Sources for more information

Do you have questions about Condor?

There is a free mailing list for Condor users (not just Condor users) that you can join. To subscribe to condor-users send a message to majordomo@cs.wisc.edu with the body of:

```
subscribe condor-users
```

To unsubscribe from condor-users send a message to majordomo@cs.wisc.edu with the body of:

```
unsubscribe condor-users
```

You can send email to the Condor developers to ask for help. Send your email to condor-admin@cs.wisc.edu. We will do the best we can to answer your question, but we do not guarantee an answer. We are pretty good though.

Using Condor Effectively

A discussion by Alain Roy
Version 1
15-Apr-2004

This document may be found online:
http://www.cs.wisc.edu/~roy/effective_condor
Send comments about this document to:
roy@cs.wisc.edu

Table of Contents

The user log viewer

If you are submitting lots of jobs and want to understand their progress in a graphical way, the Java-based user log viewer is nifty and useful. You can find it in the "contrib." section of the downloads on the Condor web page. Don't be put off by the old version number—it works fine.

Which computer should be removed?

Sometimes a Condor pool administrator would like to remove a computer from the pool. Which one should be removed? The condor_findhost command will tell you which host in the pool can be removed with the minimum impact.

Use condor_fetchlog

When you are debugging problems that require the use of the logs from the daemons on different computers in your pool, use condor_fetchlog instead of logging in to the computer you want to investigate. See the manual for more information about condor_fetchlog, including security: you'll need the correct security settings to be able to use condor_fetchlog.

Miscellaneous Hints

Use large jobs
Think carefully about what kind of jobs you submit to a Condor. Some people submit thousands of jobs that each take 30 seconds to run, then wonder why they see poor performance. Each job submission has a scheduling overhead, and you need to submit jobs that are sufficiently long to make this overhead worthwhile. Instead of submitting 1,000 jobs of 30 seconds each, try submitting 10 jobs of 3,000 seconds each by combining the work into larger portions. You will see significantly better throughput.

Bring it with you
Create jobs that do not require any more than necessary from the execution computer. If you can avoid requiring that software is preinstalled at each node, you will find it easier to switch to new versions of the software (you do not have to reinstall it), and easier to avoid problems in the installation (you don't rely on someone else installing it correctly). Similarly, if you can avoid relying on dynamic libraries that are pre-installed, you avoid problems with out of date or incorrect libraries. Not only will these help you in the short-term, but if you ever want to run your jobs on another Condor pool or on a grid, you'll be that much more ready to run elsewhere.

Synchronized clocks
Keep your clocks synchronized to standard time, and ensure the time zone is correct. If the clock on your submission machine is not close to the time on the gatekeeper host, you could experience various surprising behaviors, particularly if you are using GSI security.

Submitting Java jobs?
When you submit Java jobs, you could use the vanilla universe, and it would work. But it works better when you use the Java universe. This runs your job in a wrapper so that we can tell the difference between your job exiting with an error (which we report to you), and the Java virtual machine exiting with an error (which may cause us to try to run your job elsewhere.)

Waiting for a job to finish?
If you are waiting for a job to finish, you can use condor_wait to wait for you. Although it may not be reliable across, say, reboots of your computer, it is a simple and easy way to wait until a job finishes. See the manual for more information.

Introduction

We hope that you can use the information in this booklet to use Condor effectively. Although the basics of using Condor are not hard, there are a lot of tips and tricks that can help you to use Condor better.

This booklet assumes that you already know the basics of using Condor. If you do not, please review the Condor manual, particularly the first few sections of Chapter 2, *Users' Manual*. The Condor manual can be found online at the Condor web site:

http://www.cs.wisc.edu/condor

If you have comments on this booklet, or suggestions for how to make it better, please let us know. Send email to the Condor team at:

condor-admin@cs.wisc.edu

Always specify a log file in your submit file.

(With a note about apparently idle jobs)

Condor tells you important information in log files. For example, look at this submit file:

1
```
Executable   = analysis
Arguments    = 500
Universe     = vanilla
Input        = analysis.in
Output       = analysis.output
Error        = analysis.error
Log          = analysis.log
Notification = Error
Queue
```

When your job is submitted, Condor will create a file called analysis.log that shows you what happened to your job, and when it happened. Sometimes when job is idle there is a problem listed in the submit file. For example, the submit file says that standard input should come from "analysis.in", but that file was deleted before the job could run.

