de comentários iniciados por '#', definições de certos parâmetros chave do *crawler*:

- STATUS_SAVE_INTERVAL : intervalo de tempo após o qual o estado do *crawler* é gravado (o estado é gravado periodicamente);
- SEQUENCIAL_ACCESS_LIMIT : número de acessos sequenciais a uma determinada página (para evitar a saturação do servidor);
- BLACK_LIST_TIME : número de segundos que um site deve permanecer na lista negra, após o que o *site* pode ser de novo visitado; para evitar a saturação do servidor;
- SEEDS : lista de sementes

4.3 Versão Paralela/Distribuída Sem Cooperação

Esta versão é uma versão intermédia, entre a versão centralizada e a versão puramente distribuída. Trata-se de versão semelhante à anterior, mas cuja execução é controlada por um módulo crawler_manager.py, que permite arrancar e parar múltiplas instâncias do crawler.py que executam em máquinas distintas, embora ainda sem consciência umas das outras (sem qualquer colaboração entre elas, portanto). O Objectivo desta versão foi apenas o de forçar a implementação e a validação do módulo crawler_manager.py, absolutamente necessário para controlar a versão distribuída e cooperativa do crawler.py, a apresentar na secção 4.4.

No baixo nível, para além da codificação do módulo crawler_manager.py, as principais diferenças entre a versão centralizada e a versão paralela/distribuída sem cooperação dizem respeito ao ficheiro de configuração e à localização dos ficheiros associados ao *crawler*.

Assim, a identificação dos nós nos quais vão ser criadas instâncias do *crawler* passa a fazer parte do ficheiro de configuração, através de uma entrada HOSTS, como mostra a tabela 2:

Tabela . 2 - Ficheiro de configuração do crawler distribuído com armazenamento centralizado

```
#CRAWLER CONFIGURATION##############################
STATUS_SAVE_INTERVAL     700  #Intervalo de tempo no qual o
estado do crawler é salvo
SEQUENCIAL_ACCESS_LIMIT 4  #número de acessos sequenciais a um
site
BLACK_LIST_TIME          4  #tempo em segundos que um site fica
na lista negra
FEEDS                      http://www.ipb.pt,http://www.sapo.pt
#lista de feeds
HOSTS                       c1-1 c1-0#lista de maquinas onde o
crawler vai correr
```

Os ficheiros executáveis mantêm a sua localização na directoria ~user/.*crawler*/bin, assim como os de configuração mantêm a sua localização na directoria ~user/.*crawler*/etc; estas directorias, que se encontram no frontend do cluster, são visíveis automaticamente, via NFS, no resto dos nós do cluster. Por outro lado, em vez de uma só directoria ~user/.*crawler*/var, residente no frontend do cluster, existem agora as directorias /state/partition1/user/.*crawler*/var, em cada nó do cluster onde vai executar o *crawler*.py (sendo /state/partition1/ uma partição local de trabalho); isto deve-se ao facto de cada instância do *crawler* gerir os seus próprios ficheiros e de necessitar acesso eficiente aos mesmos, o que seria prejudicado pela continuação da utilização de NFS; adicionalmente, cada *crawler* tem à sua disposição uma maior quantidade potencial de espaço em disco para usar.

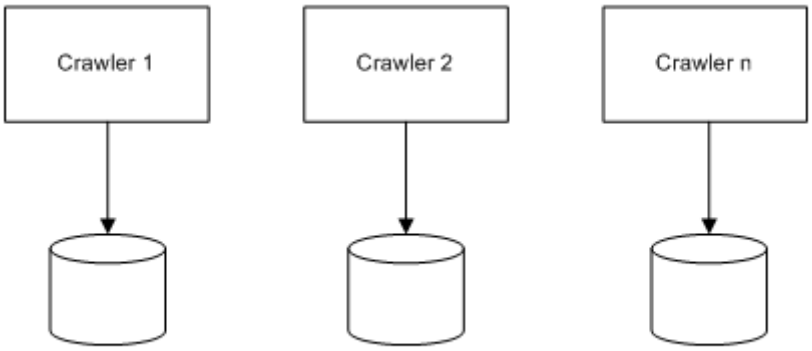


Figura . 4 - Crawlers sem partilha de dados

4.4 Versão Distribuída com Partilha de Dados via DHTs

A principal evolução desta versão em relação à anterior foi a substituição dos dicionários Python que implementavam as estruturas de dados fundamentais do *crawler*, por DHTs Domus (ver Figura 5), mantendo-se o resto do código igual, quer em termos algorítmicos, quer em termos de linguagem Python. Desta maneira, sem que as várias instâncias do *crawler* tenham consciência explícita umas das outras, acabam por partilhar o estado necessário à realização do seu trabalho de forma distribuída e cooperativa. Note-se que, desta forma, não foi necessário desenvolver mecanismos sofisticados de sincronização, dado que esta é feita, de forma completamente transparente, pela camada Domus.

