Profiling Applications to Choose the Right Computing Infrastructure plus Batch Management with HTCondor

DOSAR

Original slides by: Rob Quick rquick@iu.edu
Content Contributed by the University of Wisconsin Condor Team and Scot Kronenfeld
Follow Along at:

Some thoughts on the exercises

- It’s okay to move ahead on exercises if you have time
- It’s okay to take longer on them if you need to
- If you move along quickly, try the “On Your Own” sections and “Challenges”
Most important!

- Please ask us questions!
  - during the lectures
  - during the exercises
  - during the breaks
  - during the meals
  - over dinner
  - via email after we depart (see below)

- If I don’t know, I’ll find the right person to answer your question.
Goals for this session

- Profiling your application
- Picking the appropriate resources
- Understand the basics of HTCondor
Let’s take one step at a time

- Can you run one job on one computer?
- Can you run one job on another computer?
- Can you run 10 jobs on a set of computers?
- Can you run a multiple job workflow?
- How do we put this all together?

This is the path we’ll take in the school
What does the user provide?

• A “headless job”
  – Not interactive/no GUI: how could you interact with 1000 simultaneous jobs?
• A set of input files
• A set of output files
• A set of parameters (command-line arguments)
• Requirements:
  – Ex: My job requires at least 2GB of RAM
  – Ex: My job requires Linux
• Control/Policy:
  – Ex: Send me email when the job is done
  – Ex: Job 2 is more important than Job 1
  – Ex: Kill my job if it runs for more than 6 hours
What does the system provide?

- Methods to:
  - Submit/Cancel job
  - Check on state of job
  - Check on state of available computers

- Processes to:
  - Reliably track set of submitted jobs
  - Reliably track set of available computers
  - Decide which job runs on which computer
  - Manage a single computer
  - Start up a single job
Gedankenexperiment

• Let’s assume you have a ‘large job’
  – What factors could make it large?
• Large Data Input or Output or both
• Needs to do heavy calculation
• Needs a lot of memory
• Needs to communicate with other jobs (whether required or not)
• Reads and writes a lot of data/files
• Heavy graphics processing
• Any combination of any of the above
There is no “One Size Fits All Solution”

• But some solutions are more “Open” than others.
  – Local Laptop/Desktop
  – Local Cluster
  – HPC System
  – Shared HTC Resources
  – Clouds
Why is HTC hard?

• The HTC system has to keep track of:
  – Individual tasks (a.k.a. jobs) & their inputs
  – Computers that are available
• The system has to recover from failures
  – There will be failures! Distributed computers means more chances for failures.
• You have to share computers
  ▪ Sharing can be within an organization, or between orgs
  ▪ So you have to worry about security
  ▪ And you have to worry about policies on how you share
• If you use a lot of computers, you have to handle variety:
  – Different kinds of computers (arch, OS, speed, etc..)
  – Different kinds of storage (access methodology, size, speed, etc…)
  – Different networks interacting (network problems are hard to debug!)
Surprise! HTCondor does this (and more)

• Methods to:
  – Submit/Cancel job. `condor_submit/condor_rm`
  – Check on state of job. `condor_q`
  – Check on state of avail. computers. `condor_status`

• Processes to:
  – Reliably track set of submitted jobs. `schedd`
  – Reliably track set of avail. computers. `collector`
  – Decide which job runs on where. `negotiator`
  – Manage a single computer `startd`
  – Start up a single job `starter`
But not only Condor

• You can use other systems:
  – PBS/Torque
  – Oracle Grid Engine (né Sun Grid Engine)
  – LSF
  – SLURM
  – …

• But I won’t cover them.
  – My experience is with Condor
  – My bias is with Condor
  – Overlays exist

• What should you learn at the school?
  – How do you think about Computing Resources?
  – How can you do your science with HTC?
  – … For now, learn it with Condor, but you can apply it to other systems.
A brief introduction to Condor

• Please note, we will only scratch the surface of Condor:
  – We won’t cover MPI, Master-Worker, advanced policies, site administration, security mechanisms, submission to other batch systems, virtual machines, cron, high-availability, computing on demand, containers.
Condor Takes Computers... ...And matches them

I need a Mac!

\[ E = mc^2 \]
\[ = 1\text{kg} \times \left(3 \times 10^8 \text{ ms}^{-1}\right)^2 \]
\[ = 1\text{kg} \times \left(3 \times 10^8 \text{ ms}^{-1}\right)^2 \times \left(3 \times 10^8 \text{ ms}^{-1}\right) \]
\[ = 1\text{kg} \times \left(9 \times 10^{16} \text{ m}^2\text{s}^{-2}\right) \]
\[ = 1 \times \left(9 \times 10^{16}\right)\text{kg m}^2\text{s}^{-2} \]
\[ = 9 \times 10^{16}\text{ J} \]

I need a Linux box with 2GB RAM!
Quick Terminology

• **Cluster**: A dedicated set of computers not for interactive use

• **Pool**: A collection of computers used by Condor
  – May be dedicated
  – May be interactive

• **Remember**:  
  – Condor can manage a cluster in a machine room  
  – Condor can use desktop computers  
  – Condor can access remote computers  
  – HTC uses all available resources
Matchmaking

• Matchmaking is fundamental to Condor
• Matchmaking is two-way
  – Job describes what it requires:
    I need Linux && 8 GB of RAM
  – Machine describes what it requires:
    I will only run jobs from the Physics department
• Matchmaking allows preferences
  – I need Linux, and I prefer machines with more memory but will run on any machine you provide me
Why Two-way Matching?