Here's what I saw in the log file. (Note that the comment lines beginning with a # are comments by me—they are not in the log file.)

2
```
000 (018.000.000) 04/04 22:02:05 Job submitted from host:
<128.105.121.21:33944>...
...
# Aha—this is our problem!
007 (018.000.000) 04/04 22:02:09 Shadow exception!
     Error from starter on chopin.cs.wisc.edu:
     Failed to open standard input file
     '/scratch/roy/personal-condor/simple/analysis.in':
     No such file or directory (errno 2)
```

This message showed up repeatedly in my log file. This is because Condor kept trying to run the job over and over, hoping that the file would eventually show up. Although this seems like a vain hope, it may not be. Perhaps an NFS server is down, and the file will eventually become available. During these repeated attempts, it looked like an idle job to me, because the job ran very briefly, but remained idle between executions. If I had done condor_q at just the right time, I would have seen it running.

To fix the problem, I used condor_rm to remove the job, created analysis.in, and resubmitted the job. (I didn't have to remove it, in this case.) Now in the log file I saw:

This expression will be true when a computer is owned by the physics group, and true is considered to be 1. Otherwise, it's false, or 0. Therefore physics computers will have a higher rank than non-physics computers.

Alternatively, you might have computers prefer jobs that were submitted by physics users. To do this, you instead extend the job ClassAd:

5
```
+IsPhysicsJob = True
```

And change the machine's ranking of a job:

6
```
Rank = (IsPhysicsJob =?= True)
```

There is no security here: you have to trust your users to properly declare what kind of jobs they have.

3) Dynamically adjusting your ClassAds based on the output of a script requires more details than we can describe in this short handout. Fortunately, it is documented in the Hawkeye documentation. Hawkeye is an extension to Condor and is available to anyone with Condor installed. It allows scripts to be run on a periodic basis, and these scripts can update the ClassAd for a computer.

More information about doing this can be found on the Hawkeye web page:

http://www.cs.wisc.edu/condor/hawkeye/

If you are interested in more clever ways to configure your Condor pool, you really should check out the Bologna Batch System. It will open your eyes to the number of cool ways that you can configure Condor. It is described in a short paper on the Condor Web site.

http://www.cs.wisc.edu/condor/technical.html

Enhancing Condor's matchmaking

(Or, extending a computer's ClassAd)

While some people have Condor pools where all of the computers are effectively identical to each other, and it really does not matter where a job runs, some pools are more diverse. You might prefer to run a job on one computer instead of another, when possible, or you might even insist upon it. The easiest way to do this is to add extra attributes to a computer's ClassAd.

As we saw above, a computer is described with a series of attributes, called a ClassAd. You can add any attributes you like to the computer's ClassAd. Here are three examples where you would want to extend the computer's ClassAd:

1) A set of computers have a locked license to a particular piece of software. Or perhaps the software is only installed on a set of particular computers.
2) Some computers are owned by a particular research group, and you prefer to run on their computers first, if they are available.
3) You want to periodically run a benchmark on a computer (network bandwidth? CPU speed? Your choice) and allow jobs to select computers based on this benchmark.

Here's how to do each one:

1) Assume you have MATLAB installed. For those computer, you put this in your condor_config_file:

1
```
HaveMatlab = TRUE
STARTD_EXPRS = HAVE_MATLAB
```

Then in your jobs, put:

2
```
Requirements = (HaveMatlab =?= TRUE)
```

2) Assume some computers are owned by the physics group. For those computers, put into their configuration files:

3
```
OwnedBy = "physics"
STARTD_EXPRS = OwnedBy
```

The jobs have:

4
```
Rank = (OwnedBy == "physics")
```


3
```
# This was the above failed job.
009 (018.000.000) 04/04 22:02:40 Job was aborted by the user.
        via condor_rm (by user roy)
...
# I resubmitted the job a few minutes later
000 (019.000.000) 04/04 22:09:23 Job submitted from host:
<128.105.121.21:33944>
...
# And the job ran a few seconds after that
001 (019.000.000) 04/04 22:09:28 Job executing on host:
<128.105.121.21:33943>
......
# The job has finished.
005 (019.000.000) 04/04 22:09:33 Job terminated.
        (1) Normal termination (return value 0)
```

Without the log file, I would have not learned any of this information, nor seen the timeline for my job's lifetime. You should always specify a log file when you submit a job. Never submit a job without a log file. One day you will need the information in the log file.

