



How to Use Condor Effectively for Statistical Computing

Xianhong Xie

Department of Statistics

University of Wisconsin-Madison



Overview of the Talk

- brief introduction of Condor system
- description of relevant Condor universes
- different ways of invoking R
- specify single/multiple jobs in submit file
- requirements and rank specification
- how to use stat cluster machines
- check the progress and manage your jobs
- Condor dag and Perl scripting
- acknowledgements and pointers to resources

Introduction of Condor




- Condor is a matchmaker: jobs and machines
 - process involves matching 2 ClassAds
 - one for your job, one for the machine
- Condor jobs should be batch ready
 - no interactive input/output
 - no graphics to the screen
 - graphics to files OK
- make sure your jobs run correctly without Condor
 - small pilot study will be helpful



Condor Universes



- every Condor job runs in some universe
 - Vanilla
 - Standard
 - Scheduler
 - Globus
 - Vanilla universe can include almost any job
 - only Standard universe jobs can be checkpointed
 - Scheduler universe is used in running DAG jobs
 - Globus universe extends Condor to other batch processing systems
- 

Shared File System or Not

- For jobs on AFS

- give condor write permission

```
fs sa dir net:cs write
```

- `fs la dir` tells you the info

- For jobs on local FS

- for example /scratch, /tmp, etc..

- no need to set special permission for condor

- need to put these in the submit file for Condor jobs not running in the Standard universe

```
should_transfer_files = YES
```

```
when_to_transfer_output = ON_EXIT
```

R and Condor

- typical R running sequence
 - looks for some environment variables
 - loads in some initialization files
 - runs your R script and exits
- one R environment variable of interest
 - may want to change TMPDIR (default is /tmp)
- 2 common ways of invoking R
 - `R CMD BATCH infile outfile`
 - restores and saves `.RData`, `.Rhistory`
 - may take up a lot of time and disk space
 - `R --vanilla < infile > outfile`

R and Condor (cont'd)

- currently R doesn't have a Standard universe version
 - big R jobs better run on your own machine

```
nohup R --vanilla < infile > outfile &
```
 - a few dedicated Condor cluster machines are available
 - one solution: divide and conquer (more on it later)
 - Condor team is working on the problem
- tip: use `afs_rseta dir net:cs read` to give condor read permission recursively on a pkg dir
- add `limit coredumpsize 0` to `.cshrc.local` or add `coresize=0` to submit file to prevent core files

Condor Submit Files



- A complete submit file

```
universe           = vanilla
executable         = /unsup/R/bin/R
getenv             = True
arguments          = --vanilla
input              = foo.R
output             = foo.out
error              = foo.err
log                = foo.log
notification       = error
queue
```



Condor Submit Files (cont'd)

- send replicate jobs with `queue num_of_repl`
- jobs in a submit file can have different inputs/outputs

- just add to the previous file

```
input    = foo_2.R
output   = foo_2.out
queue
...
```

- Condor jobs will be in one cluster, save overheads
- use environment variables to pass inputs
 - In submit: `environment = ENV1=VAL1; ...`
 - In R: `Sys.getenv("ENV1")` retrieves the var

Requirements and Rank



- typical one I use in submit file

```
Requirements = Memory>=256  
Rank         = kflops
```

- more complex use

```
Requirements = Memory>=300 && Disk>=400000  
Requirements = machine==hostname.domain  
Requirements = ClockDay==0 || ClockDay==6  
Requirements = Arch == "SUN4x" && \  
                OpSys == "SOLARIS28"  
Rank = 100*kflops + 10*Memory + Disk
```

- complete list of attributes in Condor manual



Requirements and Rank (cont'd)



- check the machines with a given attribute

```
condor_status -constraint kflops
```

- check the ClassAd attributes of a machine

```
condor_status -direct public03 -l
```

```
MyType = "Machine"  
TargetType = "Job"  
Name = "public03.stat.wisc.edu"  
Machine = "public03.stat.wisc.edu"  
Memory = 999  
KFlops = 747665  
...
```



Using Stat Cluster Machines

- 4 machines running now: sclstr1-sclstr4
- to use them
 - add `+RJob=TRUE` to your submit file, and
 - add `Requirements=(PreferR==TRUE)` or `Rank=(PreferR==TRUE)` to submit file
- to check machine status

```
condor_status -const PreferR==TRUE
```

Check the Progress of Your Jobs



- the most common command to use

```
condor_q -submitter user
```

- you can always look at your log files

- graphical display of your job status (CondorView)

```
http://pumori.cs.wisc.edu/condor-view-applet/
```

- if job idle in a queue for a long time

```
condor_q -analyze jobid
```

- check all the machines running your jobs

```
condor_status -const 'RemoteUser=="user@domain"'
```



Commands for Managing Jobs



- remove Condor job(s)

```
condor_rm cluster.process/cluster
condor_rm user
condor_rm -constraint some_expr
```

- put a job on hold/release with

```
condor_hold/condor_release
```

- adjust the priorities of your jobs with

```
condor_prio
```

- vacate jobs running on your workstation with

```
condor_vacate
```



Deciphering Log File and etc.

