Père Cluster User Guide (Version 1.0)

1 Request an account on *Père*

Faculty, staff, and students at Marquette University may apply for an account on *Père* by submitting a request to ITS help desk http://www.marquette.edu/its/help/ and sending a completed research computing account request form to *its-rcs@marquette.edu*. For a student user, he/she needs to have a faculty sponsor agreed that such account is necessary.

2 Access *Père*

2.1 Access *Père* from a Windows Machine

On a Windows machine, you can use Secure Shell Client (*SSH*) or *Putty* to access Pere using the following account information.

```
Hostname: pere.marquette.edu
Username: <your-Marquette-user-id>
Password: <your-Marquette-password>
```

2.2 Access *Père* from a Linux Machine

On a Linux machine, you can use the ssh command to access *Père* . If you do not need run *X* applications, the following command is required.

```
ssh  <your-Marquette-user-id>@pere.marquette.edu
```

If you may access X-based applications, the following command is recommended.

```
ssh -Y <your-Marquette-user-id>@pere.marquette.edu
```

3 Change Your Password

Since your account on *Père* is associated with your Marquette account, you need to follow the same procedure of changing your eMarq password. The link http://www.marquette.edu/its/help/emarqinfo/password.shtml provides a guide on how to change your Marquette password.

4 Using Environment Modules

The Environment Modules http://modules.sourceforge.net/ allows the user to customize/modify environment settings via modulefiles. The provides flexible controls on what versions of a software package will be used when you compile or run a program. Below is a list of commands for using modules.

```
module avail            check which modules are available
module load <module>    set up shell variables to use a module
module unload <module>  remove a module
module list             show all loaded modules
module help             get help on using module
```

5 Compile MPI programs

There are multiple ways to compile an **MPI** program. A recommended approach is to use the compiler wrapper provided in an **MPI** implementation e.g. *mpicc*, *mpicxx*, *mpif70*, *mpif90*). We have installed several **MPI** implementations on *Père* to serve different user requirements. You may choose your favorite one as default to compile your code. Here is an example using **OpenMPI** to compile your code.

```
# set the environment for openmpi
Module load  openmpi/gcc/1.2.8
# compile a c code
mpicc -o prog prog.c
# compile a c++ code
mpicxx -o prog prog.c
# compile a f90 code
mpif90 -o prog prog.f
```

You may put the compiler compile options and commands into a Makefile to reduce later compiling efforts.

6 Running Jobs with PBS/TORQUE

Père is currently configured both PBS/TORQUE and **Condor** to manage jobs. While essentially both provide similar functionalities and you can use either one for both serial and parallel jobs, our recommendation is using **Condor** for sequential jobs and using **PBS** for serial jobs and groups of related jobs.

For **PBS** , the typical commands are listed as follows.

```
qsub  myjob.qsub          submit job scripts
qstat                     view job status
qdel <job-id>             delete job
pbsnodes                  show nodes status
pbstop                    show queue status
```

For each command, you can find its usage by typing

```
man <command>
```

Or

```
<command> -h
```

You can find brief but useful user guide on **PBS** at http://www.doesciencegrid.org/public/pbs/homepage.html and most supercomputer centers.

6.1 Submitting serial jobs

To run serial jobs, you can use the following template to write your own scripts.

```
#!/bin/sh
#PBS -N <my-job-name>
#PBS -l walltime=00:01:00
#PBS -q batch
#PBS -j oe
#PBS -o $PBS_JOBNAME-$PBS_JOBID.log

cd $PBS_O_WORKDIR

myprog param1 param2
```

6.2 Submitting parallel jobs

To run **MPI** job, you can use the following template to write your own scripts.

```
#!/bin/sh
#PBS -N my-job-name
#PBS -l nodes=8:ppn=8,walltime=01:00:00
#PBS -q batch
#PBS -j oe
#PBS -o $PBS_JOBNAME-$PBS_JOBID.log

cd $PBS_O_WORKDIR

module load openmpi/gcc/1.2.8

mpirun -np 64 --hostfile `echo $PBS_NODEFILE` myprog param1 param2
```

7 Running jobs with Condor

Condor is a workload management system for running compute-intensive jobs on distrusted computer systems. Like other full-featured batch systems, **Condor** provides capabilities like job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. When users submit their serial or parallel jobs to **Condor** , **Condor** places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

7.1 Setup shell environment for Condor

The environment for **Condor** should be automatically setup for all users. You can check if this is true by typing the following command.

```
which condor_submit
```

If the system complains that no em condor_submit was found, you may add the following lines to your shell startup files (e.g. $HOME/.bashrc).

