Getting Started with Condor

Irfan Habib

Abstract

How to try Condor for multiplatform distributed computing.

Cluster computing emerged in the early 1990s when hardware prices were dropping and PCs were becoming more and more powerful. Companies were shifting from large mini-computers to small and powerful micro-computers, and many people realized that this would lead to a large-scale waste of computing power, as computing resources were being fragmented more and more. Organizations today have hundreds to thousands of PCs in their offices. Many of them are idle most of the time. However, the same organizations also face huge computation-intensive problems and thus require great computing power to remain competitive—hence the stable demand for supercomputing solutions that largely are built on cluster computing concepts.

Many vendors offer commercial cluster computing solutions. By using free and open-source software, it is possible to forego the purchase of these expensive commercial cluster computing solutions and set up your own cluster. This article describes such a solution, developed by University of Wisconsin, called Condor.

The idea behind Condor is simple. Install it on every machine you want to make part of the cluster. (In Condor terminology, a Condor cluster is called a pool. This article uses both terms interchangeably.) You can launch jobs from any machine, and Condor matches the requirements of the job with the capabilities offered by the idle computers currently available. Once it finds a suitable idle machine, it transfers the job to it, executes it and retrieves the results of the execution. One of the features of Condor is that it doesn't require programs to be modified to run on the cluster.

In practice, however, Condor is more complicated. Condor is installed in different configurations on each machine. Each Condor pool has a central manager. The central manager, as the name implies, is the central manager of the cluster. It manages the detection of new idle machines and coordinates the matchmaking between job requirements and available resources. Machines in a Condor pool also can have Submit and Full Install configurations. Submit machines are those machines that can only submit jobs, but can't run any jobs; Full Install machines are machines that can do both, submit and execute.

Requirements and Installation

Condor does not require the addition of any new hardware to the network; the existing network itself is sufficient. Condor runs on a variety of operating systems, including Linux, Solaris, Digital Unix, AIX, HP-UX and Mac OS X as well as MS Windows 2000 and XP. It supports various architectures, including Intel x86, PowerPC, SPARC and so on. However, jobs developed on one specific architecture, such as Intel x86, will run only on Intel x86 computers. So, it is best if all the computers in a Condor pool are of a single architecture. It is possible, however, for Java applications to run on different architectures.
In this article, we cover the installation from basic tarballs on Linux, although distribution/OS-specific packages also may be available from the official site or sources. (See the Condor Project site for more details, http://www.cs.wisc.edu/condor/downloads.)

Download the tarball from the Project site, and uncompress it with:

```
tar -zvf condor.tar.gz
```

The condor_install script, located in the sbin directory, is all you need to run to set up Condor on a machine. Before you run this script, add a user named condor. For security reasons, Condor does not allow you to run jobs as root; thus, it is advisable to make a new user to protect the system.

One of the first questions the script asks is how many machines are you setting up to be part of the pool? This is important if you have a shared filesystem. If you do, the installation script will prompt you for the names of those machines, and the installation of Condor on those machines will be handled by the software itself. If a shared filesystem does not exist, you have to install Condor manually on each system. Also, if you want to be able to use Java support, you need to have Sun's Java virtual machine installed prior to installing Condor. The install script provides plenty of help and annotation on each question it asks, and you always can turn to Condor's comprehensive user manual and its associated mailing lists for help.

The variable $CONDOR is used from now on to denote the root path where condor has been installed (untarred).

After the installation, start Condor by running:

```
$CONDOR/bin/condor_master
```

This command should spawn all other processes that Condor requires. On the central manager, you should be able to see five condor_ processes running after entering:

```
ps -aux | grep condor
```

On the central manager machine, you should have the following processes:

- condor_master
- condor_collector
- condor_negotiator
- condor_startd
- condor_schedd

All other machines in the pool should have processes for the following:

- condor_master
- condor_startd
- condor_schedd
And, on submit-only machines you will see:

- condor_master
- condor_schedd

After that, you should be able to see the central manager machine as part of your Condor cluster when you run `condor_status`:

```
$CONDOR/bin/condor_status
Name OpSys Arch State Activity LoadAv Mem ActvtyTime Mycluster
LINUX INTEL Unclaimed Idle 0.115 3567 0+00:40:04
Machines Owner Claimed Unclaimed Matched Preempting
INTEL/LINUX 1 0 0 1 0 0
Total 1 0 0 1 0 0
```

If you now run `condor_master` on the other machines in the pool, you should see that they are added to this list within a few minutes (usually around five minutes).

## Launching Jobs on Your New Cluster

To test our new condor setup, let's create a simple “Hello Condor” job:

```c
#include
int main()
{
  printf("Hello World!\n");
}
```

Compile the application with gcc.

Now, to submit a job to Condor, we need to write a submit file. A submit file describes what Condor needs to do with the job—that is, where it will get the input for the application, where to produce the output and if any errors occur, where it should store them:

```text
Universe = Vanilla
Executable = hello
Output = hello.out
Input = hello.in
Error = hello.err
Log = hello.log
Queue
```

The first Universe entry defines the runtime environment under which Condor should run the job. Two Universes are noteworthy: for long jobs, such as those that will last for weeks and months, the Standard Universe is recommended, as it ensures reliability and the ability to save partial execution state and relocate the job to another machine automatically if the first machine crashes. This saves a lot of vital processing effort. However, to use the Standard Universe, the application must be “condor compiled”, and the source code is required. The Vanilla Universe is for jobs that are short-lived, but long jobs also can be executed if the stability of the machines is guaranteed. Vanilla jobs can run unmodified binaries.

Other Universes in Condor include PVM, MPI and Java, for PVM, MPI and Java applications, respectively. For more detail on Condor Universes consult the documentation.