Note that you can have multiple jobs using the same log file. You can tell which job is associated with each message by examining the numbers in parentheses. For example, the last message above has *(019.000.000)* in it. That means *cluster 19, process 0, sub-process 0*. In the condor_q display, you would see it listed as 19.0. (Sub-process is shown for historical reasons, and is not used in Condor anymore.)

Advanced condor_q usage

When you submit a job to Condor, you probably know that you can get information about the job using condor_q:

1
```
% condor_q
ID    OWNER SUBMITTED  RUN_TIME  ST PRI SIZE CMD
22.0  roy   4/4  22:29 0+00:00:21 R  0   0.0 analysis1 Hello 5
1 jobs; 1 idle, 0 running, 0 held
```

Some people are unaware that they can find out where their job is running:

2
```
% condor_q -run
ID   OWNER  SUBMITTED    RUN_TIME   HOST(S)
22.0 roy    4/4  22:29   0+00:00:28 chopin.cs.wisc.edu
```

The –run argument is a nice complement to the basic information that condor_q provides. But it does not tell you everything that Condor knows about your job. You can find out all sorts of details about your job if you ask condor_q nicely, using the –l option:

3
```
% condor_q –l

MyType = "Job"
TargetType = "Machine"
ClusterId = 22
QDate = 1081135762
...
JobUniverse = 5
UserLog = "/scratch/roy/personal-condor/simple/van.log"
Requirements = (Arch == "INTEL") && (OpSys == "LINUX")
         && (Disk >= DiskUsage)
         && ((Memory * 1024) >= ImageSize)
            && (TARGET.FileSystemDomain == MY.FileSystemDomain)
RemoteHost = "chopin.cs.wisc.edu"
...
```

The information you see here is a ClassAd. ClassAds are essentially lists of name-value pairs. Details about ClassAds are in the manual. The ClassAd listed here is incomplete, to simplify this discussion. The information in the ClassAd can be very useful, because you can customize the output that condor_q reports. What if you submitted multiple jobs to Condor, from different universes? As you can see from the above output, you normally cannot even tell which universe a job is in. normally

But notice above that we could find the JobUniverse with the –l option. A bit of probing would show you that vanilla jobs have a JobUniverse of 5, while

4
```
periodic_release = (CurrentTime – EnteredCurrentStatus) > 3600
```

If you prefer to have your job simply killed instead of being put on hold, you can do that with the periodic_remove expression instead of periodic_hold.

These expressions can refer to anything in the Job ClassAd. You can look at the ClassAd with condor_q –l to see what attributes you can refer to.

On Exit Expressions

On exit expressions are like periodic expressions, except that they are evaluated when a job completes, not while it is executing. By default, on_exit_remove is always true, so when a job finishes, it is removed from the queue. If you set it to be false, it would always rerun the job. You probably never want that behavior.

But what if you have an ill-behaved job that, if it exits with a segmentation fault it should be restarted, in order to give it another chance? You can do that:

5
```
on_exit_remove = !((ExitBySignal == True) && (ExitSignal == 4))
```

Similarly, you could put the job on hold with on_exit_hold. This would let you look at the job and decide if you want to remove it with condor_rm or run it again with condor_release.

Job policies

Condor allows you to have great control over the running of a job, in several different ways.

Job Priorities

You can control the order in which your jobs are run on the Condor pool by setting the job priority of your job. Jobs with a higher priority run first. For instance, if some of your jobs are more important than others, you can set the following in your submit file:

1
```
priority = 10
```

Jobs with a higher priority will run before jobs with lower priority—assuming everything else is equal. If you have jobs with different requirements, a lower priority job may run first, simply because it is able to run and the higher priority job is unable to run. Job priorities range from -20 to +20.

