An Introduction To
Condor
International Summer School
on Grid Computing 2005

Condor Project
Computer Sciences Department
University of Wisconsin-Madison
condor-admin@cs.wisc.edu
http://www.cs.wisc.edu/condor
First…

These slides are available from:

http://www.cs.wisc.edu/~roy/italy-condor/
This Morning’s Condor Topics

- Matchmaking: Finding machines for jobs
- Running a job
- Running a parameter sweep
- Managing sets of dependent jobs
- Master-Worker applications
Part One

Matchmaking:
Finding Machines For Jobs
Finding Jobs for Machines
CoAddition Takes Computers...
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
Matchmaking

› Matchmaking is fundamental to Condor

› Matchmaking is two-way
  • Job describes what it requires:
    - I need Linux & 2 GB of RAM
  • Machine describes what it requires:
    - I need a Mac

› 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
  - Machine owners
  - Pool (cluster) administrator

- All three of these groups have preferences
Machine owner preferences

- I prefer jobs from the physics group
- I will only run jobs between 8pm and 4am
- I will only run certain types of jobs
- Jobs can be preempted if something better comes along (or not)
System Admin Prefs

» When can jobs preempt other jobs?
» Which users have higher priority?
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

- ClassAds are:
  - semi-structured
  - user-extensible
  - schema-free
  - Attribute = Expression

Example:

MyType = "Job"  \textcolor{red}{\textbf{String}}
TargetType = "Machine"
ClusterId = 1377 \textcolor{red}{\textbf{Number}}
Owner = "roy" \textcolor{red}{\textbf{String}}
Cmd = "analysis.exe" \textcolor{red}{\textbf{String}}
Requirements =
  (Arch == "INTEL") \textcolor{red}{\textbf{Boolean}}
  && (OpSys == "LINUX") \textcolor{red}{\textbf{Boolean}}
  && (Disk >= DiskUsage) \textcolor{red}{\textbf{Boolean}}
  && ((Memory * 1024) >= ImageSize) \textcolor{red}{\textbf{Boolean}}
...
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_4 = TRUE
  - ShoeLength = 7
- Matchmaking can use these attributes
  - Requirements = OpSys == "LINUX"
    && HasJava_1_4 == TRUE
Submitting jobs

- Users submit jobs from a computer
  - Jobs described as a ClassAd
  - Each submission computer has a queue
  - Queues are not centralized
  - Submission computer watches over queue
  - Can have multiple submission computers
  - Submission handled by condor_schedd

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]
Advertising computers

- Machine owners describe computers
  - Configuration file extends ClassAd
  - ClassAd has dynamic features
    - Load Average
    - Free Memory
    - ...
  - ClassAds are sent to Matchmaker

ClassAd
Type = "Machine"
Requirements = "...

Matchmaker (Collector)
Matchmaking

- Negotiator collects list of computers
- Negotiator contacts each schedd
  - What jobs do you have to run?
- Negotiator compares each job to each computer
  - Evaluate requirements of job & machine
  - Evaluate in context of both ClassAds
  - If both evaluate to true, there is a match
- Upon match, schedd contacts execution computer
Matchmaking diagram

condor_schedd

Queue

Matchmaker

| Negotiator | Collector |

1

2

3
Condor processes

➤ **Master:** Takes care of other processes
➤ **Collector:** Stores ClassAds
➤ **Negotiator:** Performs matchmaking
➤ **Schedd:** Manages job queue
➤ **Shadow:** Manages job (submit side)
➤ **Startd:** Manages computer
➤ **Starter:** Manages job (execution side)
Some notes

› Exactly one negotiator/collector per pool
› Can have many schedds (submitters)
› Can have many startds (computers)
› A machine can have any combination
  • Dedicated cluster: maybe just startds
  • Shared workstations: schedd + startd
  • Personal Condor: everything

http://www.cs.wisc.edu/condor
Our Condor Pool

› Each student machine has
  • Schedd (queue)
  • Startd (with two virtual machines)