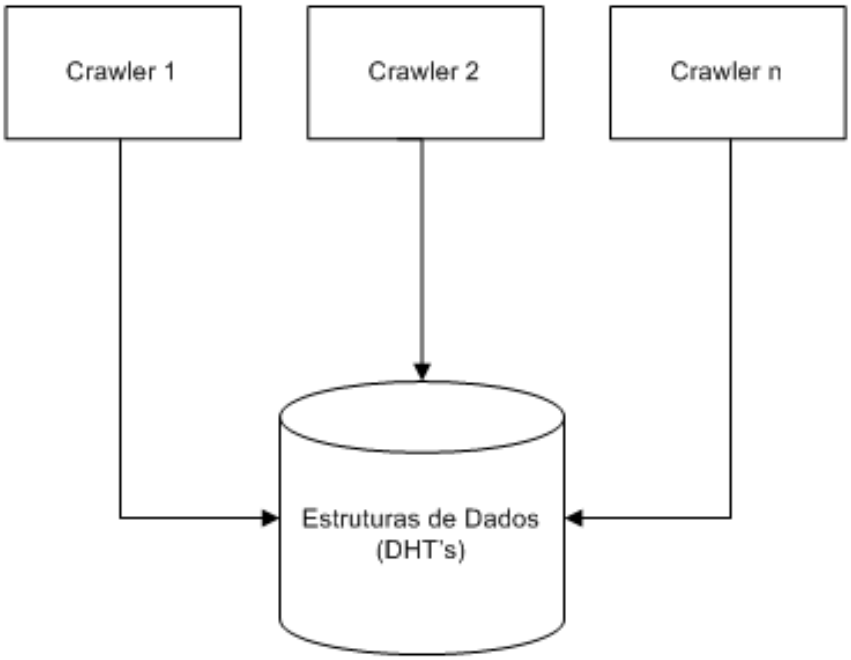


Figura . 5 - Crawlers com partilha de dados

O ficheiro de configuração foi também expandido de forma a incorporar definições associados ao cluster Domus e às DHTs Domus que é necessário operar (ver as definições CLUSTER*, SUPERVISOR* e DHTS_LIST):

Tabela . 3 - Ficheiro de configuração do web crawler distribuido com DHT's

```
#CRAWLER CONFIGURATION##############################

STATUS_SAVE_INTERVAL    700 #Intervalo de tempo no qual o
estado do crawler é salvo
SEQUENCIAL_ACCESS_LIMIT 4 #número de acessos sequenciais a um
site
BLACK_LIST_TIME         4 #tempo em segundos que um site fica
na lista negra
FEEDS                   http://www.ipb.pt,http://www.sapo.pt
#lista de feeds
HOSTS                   c1-1 c1-0#lista de maquinas onde o
crawler vai correr
#CLUSTER DOMUS CONFIGURATION########################

CLUSTER_NAME           mycluster #identificação do cluster
SUPERVISOR_INTERFACE   alfa      #identificação da maquina
supervisora
SUPERVISOR_PORT        7571
CLUSTER_CONF_FILE_PATH
/home/lmiranda/.domus/etc/domus#mycluster#conf
DHTS_LIST
DontentSeenDHT,DNSResolverDHT,URLFilterDHT,URLFrontierDHT,URLS
eenDHT

################################################
```

4.5 Políticas de Crawling

A velocidade do *crawler* (medida em número de páginas processadas por segundo) depende, em larga medida, da largura de banda disponível para descarregar as páginas. Algumas páginas demoram vários segundos a descarregar, enquanto outra demoram décimos de segundo. Uma vez que os urls a visitar são retirados aleatoriamente da estrutura URLFrontier, a probabilidade de se aceder sequencialmente ao mesmo servidor web é diminuta. No entanto, nos primeiros minutos, devido a) ao número relativamente baixo de urls contidos no URLFrontier e b) ao facto de a maior parte dos urls contidos nas páginas iniciais apontarem para o mesmo servidor dessas páginas, é elevada a probabilidade de o mesmo servidor ser visitado várias vezes, num curto espaço de tempo. Donde, implementou-se um mecanismo configurável através de parâmetros definidos no ficheiro de configuração do crawler, que define o número máximo de acessos a uma determinada Após esse número

máximo de acessos, a máquina em causa é referenciada numa "lista negra" dos sites que não podem ser visitados, onde vai permanecer um determinado período de tempo, também configurável, que se encontra definido no ficheiro de configuração do crawler. Expirado esse tempo, se houver uma tentativa de acesso a máquina, a mesma é retirada da "lista negra" e pode ser acedida novamente.

Capítulo 5

5 Experimentação e Benchmarking

5.1 Testes realizados

Nos testes de *benchmarking* comparou-se o desempenho do *crawler* centralizado com o desempenho do *crawler* distribuído com DHTs. Os testes efectuados concentram-se nos tempos de acesso às estruturas de dados, tempo médio de descarga das páginas e dos ficheiros robots.txt, e ainda tempo médio de processamento de uma página.

5.2 Desempenho esperado

À partida, espera-se que o *crawler* centralizado tenha um desempenho maior nos testes de tempo de acesso as estruturas de dados, uma vez que estas são estruturas de dados nativas do Python e se encontram residentes na memória RAM do único nó que sexecuta o crawler. Obviamente, esta expectativa assume que a RAM disponível é suficiente.

Quanto ao tempo de descarga do ficheiro robots.txt, espera-se que seja idêntico enter as duas abordagens. No entanto, como a versão distribuída terá uma maior base de dados sobre ficheiros robots.txt, espera-se também que o tempo dispendido no módulo responsável seja menor sempre que se visite uma página que já tenha sido visitado por outra instância do *crawler*, pois o ficheiro robots.txt estará na cache, não sendo assim necessária nova descarga.

Relativamente ao tempo de descarga das páginas, prevê-se que seja semelhante, embora o número dessas páginas deva ser proporcional ao número de máquinas no qual existem instâncias do *crawler* a correr.

5.3 Desempenho real

Após serem realizados testes de benchmarking aos *crawlers*, obtiveram-se resultados algo surpreendentes. Estes resultados são consequência directa da realização dos testes ter sido feita num ambiente real e não num ambiente controlado (por exemplo, com um ou mais nós, no mesmo *cluster*, albergando servidores WWW virtuais). Num ambiente controlado espera-se que o tempo médio de descarga das páginas seja semelhante. Para além do já descrito, houve alguns comportamentos inesperados por parte dos *crawlers*.

Foram feitas duas rondas de testes. Os resultados são representados de seguida.