If you are using bash, add the following line to $HOME/.bashrc:

```
source /etc/profile.d/condor.sh
```

If you are using tcsh, add the following line to $HOME/.cshrc:

```
source /etc/profile.d/condor.csh
```

7.2 Create Condor job submit file

A **Condor** job submit file tells the **Condor** system how to run a specific job for the user. The complexity of a**Condor** job submit file varies with the nature and the complexity of the user's job. We recommend the user reading the **Condor** user manual (http://www.cs.wisc.edu/condor/manual/) before submitting a large number of jobs to *Père* . You may also find many excellent tutorials about **Condor** at http://www.cs.wisc.edu/condor/tutorials/.

Below we show some sample job scripts for several most commonly used job types.

7.2.1 Job submit file for serial job

Assuming you have a serial job you can run with the following command.

```
myprog 4 10
```

You can write a **Condor** job submit file named *serial.sub* as follows:

```
Universe   = vanilla
Executable = myprog
Arguments  = 4 10
Log        = myprog.log
Output     = myprog.out
Error      = myprog.error

Queue
```

The lines in this file have the following meanings.

- Universe: Universe tells CONDOR the job types. The vanilla universe means a plain old job.
- Executable: The name of your program
- Arguments: These are the arguments you want. They will be the same arguments we typed above.
- Log: This is the name of a file where Condor will record information about your job's execution. While it's not required, it is a really good idea to have a log.
- Output: Where Condor should put the standard output from your job.
- Error: Where Condor should put the standard error from your job.

7.2.2 Job submit file for parameter sweep

Parameter sweep is typical case in computational experiments in which you run the same program with a set of inputs.

Assuming you are running the program *myprog* with the following three sets of parameters.

```
myprog 4 10
myprog 4 11
myprog 4 12
```

You can write a **Condor** job submit file named *sweep.sub* as follows.

```
Universe   = vanilla
Executable = myprog
Arguments  = 4 10
Log        = myprog.log
Output     = myprog.$(Process).out
Error      = myprog.$(Process).error

Queue

Arguments = 4 11
Queue

Arguments = 4 12
Queue
```

7.3 Submit Condor job

Once you have a **Condor** job submit file, you can use **condor_submit** to submit your job to the Condor system. For the above two case, the command would be like

```
condor_submit serial.sub
```

or

```
condor_submit sweep.sub
```

7.4 Monitor Condor job

condor_q is a powerful utility provided in the **Condor** system to show the condor the information about **Condor** jobs in the queue. You can find the usage of *condor_q* with either of the following commands.

```
man condor_q
```

or

```
condor_q -h
```

Below are some of the typical usages.

```
condor_q                    list all jobs in the queue
condor_q <user-id>          list all jobs submitted by user <user-id>
condor_q <job-id>           list the job <job-id>
condor_q -long <job-id>     find detailed information for job <job-id>
                            such as which host the job is running on.
```

Another useful **Condor** command is *condor_status*. You can use this command to find the status of the Condor system such as how many jobs is running and how many processors are available for new jobs. Similar, you may consult the man page of *condor_status* to find its advanced usages.

7.5 Stop a Condor job

If you need to stop a **Condor** job you have submitted. You can delete that job using the following command.

```
condor_rm job-id
```

7.6 Job scripts for MPI jobs

Running **MPI** job is similar as running sequential jobs but requires a few changes in the Condor job scripts.

- Modify the job scripts to use "parallel" universe.
- Replace the value of "executable" to an appropriate MPIRUN wrapper.
- Insert your parallel program to the head of the values of the arguments.
- Specify how many processors you want to use using the following option:

  ```
  machine_count =  <NUM_PROCESSORS>
  ```

- Add instruction on whether transfer file and when to transfer.

Here we use an example to show how to run an **MPI** job using a condor.

1. Get an **MPI** sample code and compile it

   ```
   rsync -av /cluster/examples/condor/mpi samples
   module load mvapich/gcc/1.1.0
   cd sample/mpi
   make
   ```

   The command em rsync copy the sample files from a shared directory to a local directory.