This does not affect interaction with other users's jobs. Just because you set your jobs to have a high priority does not mean that you get to run before other people. It only affects the relative ordering of your jobs.

Periodic Expressions

Not all jobs are well-behaved. What if you have a job that you know must run less than one hour, but every so often it gets into an infinite loop and runs forever. If you run the jobs again, it will probably work. Yes, you should fix your job, but until you do, Condor can help. You can have Condor periodically check your job and put it on hold if it has been running to long. For instance, you can do:

2
```
periodic_hold = (CurrentTime - JobCurrentStartDate) > 3600
```

This will put your job on hold after about an hour (Condor doesn't check every second, so it may be a bit more than an hour), and that will remove it from the computer it is running on. You can do run condor_release to start it again, or you can have another periodic expression to try it again:

3
```
periodic_release = TRUE
```

Or perhaps, if you wanted to release it from hold after at least an hour passed by, you could do:

standard universe jobs have a JobUniverse of 1.You can constrain condor_q to show you just the vanilla jobs by referring to the Job Universe:

4
```
% condor_q -constraint 'JobUniverse == 5'
ID   OWNER SUBMITTED  RUN_TIME ST PRI SIZE CMD
22.0 roy   4/4  22:29 0+00:04:18 R  0 0.0  analysis1 Hello 5
```

You can constrain based on any attribute in the ClassAd. See the condor_status manual page for more information about using constraints.

You can also tell condor_q to format its output differently. For instance, what if you were annoyed because condor_q chopped off the output in the above example? In fact, condor_q did chop off the arugments: the second argument to the analysis program was 500, not 5. You can use the format command to both show attributes that condor_q does not show, and to show the attributes without being chopped off. For example, to show the job identifier, the job universe, and the arguments, you could do: (The command is one line, but formatted for clarity.)

5
```
% condor_q -format "%d" ClusterId -format ".%d" ProcId \
          -format ": %d" JobUniverse -format " %s\n" Args
22.0: 5 Hello 500
23.0: 1 Hello 500
```

The format command uses flags that are identical to the flags for the C printf() function. If you are familiar with that function, you're all set. If not, check out the man pages, or pick up a book on C.

The format command can be extremely useful for getting customized output. Because the format arguments are so long, you may wish to use a script.

You'll notice that the job universe was printed as a number. There is no easy way to have the –format command do translations on the data, though you could use a simple script to do such translations for you. For instance:

6
```
% condor_q -format "%d" ClusterId -format ".%d" ProcId       \
          -format ": =U%d" JobUniverse -format " %s\n" Args \
 | sed -e "s/=U5/Vanilla/" -e "s/=U1/Std    /"
24.0: Vanilla Hello 500
25.0: Std     Hello 500
```

Clever use of condor_status

You probably know that you can find information about all the computers in your Condor pool with the condor_status command:

1
```
% condor_status

Name          OpSys Arch  State   Actvty LoadAv Mem  ActvtyTime
abulafia.cs.w LINUX INTEL Claimed Busy   1.000   501 0+09:24:57
adenine.stat. LINUX INTEL Owner   Idle   0.390  1006 0+00:00:04
algonquin.sta LINUX INTEL Owner   Idle   0.000   501 5+20:08:46
...
```

You can find out all the information you would like about a computer with the -l argument: (The output has been trimmed to fit the space.)

2
```
% condor_status -l c2-022.cs.wisc.edu
MyType = "Machine"
TargetType = "Job"
Name = "vm1@c2-022.cs.wisc.edu"
Machine = "c2-022.cs.wisc.edu"
...
OpSys = "LINUX"
State = "Claimed"
EnteredCurrentState = 1081138050
Activity = "Busy"
...
Start = (TARGET.ImageSize <= ((Memory - 15) * 1024))
RemoteUser = "roy@cs.wisc.edu"
ClientMachine = "beak.cs.wisc.edu"
...
```

This is a ClassAd, just like the condor_q example above. Notice that the Name attribute contains the “vm1@”. You will only see this on a computer with multiple virtual machines—which usually correspond to having multiple CPUs. The Machine name is the name without the virtual machine identifier.