5.3.1 Tempo médio de descarga de uma página

Gráfico . 1 - 1ª Ronda: Tempo médio de descarga de uma página

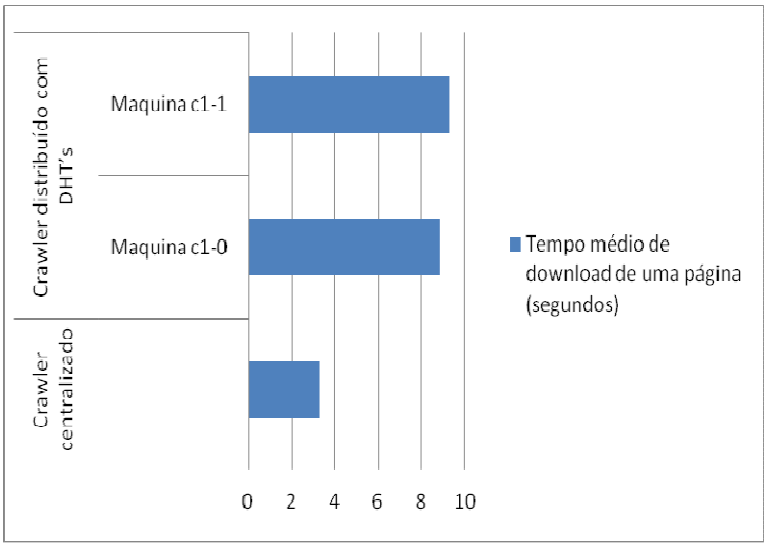
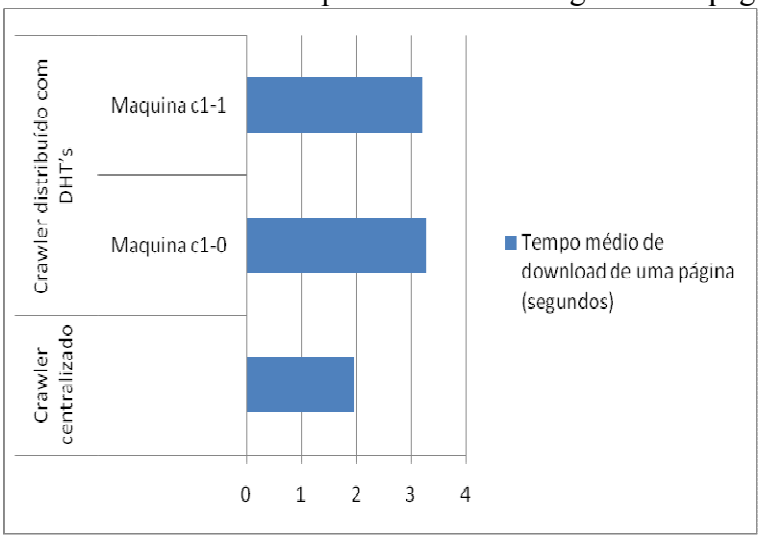


Gráfico . 2 - 2ª Ronda: Tempo médio de descarga de uma página



O tempo médio de descarga no *crawler* distribuido é algo inesperado. Verifica-se, em ambas as rondas, que o seu valor é bastante maior que o valor obtido pelo *crawler* centralizado. Uma vez que o algoritmo de descarga é identico para todas as versões do *crawler*, os tempos de descarga não deveriam ter um diferença tão acentuada. Infelizmente, não houve oportunidade (tempo) para descobrir as causas objectivas deste diferencial.

5.3.2 Tempo médio de descarga do robots.txt

Gráfico . 3 - 1ª Ronda: Tempo médio de descarga de um ficheiro robots.txt

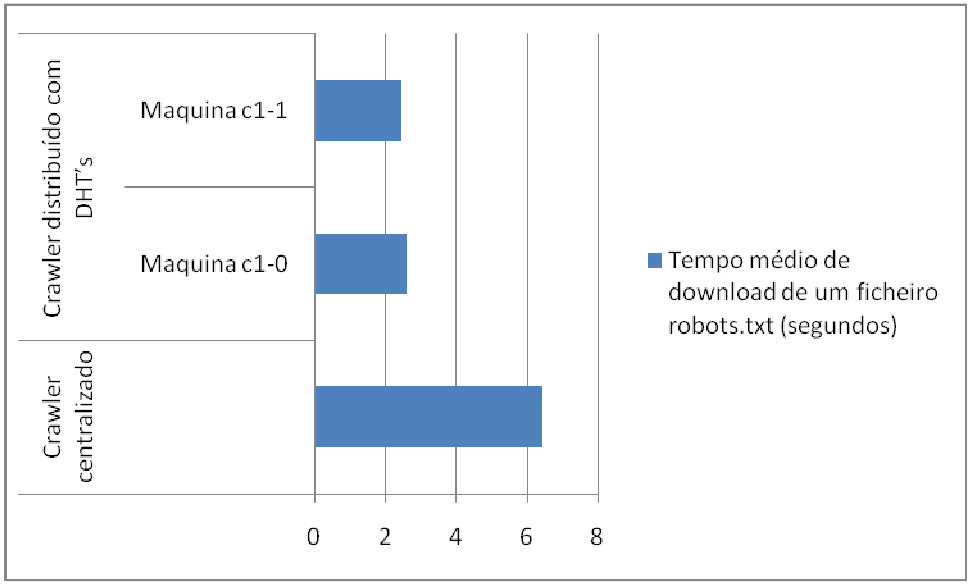
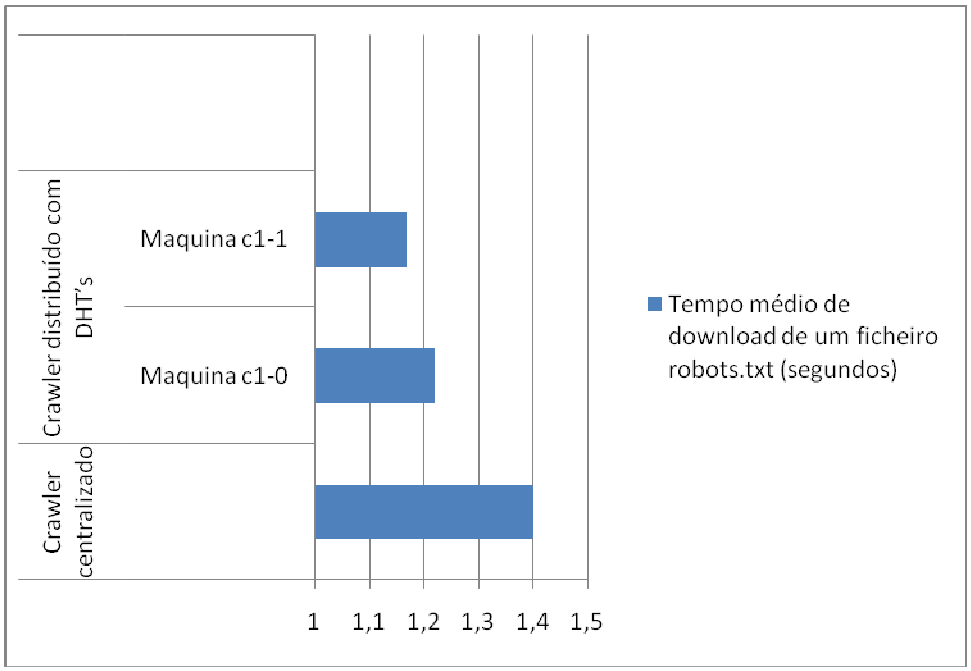