   The command em module load set up the **MPI** environment to use *mvapich 1.1.0* compiled with *gcc*. We strongly recommend to use the same implementation to compile and launch an **MPI** program.

If you mix two different implementations, you may unexpect run time errors such as the job is not running as several independent serial jobs instead of a single parallel job.

After the above operations, you will find at least four files:

Makefile simple simple.c simple-mvapich1.sub

The file *simple* is the **MPI** program we will run.

2. Create a condor job submit file named as "simple.sub" which may look like:

```
universe = parallel
executable = /cluster/share/bin/condor-mpirun-mvapich1
output = mpi.out
error = mpi.err
log = mpi.log
arguments = simple
machine_count = 4
should_transfer_files = IF_NEEDED
when_to_transfer_output = on_exit

queue
```

   Here is an **MPI** wrapper for condor that we have created. Since there are multiple **MPI** implementations on *Père* serving different requirements from users, you may choose the one that is appropriate for you. Typically if you are compiling your program from source code, you are free to choose any of them. However, some vendor provided **MPI** programs do require a specific **MPI** implementation and the user should be aware of this.

   In the above example, we uses mvapich_gcc-1.1.0. You can modify the condor-mpirun-mvapich1 to match other **MPI** implementation you want to use. Some **MPI** implementations use *MPD* to manager the **MPI** processors. This is normally unnecessary on *Père* if you are using **OpenMPI** or **MVAPICH** .

3. Submit the **MPI** job to **Condor** Once you have the correct submit file for an **MPI** job, you can treat it as same as a serial job and use *condor_submit*, *condor_q*, and *condor_rm* to manage it.

8 Condor DAGMan

8.0.1 Computation work flow and DAGMan

Frequently, users may run complicated jobs which consist of a set of related jobs. The relations among these jobs usually can be described as a directed acyclic graph (DAG). For example, a typical computational experiment may consist of several steps as shown in the following figure.

Condor provides a useful utility (meta-scheduler) called DAGMan (Directed Acyclic Graph Manager) help the user to construct a work flow to manage dependent jobs. User may refer the DAGMan documentation at http://www.cs.wisc.edu/condor/dagman/ for more detailed information.

8.0.2 the Condor submit file for DAG job

You can write a condor submit file for each task in the above figure and then write another DAG description file to describe all these jobs in a coordinated order. For the above figure, we assume the list of job scripts are:

```
prepare.sub
analyze1.sub
analyze2.sub
analyze3.sub
```

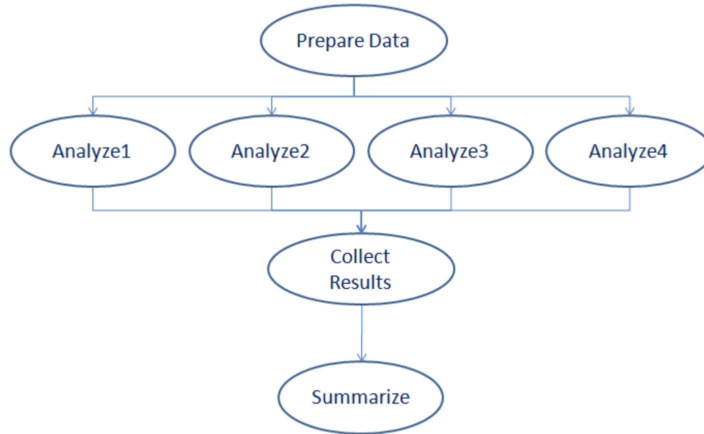

Figure 1: A set of dependent jobs represent as a DAG

```
analyze4.sub
collect.sub
summarize.sub
```

Then the DAG will look like the following:

```
Job prepare  prepare.sub
Job analyze1 analyze1.sub
Job analyze2 analyze2.sub
Job analyze3 analyze3.sub
Job analyze4 analyze4.sub
Job collect  collect.sub
Job summarize summarize.sub

PARENT prepare CHILD analyze1 analyze2 analyze3 analyze4
PARENT analyze1 analyze2 analyze3 analyze4 CHILD collect
PARENT collect CHILD summarize
```

8.0.3 Submit Condor DAG job

Different from normal condor jobs, you need to use *condor_submit_dag* to submit the **Condor** DAG job. Once you submit the DAG job, the **Condor** DAGMan system will keep track of all sub jobs and run them unintendedly based on the the DAG description file and available system resources.