• Condor conceptually divides people into three groups:
  – Job submitters
  – Computer owners
  – Pool (cluster) administrator

• All three of these groups have preferences

May or may not be the same people
ClassAds

- ClassAds state facts
  - My job’s executable is analysis.exe
  - My machine’s load average is 5.6
- ClassAds state preferences
  - I require a computer with Linux
- ClassAds are extensible
  - They say whatever you want them to say
Example ClassAd

MyType = "Job" ← String
TargetType = "Machine"
ClusterId = 1377 ← Number
Owner = "roy"
Cmd = "analysis.exe"
Requirements =
   (Arch == "INTEL")
   && (OpSys == "LINUX")
   && (Disk >= DiskUsage)
   && ((Memory * 1024) >= ImageSize)
   ...

Schema-free ClassAds

- Condor imposes some schema
  - Owner is a string, ClusterID is a number…
- But users can extend it however they like, for jobs or machines
  - AnalysisJobType = “simulation”
  - HasJava_1_6 = TRUE
  - ShoeLength = 10
- Matchmaking can use these attributes
  - Requirements = OpSys == "LINUX"
    && HasJava_1_6 == TRUE
Don’t worry

• You won’t write ClassAds (usually)
  – You’ll create a simple submit file
  – Condor will write the ClassAd
  – You can extend the ClassAd if you want to

• You won’t write requirements (usually)
  – Condor writes them for you
  – You can extend them
  – In some environments you provide attributes instead of requirements expressions
Matchmaking diagram

Matchmaking Service

condor_schedd

Job queue service

Matchmaker

Negotiator

Collector

Information service

1

2

3
Why do jobs fail?

- The computer running the job fails
  - Or the network, or the disk, or the OS, or…
- Your job might be *preempted*:
  - Condor decides your job is less important than another, so your job is stopped and another started.
  - Not a “failure” per se, but it may feel like it to you.
Reliability

• When a job fails or is preempted:
  – It stays in the queue (on the schedd)
  – A note is written to the job log file
  – It reverts to “idle” state
  – It is eligible to be matched again

• Relax! Condor will run your job again
Access to data in Condor

- **Option #1: Shared filesystem**
  - Simple to use, but make sure your filesystem can handle the load

- **Option #2: Condor’s file transfer**
  - Can automatically send back changed files
  - Atomic transfer of multiple files
  - Can be encrypted over the wire
  - Most common for small applications and data

- **Option #3: Remote I/O**
Condor File Transfer

• ShouldTransferFiles = YES
  – Always transfer files to execution site
• ShouldTransferFiles = NO
  – Rely on a shared filesystem
• ShouldTransferFiles = IF_NEEDED
  – Will automatically transfer the files if needed

Universe = vanilla
Executable = my_job
Log = my_job.log
ShouldTransferFiles = YES
Transfer_input_files = dataset$(Process), common.data
Queue 600
Condor File Transfer with URLs

- Transfer_input_files can be a URL
  For example:

  ```
  transfer_input_files = http://www.example.com/input.data
  ```
Clusters & Processes

• One submit file can describe lots of jobs
  – All the jobs in a submit file are a *cluster* of jobs
  – Yeah, same term as a cluster of computers

• Each cluster has a unique “cluster number”

• Each job in a cluster is called a “process”

• A Condor “job ID” is the cluster number, a period, and the process number (“20.1”)

• A cluster is allowed to have one or more processes.
  – There is always a cluster for every job
The $(Process) macro

• The initial directory for each job can be specified as run_${Process}, and instead of submitting a single job, we use “Queue 600” to submit 600 jobs at once

• The $(Process) macro will be expanded to the process number for each job in the cluster (0 - 599), so we’ll have “run_0”, “run_1”, … “run_599” directories

• All the input/output files will be in different directories!
Example of \$(Process)\$

# Example condor_submit input file that defines
# a cluster of 600 jobs with different directories
Universe   = vanilla
Executable = my_job
Log        = my_job.log
Arguments  = -arg1 -arg2
Input      = my_job.stdin
Output     = my_job.stdout
Error      = my_job.stderr
InitialDir = run_$\$(Process)$
Queue 600

run_0 ... run_599
Creates job 3.0 ... 3.599
More $(Process)$

- You can use $(Process)$ anywhere:

  **Universe** = vanilla
  **Executable** = my_job
  **Log** = my_job.$(Process).log
  **Arguments** = -randomseed $(Process)$
  **Input** = my_job.stdin
  **Output** = my_job.stdout
  **Error** = my_job.stderr
  **InitialDir** = run_$(Process)$

  **Queue** 600
Sharing a directory

• You don’t have to use separate directories.

• $(Cluster)$ will help distinguish runs

Universe = vanilla
Executable = my_job
Arguments = -randomseed $(Process)
Input = my_job.input.$(Process)

Output = my_job.stdout.$(Cluster).$(Process)
Error = my_job.stderr.$(Cluster).$(Process)
Log = my_job.$(Cluster).$(Process).log

Queue 600
Not Only Programming Language

• You ran a C program this morning
• You can also run scripting languages such as bash, python, and perl
• You can also executing programs via the command like R
• There are several different computing environments
• There is a very diverse set of computing jobs
• Matching jobs to resources is key to not wasting resources
• Not all of the available environments are open environments
• Research Computing is Complex
Quick UNIX Refresher
Before We Start

• $ 
• nano, vi, emacs, cat >, etc. 
• source, module, chmod, ls
That was a whirlwind tour!

• Enough with the presentation: let’s use HTCondor!

• Goal: Extend the diversity of our jobs and add some data to the mix.
Questions?

- Feel free to ask us questions now or later:
  - Julia Gray julia.ann.gray@gmail.com
  - Horst Severini severini@ou.edu
  - Pat Skubic pskubic@ou.edu

Exercises start here:

Presentations are also available from this URL.