Gráfico . 4 - 2ª Ronda: Tempo médio de descarga de um ficheiro robots.txt



Em ambas as rondas, a diferença de tempos na descarga dos ficheiros *robots.txt* pode ser explicada pelas condições oferecidas pela rede quando os testes foram realizados.

5.3.3 Tempo médio de acesso ao URLFrontier

Gráfico . 5 - 1ª Ronda: Tempo médio de acesso ao Url Frontier

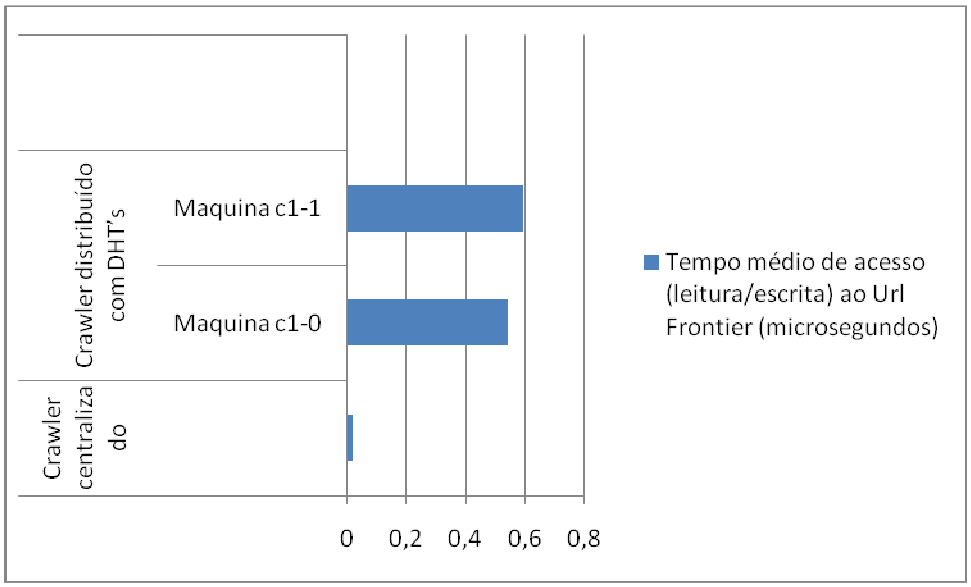
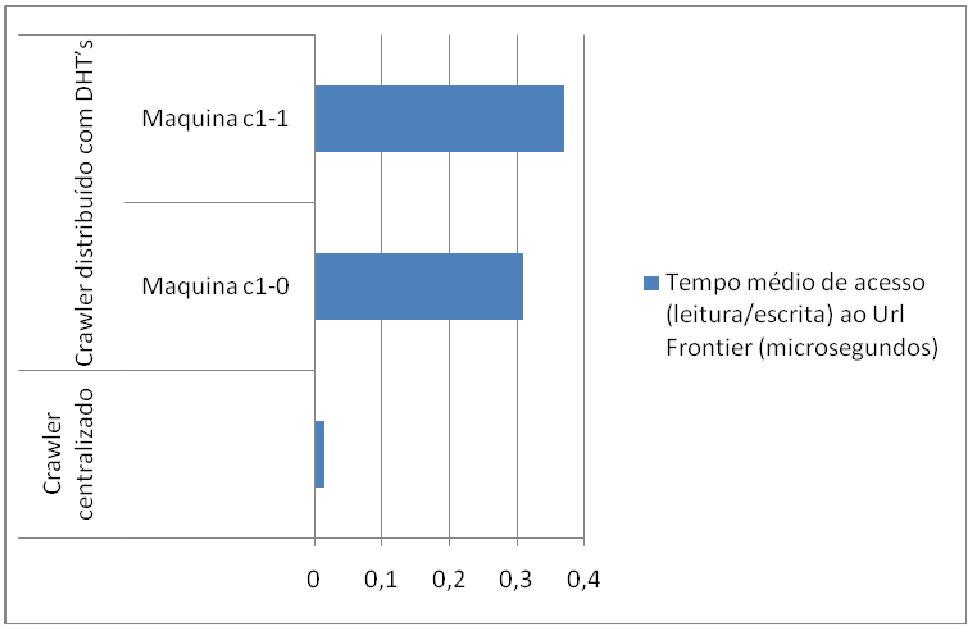


Gráfico . 6 - 2ª Ronda: Tempo médio de acesso ao Url Frontier



O tempo médio de acesso as estruturas de dados vai de encontro ao resultado esperado: o tempo de acesso é menor na versão centralizada.

