A Guide for Effectively Running Experiments on Condor

Ken Bayer

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln

kbayer@cse.unl.edu

April 8, 2007 at 2:39 p.m.

Abstract

This report explains the use of the Condor scheduling system. We describe the purpose of the Condor, how to use it, and how to design programs to work around its disadvantages.

Contents

1 Introduction

This report explains how to execute a basic job on `Condor`, a distributed computing job scheduler. First, Section 2 describes the advantages and disadvantages of Condor. Section 3 describes how to submit and monitor jobs on Condor. Finally, Section 4 discusses some techniques for structuring your code to best take advantage of how the scheduler operates.

2 When to use Condor

Condor is a job scheduler for distributed computers or clusters. However, it is different from other schedulers such as PBS, the system normally used to run jobs on Prairiefire. PBS is an advanced scheduling system that separates users into prioritized queues, as well as allowing the scheduling of complex parallel jobs. PBS is intended for long running jobs, and thus schedules and reserves resources accordingly.

However, many users do not run jobs that fit the standard PBS profile. When running experiments on CSPs, for example, we often run thousands of very small jobs. Additionally, each atomic experiment usually requires a single processor.

The Condor scheduler is designed for jobs that are short (less than 24 hours) and serial (not using multiple processors). Rather than waiting until a large block of time is available on a single machine, Condor schedules jobs opportunistically. That is, whenever PBS cannot find a job that fits in a small time slot, Condor takes advantage of this time to run one of its jobs. If the Condor job is still running when PBS needs the CPU again, the Condor job is killed.

Thus, there are tradeoffs to using Condor. The biggest advantage is that the opportunistic nature of Condor usually results in a much shorter delay before a job is executed. In practice, there is usually little to no wait before Condor finds a place to run a submitted job. Additionally, there are no separate queues in Condor, so multiple jobs submitted from the same lab all have the same priority as jobs submitted by anyone else.

Condor has two primary disadvantages:

- First, because jobs may be pre-empted at any time, Condor is not an appropriate platform for jobs that require more than 24 hours to terminate.
- Second, also because jobs may be pre-empted at any time, you must design yoursystem such that you can detect which experiments Condor kills in order to rerun them.

If these disadvantages are unacceptable, PBS is a better choice. The lab report written by Lal et al. [2004] describes how to run jobs through PBS.

3 How to submit a job to Condor

Submitting a job to Condor requires first transferring the experimental files to Prairiefire using `scp` (secure copy), and then logging in using `ssh` (secure shell). The steps to running a job on Condor are as follows:

1. Create a script
2. Submit the script
3. Monitor the status of the job

3.1 Creating a Condor script

The following is an example of a Condor script.

```
Universe = vanilla
Executable = /home/programs/java/jdk1.5.0_06/bin/java
Arguments = -cp /home/choueiry/kbayer/java/ MapSolver
Output = address-cons.out
Error = address-cons.err
Notification = Never
Requirements = (Machine!="r04n01" && Machine!="r04n02" && Machine!="r04n03")
Queue 10
```

Next, we describe each part of this script.

Universe: At the time of this writing, this option must always by `vanilla`.

Executable: This line contains the path to the executable that the job will run. In the example above, the path to the Java runtime is used, because that is the executable used to run a Java application.

Arguments: This line contains the command-line arguments to pass to the executable. In the example above, the java executable is passed a classpath as well as the name of the class file to execute.

Output: This line contains the name of the file to which Condor will redirect `stdout`.

Error: This line contains the name of the file to which Condor will redirect `stderr`.

Notification: If this line is omitted, Condor will e-mail the user when the job finishes. If this line has the value `Never`, Condor will not notify the user when the job terminates.

Requirements: This line restricts the job from running on certain machines. If the job being run does not rely on measuring CPU time, this line should be omitted. However, if you must measure CPU time, you must use an appropriate `Requirements` line to restrict the job to identical machines. The exact contents of this line will change whenever Prairiefire is updated, so you should contact the Prairiefire administrators (rcf-support@unl.edu) to help you decide what the `Requirements` should be.

Queue: This line tells Condor to actually submit the job. The number after the word `Queue` indicates how many times Condor should execute the job. In the simplest cases, this value will be 1. However, if you wish to run an identical job multiple times, you can increase this number. Sometimes, if you are looking for reliable CPU measurements, you may want to consider running the same job 10 times and record the minimum value or the average value of the CPU time.

3.2 Submitting a job

Submit a job to Condor by logging into the Prairiefire head node and typing:

`condor_submit <scriptfile>`.

3.3 Monitoring a job

The following commands are useful to know:

- The command `condor_q` lists the status of all jobs in the Condor queue. This command optionally takes arguments to filter the list.
- The command `condor_q <number>` lists the status of the jobs with a given job number.
- The command `condor_q <username>` lists all jobs submitted by a given user.
- The command `condor_rm <number>` removes the specified job from the queue.

4 Designing experiments for Condor

The biggest challenge when using Condor is dealing with a job that Condor killed. One option is to simply submit jobs, wait for them to terminate, and manually rerun any that Condor killed. However, this technique is inefficient.

A better option is to design your code to handle this situation for you. The following steps describe one approach.

1. Choose some shared location for your jobs to communicate. A simple choice is a file. If available, a database would be a better choice.
2. Add code before your actual experimental code that knows all of the experiments you intend to run. This code selects one such experiment and then looks in the shared file to see if it is already being executed. If it is, the code selects another experiment, until it finds one that is not already claimed.
3. After selecting an experiment, the program puts information in the shared file indicating that it is working on the chosen experiment. Thus, when other jobs run, they will not select the same experiment.
4. After finishing the experiment, the program records information in the shared file indicating that the experiment is complete.

Any job that is interrupted will not perform Step 4. Thus, after all jobs terminate, it is possible to write a script that searches through the shared file to remove experiments that did not complete. Then, the original job is executed again. It will automatically search through the file and only run the experiments that were removed by the script. This process is repeated until the script determines that all experiments finished.

This approach is still not ideal, but automates some of the work of rerunning killed experiments. It may be possible to automate the process even further by taking advantage of advanced Condor features.

References

[Lal *et al.*, 2004] Anagh Lal, Praveen Guddetti, Joel Gompert, and Berthe Y. Choueiry. A Practical Guide to the Empirical Evaluation and Comparison of the Performance of Algorithms Operating on CSPs. Working Note of the Constraint Systems Laboratory (UNL), December 2004.