Batch Scheduling and Resource Management

Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

October 18, 2007

Background

- The earliest computers could run only one program at a time.
- Programs and data were written on punched cards.
- Decks of cards were submitted in batches.
- These batches were placed in a queue and run one at a time.
- Later, time-sharing allowed multiple simultaneous interactive users.
- Batch processing, or batch queueing, is still useful:
    - Two programs running simultaneously can slow each other down.
    - On a single core the slowdown will be at least a factor of two.
    - It can be *much* more with heavy memory or I/O use.
    - Programs may fail due to insufficient memory.

Batch System Features

- Basic batch systems provide:
  - A means of placing jobs on a queue.
  - Some means of examining the queue.
  - A way to remove jobs.
  - Jobs are run when resources are available.
- Additional features provided by some systems:
  - Load balancing across multiple processors
  - Management of resource usage (memory in particular).
  - Protection against runaway jobs.
  - A priority system.
  - Ways of managing parallel jobs.
- More sophisticated features:
  - Checkpointing, suspending, resuming, moving running programs.
  - Integration with grid computing frameworks.

Issues in Managing Parallel Jobs

- Need to be able to request a certain number of processors.
- Job can only run once the required number of processors is available.
- The system has to enforce processor limit.
- System, program/framework need to agree on machines to use.
  - For Lam, could have system write a host file, run `lamboot`.
  - For PVM, could do something similar.
  - System could also provide its own LAM/PVM daemon.
- Open MPI (successor to LAM?) has support built in for
  - SLURM
  - Xgrid
  - SGE (N1)

  Support for others, such as OpenPBS is available but optional.

Some Open Source Batch Systems

- **PBS** and **OpenPBS**. Originally developed for NASA.
- **SLURM**. Developed at Lawrence Livermore National Laboratory.
- **Sun Grid Engine**. Originally from Sun; commercial version is **N1**.
- **Xgrid**. From Apple.
- **Maui Cluster Scheduler**. Commercial version is **Moab**.
- **Condor**. From Computer Science at Wisconsin.

Condor

- Currently Condor is the batch system available on beowulf.
- Some features:
  - Originally developed for scavenging free cycles from workstations.
  - Can support checkpointing and job migration.
  - Requires compilation against Condor libraries.
  - Can be used as scheduler for *vanilla* jobs.
  - PVM jobs are also supported but seem to require some adjusting.
  - LAM jobs are now supported (as of yesterday).
  - Integrates with the Globus grid computing toolkit.
- Previous current version was from the 6.6 series.
- Has just ben upgraded to 6.8.

Basic Condor Usage

- Basic use:
  - prepare a submission script
  - submit the script
  - check the queue periodically
  - or check your email to see if the job is done
- Some commands:
  - `condor_submit` for submitting a job
  - `condor_q` for examining the queue
  - `condor_rm` for removing a job
  - `condor_status` for examining available pool
- Condor universes:
  - standard — supports checkpointing, requires special compilation
  - vanilla — no restrictions
  - PVM
  - MPI — only MPICH 1.2; no longer available in 6.8
  - parallel — available in 6.8; use this for LAM

Submitting a Condor Job

- It is a good idea to create a new directory for your job.
- Place in that directory any files for the job along with a submit file.
- Run the `condor_submit` command from that directory.
- Some submit file commands:
  - `executable`: name of the script or binary file to run. One per file. Path name can be absolute or relative to the current directory.
  - `arguments`: command line arguments for the executable
  - `environment`: `name=value` pairs separated by semicolons.
  - `universe`: most likely `vanilla`, `PVM`, or `parallel`
  - `input`: file(s) for standard input
  - `output`: file(s) for standard output
  - `error`: file(s) for standard error
  - `log`: file for log messages from Condor
  - `queue`: place one or more jobs on the queue
  - `notification = Never` to turn off email notification
- There are many others but these are the most important ones.

Some Simple Examples

Submit file `sub-sleep` for a single job that sleeps for 5 seconds:

```
executable = /bin/sleep
arguments  = 5
universe   = vanilla
output     = out
error      = err
log        = log
notification = Never
queue
```

Submit the job with

```
condor_submit sub-sleep
```

and check it with

```
condor_q
```

Examples are available on line.

Some Simple Examples

Submit file sub-hostname for two jobs computing the hostname of the executing machine:

```
executable = /bin/hostname
universe   = vanilla
output     = out.$(Process)
error      = err.$(Process)
log        = log
notification = Never
queue 2
```

This produces two separate output files and two error files.

Some Simple Examples

Submit file `sub-R` for running two R jobs:

```
environment= R_LIBS=/cluster/statsoft/Rlibs
executable = /usr/bin/R
arguments  = --slave
universe   = vanilla
input      = in.$(Process)
output     = out.$(Process)
error      = err.$(Process)
log        = log
notification = Never
queue 2
```

This uses two separate input files, `in.0` and `in.1`. File `in.0` looks like

```
Sys.info()["nodename"]
.libPaths()
print(0)
```

and `in.1` looking like

```
Sys.info()["nodename"]
.libPaths()
print(1)
```

Some Simple Examples

Another approach is to use the submit file `sub-R2` given by

```
environment= R_LIBS=/cluster/statsoft/Rlibs
executable = /usr/bin/R
arguments  = --slave --args $(Process)
universe   = vanilla
input      = in
output     = out.$(Process)
error      = err.$(Process)
log        = log
notification = Never
queue 2
```

and the common input file `in` given by

```
Sys.info()["nodename"]
.libPaths()
print(commandArgs(TRUE))
```

Some Simple Examples

A snow/LAM job using submit file `sub-snow-lam`

```
environment= R_LIBS=/cluster/statsoft/Rlibs
executable = /cluster/condor/condor/etc/examples/lamscript
arguments  = RMPISNOW
machine_count = 3
universe   = parallel
input      = in-snow-lam
output     = out
error      = err
log        = log
notification = Never
queue
```

and the input file `in-snow-lam` given by

```
cl <- makeCluster()
clusterCall(cl, function() Sys.info()["nodename"])
stopCluster(cl)
```