5.3.4 Número de páginas processadas

Gráfico . 7 - 1ª Ronda: Número de páginas processadas

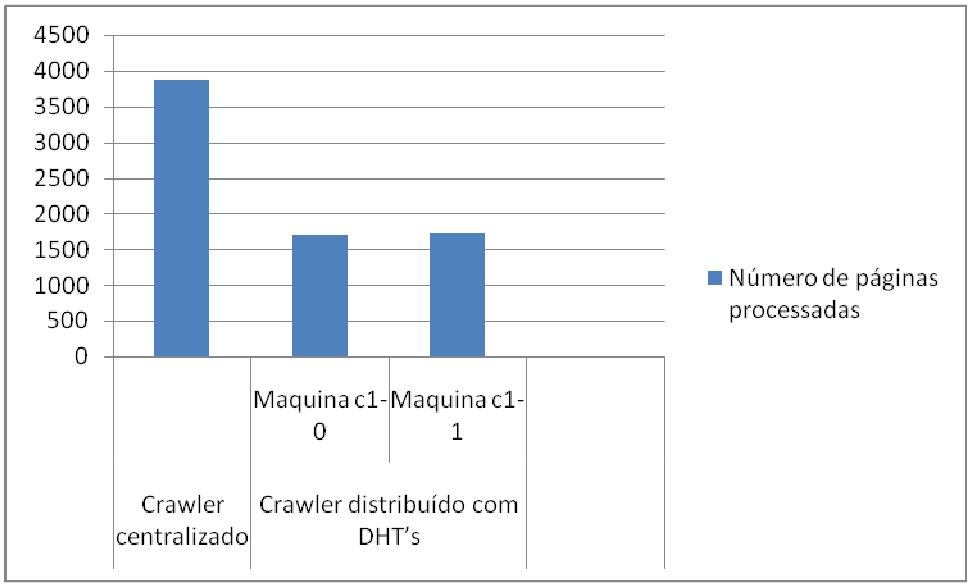
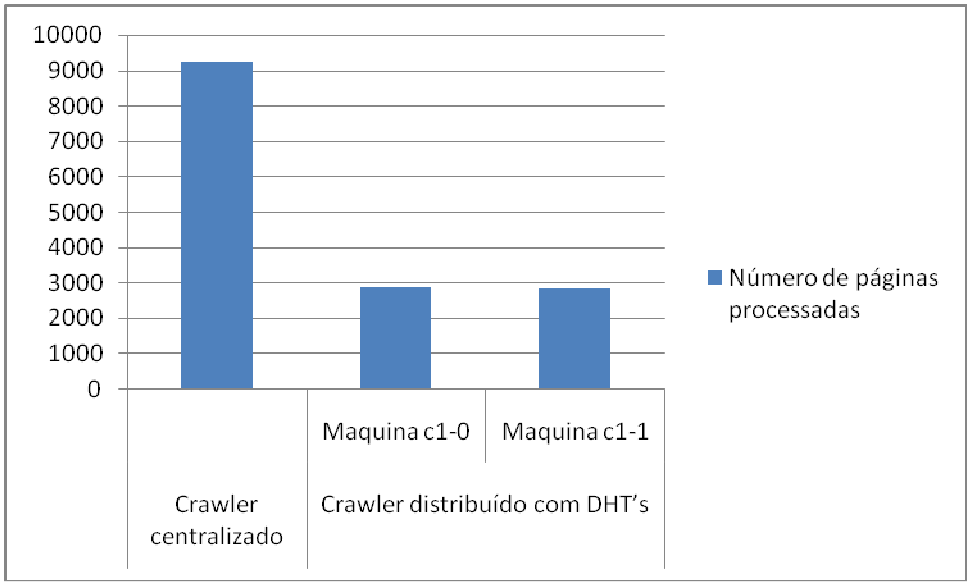


Gráfico . 8 - 2ª Ronda: Número de páginas processadas



No primeira ronda de testes, o desempenho dos crawlers ficou aquém do esperado. Mais uma vez, as condições ofecidas pela rede tiveram um papel decisivo.

Na segunda ronda, a versão centralizada do *crawler* obteve um excelente desempenho, provando que apesar da falta de rigor no *benchmark*, se contsruiu um *web crawler* com potencial para desempenhar com sucesso as suas funções. Mais um vez, desempenho do *web crawler distribuido* ficou aquem do esperado. Durante a execução do *web crawler* distribuído

verificou-se que este caiu numa armadilha para *web crawlers*, situação não prevista infelizmente no *web crawler* desenvolvido e que afectou bastante o seu desempenho.