Now that we know the attributes, we can tell condor_status to only show us specific computers. For example, we can see all Linux computers that are claimed by a user. (A claimed computer is either running a job, or just about to.)

3
```
% condor_status -constraint \
    'State == "Claimed" && OpSys == "LINUX"'

Name          OpSys Arch  State  Actvty   LoadAv Mem ActvtyTime
abulafia.cs. LINUX INTEL Claimed Suspended 0.330 501 0+00:03:33
anfrom.cs.wi LINUX INTEL Claimed Busy      1.000 248 0+13:08:17
arosa.stat.w LINUX INTEL Claimed Busy      0.000 247 0+06:05:04
```


the merge job. If there are any fatal errors, it will create a *rescue DAG*: you can fix the cause of the error, submit the rescue DAG, and DAGMan will resume where it left off.

DAGMan provides a slew of features to control how DAGs are run. One feature is the ability to run a script just before or after a job finishes. This script is not a Condor job, but is directly executed on the computer from which you submit your DAG.

For example, to satisfy the first user request above—“how do I run a script right after my job completes”—you can make a very simple DAG like this:

2
```
Job A myjob
Script POST A myscript
```

After the job completes, DAGMan will run “myscript”.

DAGMan can be used in many situations for many problems. It is not unusual to find users that create very small DAGs, and we also people who have regularly used DAGs with thousands of jobs in them. DAGMan works well in both scenarios.

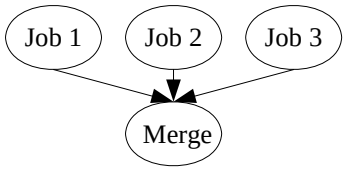
This short introduction just scratches the surface of what DAGMan can do. See Section 2.11 of the Condor manual for more information about DAGMan.

Controlling sets of jobs

A common question we get from users is something like, “I want to run a script after my job runs, but I don’t see an option for it that I can put into my submit file”. Another common question is, “I have three jobs, and when they are done, I want to merge the output of each of them with another job. How do I do it?”

There are no options to condor_submit to do these things. But all is not lost: Condor provides an excellent facility called DAGMan for controlling sets of jobs. Let’s look at the second example first: run three jobs, then another job to merge the output of these jobs. We have to assume that you can provide the job that can merge the output, because Condor will not do that for you.

Let’s say that your three jobs are in three submit files names job1.job, job2.job, and job3.job, and you have a merge job in a submit file named merge-output.job. You then describe these jobs and their relationships in a DAG file. DAG is an acronym for directed acyclic graph. If you can draw non-looping arrows between your jobs to show their order like the picture below, you have a DAG.



Create this DAG file:

1
```
Job job1 job1.job
Job job2 job2.job
Job job3 job3.job
Job merge merge-output.job
Parent job1 job2 job3 Child merge.job
```

This file specifies a mapping between names (like job1) and files (like job1.job). It also says which jobs should run before other jobs. In this case, jobs 1, 2, and 3 will all run before merge.job, as shown in the picture above.

Note that each submit file must contain exactly one job in it. You cannot combine jobs into a single submit file.

You are all set. Just submit your jobs with condor_submit_dag, and DAGMan will submit all of your jobs to Condor in the correct order. It will submit the first three jobs and wait until all three have finished successfully before submitting

This output may be dissatisfying to you though, because the computer names are chopped off. Just like we can specify a format for condor_q, we can specify a format for condor_status, and then you will get the exact information you want.

4
```
% condor_status -constraint \
    'State == "Claimed" && OpSys == "LINUX"'      \
    -format "%s:\t" Name -format "%s\t" OpSys     \
    -format "%s\t" State -format "%s\t" Activity  \
    -format "%s\n" RemoteUser | head -10

anfrom.cs.wisc.edu:   LINUX Claimed Busy      roy@cs.wisc.edu
arosa.stat.wisc.edu:  LINUX Claimed Busy      roy@cs.wisc.edu
barney.cs.wisc.edu:   LINUX Claimed Suspended roy@cs.wisc.edu
```