In this example, our executable file is called hello (the traditional “Hello Condor” program), and we're using the Vanilla Universe. The Input, Output, Error and Log directives tell Condor which files to use.
for stdin, stdout and stderr and to log the job’s execution. Finally, the Queue directive specifies how many copies of the program to run.

After you have the submit file ready, run `condor_submit hello.sub` to submit it to Condor. You can check on the status of your job using `condor_q`, which will tell you how many jobs are in the queue, their IDs and whether they're running or idle, along with some statistics.

Condor has many other features; so far we have covered only the basics of getting it up and running. A number of tutorials are available on-line, along with the Condor Manual ([http://www.cs.wisc.edu/condor/manual](http://www.cs.wisc.edu/condor/manual)), that will teach you the basic and advanced capabilities of Condor. When reading the Condor Manual, pay particular attention to the Standard Universe, which allows you to checkpoint your job, and the Java Universe, which allows you to run Java jobs seamlessly.

You also can add Condor to the boot sequence of your central manager and other machines. You can shut down cluster machines, and their jobs will continue or restart on a different machine (depending on whether it's a Standard Universe job or a Vanilla job). This allows for a lot of flexibility in managing a system.

### Beyond Clusters

Condor is not only about clusters. An extension to Condor allows jobs submitted within one pool of machines to execute on another (separate) Condor pool. Condor calls this flocking. If a machine within the pool where a job is submitted is not available to run the job, the job makes its way to another pool. This is enabled by special configuration of the pools.

The simplest flocking configuration sets a few configuration variables in the `condor_config` file. For example, let’s set up an environment where we have two clusters, A and B, and we want jobs submitted in A to be executed in B. Let's say cluster A has its central manager at a.condor.org and B at b.condor.org. Here's the sample configuration:

```plaintext
FLOCK_TO = b.condor.org
FLOCK_COLLECTOR_HOSTS = $(FLOCK_TO)
FLOCK_NEGOTIATOR_HOSTS = $(FLOCK_TO)
```

The `FLOCK_TO` variable can specify multiple pools, by entering a comma-separated list of central managers. The other two variables usually point to the same settings that `FLOCK_TO` does. The configuration macros that must be set in pool B authorize jobs from pool A to flock to pool B. The following is a sample of configuration macros that allows the flocking of jobs from A to B. As in the `FLOCK_TO` field, `FLOCK_FROM` allows users to authorize the flocking of incoming jobs from specific pools:

```plaintext
FLOCK_FROM=a.condor.org
HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR = $(HOSTALLOW_READ), $(FLOCK_FROM)
HOSTALLOW_READ_STARTD = $(HOSTALLOW_READ), $(FLOCK_FROM)
```

The above settings set flocking from pool A to pool B, but not the reverse. To enable flocking in both directions, each direction needs to be considered separately. That is, in pool B you would need to set the `FLOCK_TO`, `FLOCK_COLLECTOR_HOSTS` and `FLOCK_NEGOTIATOR_HOST` to point to pool A, and set up the authorization macros in pool A for B.
Be careful with HOSTALLOW_WRITE and HOSTALLOW_READ. These settings let you define the hosts that are allowed to join your pool, or those that can view the status of your pool but are not allowed to join it, respectively.

Condor provides flexible ways to define the hosts. It is possible, for example, to allow read access only to the hosts that belong to a specific subnet, like this:

HOSTALLOW_READ=127.6.45.*

**Condor-G**

Another way to link distributed Condor pools together is by using Condor's grid computing features, which utilize the Globus Toolkit (http://www.globus.org). The Globus Toolkit is an open-source software toolkit used for building Grid systems and applications. It provides an infrastructure for authentication, authorization and remote job submission (including data transfer) on Grid resources. Condor-G, an extension of Condor, provides all of Condor's job submission features, but for far-removed resources on the Grid.

Condor-G is sort of a gateway to the Grid for Condor pools. Condor-G is a program that manages both a queue of jobs and the resources from one or more sites where those jobs can execute. It communicates with these resources and transfers files to and from these resources using the Globus mechanisms. For more detail on setting up Condor-G, consult the Condor Manual mentioned previously.

A sample submit file for a job to be executed over Globus looks like this:

```none
executable = mygridjob
globusscheduler = grid.sample.net/jobmanager
input = mygridi.txt
universe = globus
output = mygridjob.out
log = mygridjob.log

queue
```

As you can see, there are only two differences with Grid jobs and normal local pool jobs. The Universe is Globus, which tells Condor that this job will be scheduled to the Grid. And, we specify the globusscheduler, which points to the Globus Job manager at the remote site. The jobmanager is the Globus service that is spawned at the remote site to submit, keep track of and manage Grid I/O for jobs running on the local system there. Grid jobs can be monitored the same way as ordinary Condor jobs with condor_q.

**Conclusion**

Condor provides the unique possibility of using our current computing infrastructure and investments to target processing of jobs that are simply beyond the capabilities of our most powerful systems. Condor is easy-to-install and easy-to-use software for setting up clusters. Condor is scalable. It provides options to extend its reach from a single cluster to interconnecting clusters that can be located anywhere in the world. Condor has been fundamental software for many grid computing projects. Various success stories with Condor have been reported in the press. One of the recent ones is of Micron Technologies. Micron is one of the world's leading providers of advanced semiconductor solutions. In an interview
with GridToday in April 2006, a senior fellow at Micron said that they had deployed 11 Condor pools consisting of 11,000 processors, located in four countries in seven different sites. Why Condor? Because it supported all the platforms Micron was interested in, and it was already widely used, well supported, and of course, it was open source. These pools have become a vital asset for Micron. They are used for everything from manufacturing, engineering, reporting and software development to security. Condor is not only a research toy, but also a piece of robust open-source software that solves real-world problems.