5.3.5 Número de URLs na fila para descarga

Gráfico . 9 - 1ª Ronda: Número de urls na fila para descarga

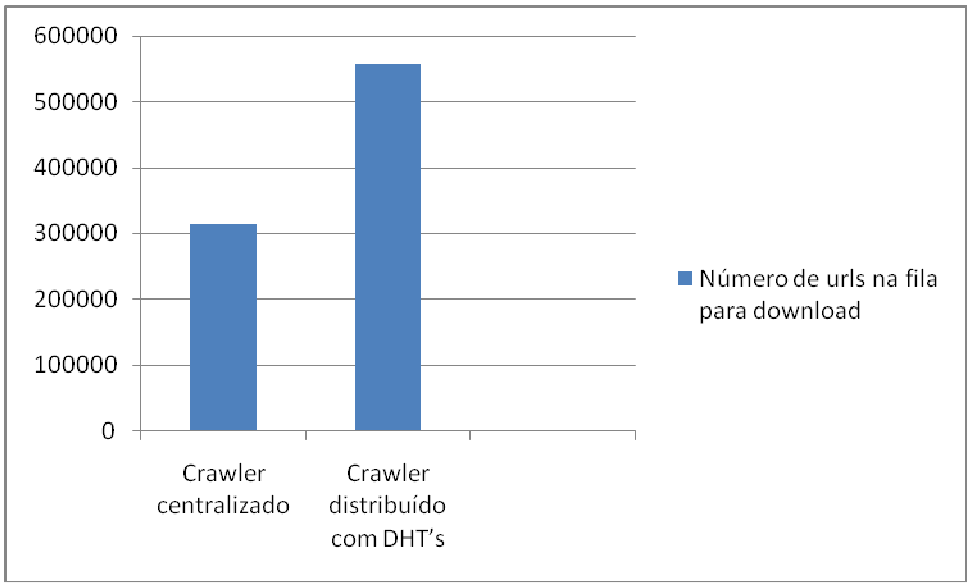
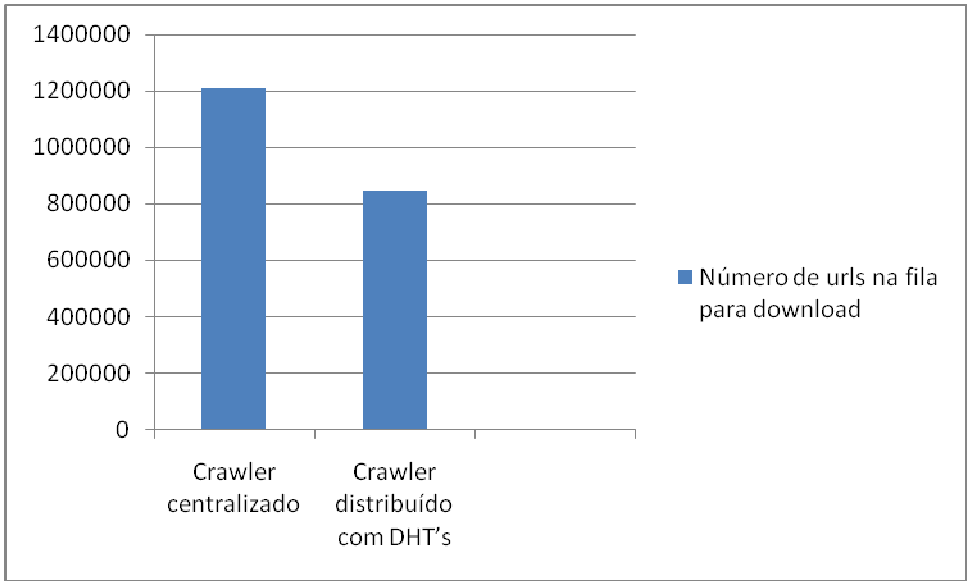


Gráfico . 10 - 2ª Ronda: Número de urls na fila para descarga



Também aqui ficou provado na segunda ronda o bom desempenho do crawler centralizado, com cerca de um milhão de duzentos mil urls gravados na sua base de dados.