- return code 0 means successful completion
- any other code means some kind of error
- always specify log and error in your submit files
 - log file for Condor execution
 - error file for your code execution
- jobs can share the same log and error file
- some possible errors: file access, error in your code, insufficient disk/memory, ...

Condor DAG: Solution to Big Jobs

- very useful for running jobs with dependencies
 - e.g., $B \leftarrow A$, $D \leftarrow C$
- the simplest DAG, just one node
- runs a scheduler universe job, which manages your DAG jobs (can be in any universe)
- can be used to run your big Vanilla jobs, if you know how to divide them into smaller ones
 - use files to pass input/output between jobs
- side benefit: can be utilized to implement some high level parallel computing

Condor DAG (cont'd)

- need a DAG file besides job submit files
 - support `job`, `script`, `parent`, `retry`, `var`
 - submit with `condor_submit_dag dagfile`
- DAG jobs can share submit files, one DAG example

```
Job      A1      dummy_script.submit
Job      A2      dummy_script.submit
Job      B1      dummy_script.submit
PARENT  A1 A2    CHILD  B1
VARS    A1      i="1" j="2"
VARS    A2      i="1" j="3"
VARS    B1      i="1" j="0"
```

Condor DAG (cont'd)



- submit file for the previous DAG

```
Universe           = vanilla
Executable         = dummy_script.sh
Arguments          = $(i) $(j)
output             = dummy_$(i)_$(j).out
error              = dummy_script.err
log                = dummy_script.log
notification       = never
Queue
```

- to query `condor_q -dag -submitter user`
- to remove `condor_rm dagman.condor.id`



Enhancing Condor with Scripting

- a little scripting goes a long way (Shell or Perl)
- I use Perl for generating submit files and DAG files
- can submit and manage jobs within script too
- Perl is much like C but with its own peculiarity (such as lots of \$, @, and etc.)
- scripting is good for files with a lot of repetitive typing and files with regular structure, e.g.
 - 100 DAG jobs with the same submit file
 - submit file with lots of jobs having similar inputs/outputs

A Real DAG Example

- the DAG file

```
Job      A1      tps_seg_slice_s1.submit
Job      B1      tps_seg_slice_s2.submit
...
Job      A35     tps_seg_slice_s1.submit
Job      B35     tps_seg_slice_s2.submit
PARENT   A1      CHILD    B1
...
PARENT   A35     CHILD    B35
Vars     A1      i="1"   j="1"
...
Vars     B35     i="5"   j="7"
```

Real DAG Example (cont'd)

• the submit file

```
Universe          = vanilla
Executable        = /unsup/R/bin/R
...
Environment       = i=$(i);j=$(j)
Should_transfer_files = YES
When_to_transfer_output = ON_EXIT
Transfer_input_files = file_lists
Log               = tps_seg_slice.log
Error             = tps_seg_slice.err
Notification      = never
Queue
```

Perl Code Snippet

```
$dagfname      = '>tps_seg_slice.dag';
$submitfname1  = 'tps_seg_slice_s1.submit';
$submitfname2  = 'tps_seg_slice_s2.submit';
open(fh, $dagfname);
for ($i=1; $i<=5; $i++) {
    for ($j=1; $j<=7; $j++) {
        $jobid = ($i-1)*7 + $j;
        printf(fh "%s\t%s\t%s\n", 'Job', 'A'."$jobid",
                $submitfname1);
        printf(fh "%s\t%s\t%s\n", 'Job', 'B'."$jobid",
                $submitfname2);
    }
}
close(fh);
```

Recap of the Talk

- Condor jobs should be batch ready
- 2 often used Condor universes: Vanilla and Standard
- AFS: always set access permission; Local FS: add lines to submit file for Vanilla jobs
- invoke R with `--vanilla` flag
- 2 useful command: `condor_q` & `condor_status`
- DAG an important tool of Condor
- Perl scripting enhances Condor

Acknowledgements & Resources



- Condor homepage
 - `http://www.cs.wisc.edu/condor/`
- 2 related talks
 - Condor Week 2004 talk by Nick LeRoy
 - Condor Tutorial by Jun Yan (Former student)
- Resources
 - Condor manual and FAQ
 - Condor mailing list `condor-users`
 - Google for answers
 - email `condor-admin@cs.wisc.edu`