Notice how we combined both the constraint and the format. Also note the use of \t to include tabs in the output. In general, it is hard to get perfectly neat output using the –format option, but at least you get exactly the information you want. However, you can use printf-style attributes to get slightly better output. For instance, if you know that none of your computer names is more than 25 characters long, you can do this:

5
```
condor_status -constraint 'State == "Claimed"' \
    -format "%25.25s: " Name -format "%s\n" RemoteUser

    adenine.stat.wisc.edu: roy@cs.wisc.edu
       anfrom.cs.wisc.edu: roy@cs.wisc.edu
      arosa.stat.wisc.edu: roy@cs.wisc.edu
```

Note, if you try to print attribute names that don’t exist, or you use the wrong printf-style flags (like %s for a floating-point number) you may have surprising results, or even crashes. Some attributes, like RemoteUser, only exist when a job is running.

Why won't my job run?

A common question from new users is "why won't my job run?" They submit their job, but it sits idly in the queue. Or perhaps it runs, but it only runs on the computer it was submitted from, and not any other computer.

First, check the log file. Did it actually run for a very brief time, but repeatedly fails? See the example above in the section on log files.

Another common mistake is to have forgotten to tell Condor to transfer files. Condor decides that two computers have a shared filesystem (like NFS) if they are in the same FILESYSTEM_DOMAIN. Two common problems arise. First, the computers share a file system, but the FILESYSTEM_DOMAIN is configured differently on each computer. Instead of $(FULL_HOSTNAME), use the domain name your computers are in. For instance, if you are in the example.com domain, set FILESYTEM_DOMAIN to "example.com". Second, if the computers do not share a file system, then you need to transfer files. (On Windows, you usually need to transfer files.) See Section 2.5.4 for information on how to transfer files. You will add a couple of lines to your submit file, probably something like this:

1
```
should_transfer_files = YES
when_to_transfer_ouput = ON_EXIT
transfer_input_files = file1, file2
```

You do not need to specify files like your executable or your standard input file in your transfer_input_files statement: Condor will do those for you automatically.

Make sure that there are computers in your Condor pool that are not busy, and are willing to run jobs. If a computer is busy (have you configured it to not accept jobs when the keyboard is busy?) or running a job, it will not run your new jobs until it is no longer occupied.

If these three common problems don't help you in getting your job running, you may need to get your hands a bit dirtier and look under the hood. We will start simple with condor_q's analyze option. I submitted a job that sat idle, and condor_q can tell me: (I elided some of the output for clarity.)

2
```
% condor_q -analyze
030.000:  Run analysis summary.  Of 2 machines,
      2 are rejected by your job's requirements
WARNING:  Be advised:
   No resources matched request's constraints
   Check the Requirements expression below:

Requirements = (OpSys == "WINNT") && (Arch == "INTEL") && (Disk
>= DiskUsage) && ((Memory * 1024) >= ImageSize) &&
(TARGET.FileSystemDomain == MY.FileSystemDomain)
```


Great—no computers can be used because of my jobs's requirements? Now what?

If you are one of the lucky few, you can use condor_analyze to answer this question for you. This is an enhanced version of condor_q that will tell you which of your requirements (or which computer requirements are failing). It is currently only available as a contrib. module on RedHat Linux, but it will be available on more platforms in the future.

You can analyze this by hand without much difficulty though. Here's what you do:

1) Look at the details of your job. Assuming your job has an id of 30.0, you can do:

3
```
condor_q –l 30.0
```

2) Choose a computer that you believe your job should be able to run on. Pretend the computer's name is houdin.example.com:

4
```
condor_status –l houdin.example.com
```

3) The job and the computer both have a list of requirements. (The computer lists the requirements in the Start expression.) The job's requirements state what must be true about a computer, and the computer's requirements state what must be true about the job. If either of these requirements expressions are not satisfied, then the job will not run.

In the example above, when I looked at the computer, I found:

5
```
OpSys = "LINUX"
```

but the job requires it to be WINNT. There is the crux of my problem: the computer is running Linux, but I require Windows for my job.

Sometimes it takes a while to go through all of the expressions to find the problem, but it usually will help you track down your problem. It will also teach a lot about how Condor works.