5.3.6 Tempo médio de processamento de uma página

Gráfico . 11 - 1ª Ronda: Tempo médio de processamento de uma página

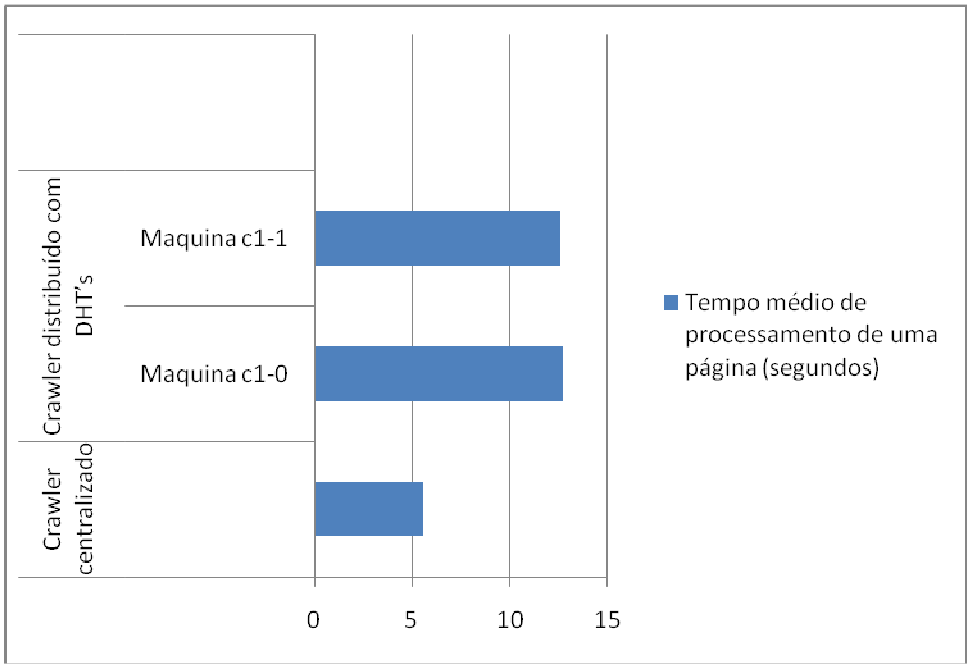
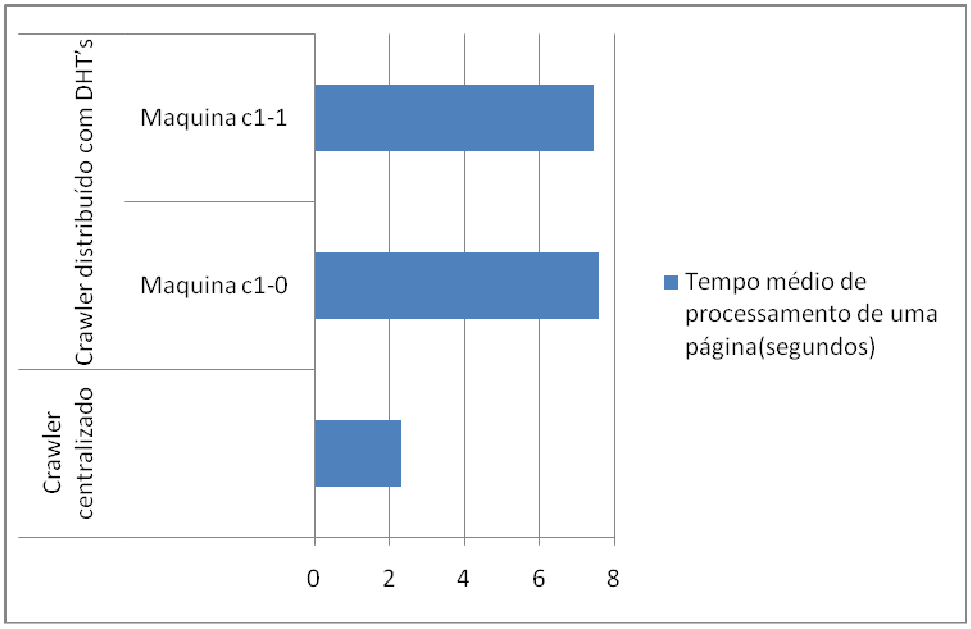


Gráfico . 12 - 2ª Ronda: Tempo médio de processamento de uma página



5.3.7 Discussão

Após análise dos resultados conclui-se que o facto de estes terem sido obtidos num ambiente não controlado contribuiu bastante para os resultados observados. No entanto, pode-se verificar que o algoritmo construído tem potencial para atingir um desempenho satisfatório, como aliás ficou comprovado no teste da versão centralizada na segunda ronda.

Os resultados do *web crawler* distribuído, foram afectados possivelmente por 1) mau desempenho da rede nas horas em que se efectuou os testes, 2) na segunda ronda de testes verificou-se que o *web crawler* distribuído caiu numa armadilha para crawlers. Em alguns testes a soma dos resultados obtidos pelas duas instâncias do crawler distribuído, aproxima-se dos resultados atingidos pela versão centralizada. Sendo assim, conclui-se que em condições ideais e após a resolução de alguns problemas com a versão distribuída, esta terá um desempenho superior a versão centralizada.

Capítulo 6

6 Conclusões

No início do deste projecto, a perspectiva de construir um *web crawler* que respeitasse o algoritmo base r parecia bastante simples. À medida que o trabalho avançou tornou-se notável a complexidade que é desenvolver um *web crawler* capaz de responder minimamente aos requisitos descritos na secção 2.1.2. Mesmo assim, apesar da toda a complexidade por trás de um bom *web crawler*, foi possível contornar algumas características e focar a atenção no desenvolvimento do mais importante e fundamental tendo em conta o objectivo deste projecto.

Considera-se a primeira versão do *web crawler* (*web crawler* centralizado) bem sucedida, na medida que permitiu o desenvolvimento de uma versão distribuída com bastante rapidez. Uma das características que mais contribuiu para isto foi o facto de se ter tido a preocupação de construir um *crawler* bastante modular desde o início. Assim, sempre que foi necessário fazer alterações profundas em determinadas partes do código, essas alterações não implicaram mudança no restante código, aumentando assim a velocidade de desenvolvimento. Outro factor de sucesso, foi a linguagem usada para o desenvolvimento. O Python é uma linguagem simples que se aprende rapidamente. As funções necessárias já se encontravam desenvolvidas na maior parte das vezes, o que contribuiu para o rápido desenvolvimento.

Durante o desenvolvimento e fase de testes foram detectados alguns problemas, o que reflecte uma necessidade de um planeamento mais pormenorizado quando se executa um

trabalho com esta complexidade. Mais um vez o facto de o *crawler* estar bem estruturado foi útil aqui. As alterações necessárias foram efectuadas rapidamente.

Em relação ao testes sobre o desempenho do *crawler*, seria necessário criar um ambiente controlado, possivelmente criar uma rede privada com vários servidores web e fazer o crawling nos sites alojados nesses servidores. Apesar de os testes não terem sido conduzidos num ambiente controlado mas sim num ambiente real, foi possível tirar algumas conclusões.

Nos últimos testes verificaram-se alguns problemas com a versão distribuída. O desempenho ficou aquém do esperado. Para identificar os problemas e desenvolver soluções para os mesmos será necessário testar os *web crawlers* em ambientes controlados. No entanto é previsível que, resolvidos os problemas identificados, o *web crawler* distribuído tem potencial para um desempenho superior.

6.1 Contributo Individual

Com este projecto foi possível compreender a dinâmica de funcionamento de um *web crawler*, ferramenta importante na medida em que é a principal fonte de informação dos motores de busca.

A nível de sistemas distribuídos, ajudou na compreensão dos problemas de tais sistemas assim como as melhores estratégias para a sua resolução e ainda da mecânica de funcionamento desses mesmos sistemas.

O facto de o projecto ser escrito na linguagem Python foi extremamente vantajoso, na medida que possibilitou um contacto sólido com uma linguagem bastante utilizada e poderosa.

6.2 Desenvolvimento futuro

Existem diversas modificações que se podem fazer a este projecto. No que diz respeito ao desempenho, tornar a aplicação *multi-threaded* e optimizar o código no sentido de eliminar alguns passos redundantes ou desnecessários aumentaria o desempenho. No que diz respeito a facilidade de utilização, a criação de um interface gráfico onde seja possível controlar o *web crawler* ou consultar as várias estatísticas constituiria um método de acesso simples ao interface do *web crawler*.

Bibliografia

[a] http://pt.wikipedia.org/wiki/Sistema_de_processamento_distribu%C3%ADdo

[b] http://www.pythonbrasil.com.br/moin.cgi/PerguntasFrequentes/SobrePython

[c] http://www.ipb.pt/~rufino/domus/

[e] http://mail.python.org/pipermail/python-list/2007-October/460650.html

[d] http://doc.vic.computerbank.org.au/tutorials/linuxdirectorystructure/

[f] Pedro Morais, José Nuno Pires, Python – Curso Completo, FCA Editora 2000

[g] http://en.wikipedia.org/wiki/Distributed_hash_table

[h] Allan Heyon, Marc Najork, Compaq Systems Research Center. Mercator: A Scalable, Extensible Web Crawler

[i] http://en.wikipedia.org/wiki/Rocks_Cluster_Distribution

[j] http://en.wikipedia.org/wiki/Web_crawler

Outra referencias online:

http://cis.poly.edu/cs912/parsing.txt

http://www.example-code.com/python/pythonspider.asp

http://www.ibm.com/developerworks/linux/library/l-spider/

http://www.voidspace.org.uk/python/articles/urllib2.shtml

http://www.googleguide.com/google_works.html

http://infolab.stanford.edu/~backrub/google.html

Apêndice

A Proposta Inicial do Projecto

Um Web Crawler (também conhecido por Web Robot) é uma aplicação que varre a World Wide Web, de servidor em servidor, com o objectivo identificar as páginas aí disponíveis e de colectar/produzir informação diversa sobre as mesmas. Basicamente, são mecanismos deste tipo que alimentam os motores de pesquisa que estamos habituados a usar no dia a dia (e.g., Google, Sapo, etc.).

Existem diversas abordagens à realização (implementação) de um Web Crawler. Na sua encarnação mais simples, pode basear-se numa única instância de software, que consulta a Web a partir de uma só máquina cliente (abordagem centralizada). Hipótese mais evoluídas (e realistas, dada a imensidão do espaço de procura, e das quantidades massivas de dados a processar) envolvem a utilização de várias instâncias distribuídas, em cooperação; essas instâncias podem distribuir-se num ambiente do tipo cluster, ou num cenário geograficamente mais alargado (por exemplo, vários clusters, em países/continentes diferentes, cooperando entre si, como é o caso do Google e de sistemas similares).

Neste trabalho pretende-se desenvolver um Web Crawler paralelo/distribuído para ambiente de cluster. Pretende-se um sistema relativamente simples, capaz de executar o algoritmo básico do crawling, sem lugar à indexação das páginas (tarefa a equacionar apenas numa fase avançada do trabalho). O trabalho deverá ser desenvolvido sobre uma plataforma de Dicionários Distribuídos (DDs). Neste contexto, um dicionário é entendido como um repositório de pares <chave,valor>, realizável de múltiplas formas (tabelas de hash, árvores binárias, etc.). Uma abordagem em voga aos DDs são as Tabelas de Hash Distribuídas (DHTs). Assim, uma possibilidade para a concretização do projecto é a utilização da plataforma Domus para DHTs em cluster, disponível no Laboratório de Informática. A interacção com o Domus faz-se via Python, uma linguagem de alto nível que permite o desenvolvimento rápido de aplicações multi-plataforma e cuja utilização se recomenda para a codificação do Web Crawler, dada a disponibilidade de módulos e bibliotecas afins, que poderão auxiliar na prossecução do projecto (por exemplo, a conversação, via HTTP, com servidores WWW, recorrerá a módulos

B Código Fonte

O código fonte deste projecto pode ser encontrado em www.alunos.ipb.pt/~ei13321/crawlers.rar

C Manual de Utilização do Web Crawler Distribuido com DHT's

```
[user@frontend]# ./crawler_manager.py -h
Usage: /home/'user'/.crawler/bin/crawler_manager.py "
       -c| -h | { -o operation [specific_options] }"
       operation:        { start | resume | kill | remove}"
       specific_options: { -o kill ... [-k kill_signal]}"
       kill_signal:      { TERM (=> exit) | USR1 (=> save and exit)  }
```

Parametros:

- -h: imprime informação de uso;
- -c: operações sobre o cluster domus
  - destroy: destrói um cluster domus;
  - create: cria um cluster domus
- -o: opções de manuseamento (inclui parâmetros adicionais) são:
  - start : inicia um novo crawler;
  - remove: remove ficheiros do crawler após uma terminação anormal;
  - resume: resume o estado guardado em disco;
  - kill: envia um sinal para o crawler; o sinal é especificado pelo parametro -k kill_signal, onde kill_signal pode ser:
    - TERM: termina o web crawler;
    - USR1: salva o estado do web crawler em disco e depois termina;

Para iniciar o *web crawler* basta usar o parâmetro *start*. Todas as operações sobre as DHT's ou sobre Cluster Domus (criar, destruir, desligar) são geridas pelo próprio *web crawler* com recurso a API do Domus, não sendo necessário por parte do utilizador qualquer preocupação em iniciar esses serviços previamente.