› Several servers
  • Most: Only a startd
  • One: Startd + collector/negotiator

› At your leisure:
  • condor_status
# Our Condor Pool

<table>
<thead>
<tr>
<th>Name</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>ActivityTime</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:vm1@server4.g">vm1@server4.g</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+03:45:08</td>
</tr>
<tr>
<td><a href="mailto:vm2@server4.g">vm2@server4.g</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+03:45:05</td>
</tr>
<tr>
<td><a href="mailto:vm1@server5.g">vm1@server5.g</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+03:40:08</td>
</tr>
<tr>
<td><a href="mailto:vm2@server5.g">vm2@server5.g</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+03:40:05</td>
</tr>
<tr>
<td><a href="mailto:vm1@ws-01.gs">vm1@ws-01.gs</a>. LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+00:02:45</td>
</tr>
<tr>
<td><a href="mailto:vm2@ws-01.gs">vm2@ws-01.gs</a>. LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+00:02:46</td>
</tr>
<tr>
<td><a href="mailto:vm1@ws-03.gs">vm1@ws-03.gs</a>. LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+02:30:24</td>
</tr>
<tr>
<td><a href="mailto:vm2@ws-03.gs">vm2@ws-03.gs</a>. LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+02:30:20</td>
</tr>
<tr>
<td><a href="mailto:vm1@ws-04.gs">vm1@ws-04.gs</a>. LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.080</td>
<td>501</td>
<td>0+03:30:09</td>
</tr>
<tr>
<td><a href="mailto:vm2@ws-04.gs">vm2@ws-04.gs</a>. LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>501</td>
<td>0+03:30:05</td>
</tr>
</tbody>
</table>

...
Evolution of ClassAds

▶ **ClassAds are very powerful**
  - Schema-free, user-extensible, ...

▶ **ClassAds could be fancier**
  - No lists
  - No nested ClassAds
  - No functions
  - Ad-hoc evaluation semantics
New ClassAds

Future versions of Condor will have new ClassAds to solve these problems

```c
[ Type = "Machine";
  Friends = {"alain", "miron", "peter"};
  LoadAverages = [ OneMinute = 3; FiveMinute=2.0];
  Requirements = member(other.name, Friends);
  ...
]
```

- **list**
- **Nested classad**
- **Built-in function**
Summary

- Condor uses ClassAd to represent state of jobs and machines
- Matchmaking operates on ClassAds to find matches
- Users and machine owners can specify their preferences

http://www.cs.wisc.edu/condor
Let’s take a break!
Part Two

Running a Condor Job
Getting Condor

› Available as a free download from http://www.cs.wisc.edu/condor

› Download Condor for your operating system
  • Available for many UNIX platforms:
    • Linux, Solaris, HPUX, IRIX, Tru64...
  • Also for Windows
Condor Releases

- Naming scheme similar to the Linux Kernel...

- **Major.minor.release**
  - Stable: Minor is even (a.b.c)
    - Examples: 6.4.3, 6.6.8, 6.6.9
    - Very stable, mostly bug fixes
  - Developer: Minor is odd (a.b.c)
    - New features, may have some bugs
    - Examples: 6.5.5, 6.7.5, 6.7.6

- Today’s releases:
  - Stable: 6.6.10
  - Development: 6.7.9

http://www.cs.wisc.edu/condor
Try out Condor:
Use a Personal Condor

- **Condor:**
  - on your own workstation
  - no root access required
  - no system administrator intervention needed

- We'll try this during the exercises

[http://www.cs.wisc.edu/condor](http://www.cs.wisc.edu/condor)
Personal Condor?!

What’s the benefit of a Condor Pool with just one user and one machine?
Your Personal Condor will ...

- keep an eye on your jobs and will keep you posted on their progress
- implement your policy on the execution order of the jobs
- keep a log of your job activities
- add fault tolerance to your jobs
- implement your policy on when the jobs can run on your workstation

http://www.cs.wisc.edu/condor
After Personal Condor…

› When a Personal Condor pool works for you…
  • Convince your co-workers to add their computers to the pool
  • Add dedicated hardware to the pool
Four Steps to Run a Job

1. Choose a Universe for your job
2. Make your job batch-ready
3. Create a submit description file
4. Run condor_submit
1. Choose a Universe

- There are many choices
  - Vanilla: any old job
  - Standard: checkpointing & remote I/O
  - Java: better for Java jobs
  - MPI: Run parallel MPI jobs
  - ...

- For now, we'll just consider vanilla
2. Make your job batch-ready

▷ Must be able to run in the background: no interactive input, windows, GUI, etc.

▷ **Can still use** `stdin`, `stdout`, and `stderr` (the keyboard and the screen), but files are used for these instead of the actual devices

▷ Organize data files
3. Create a Submit Description File

› A plain ASCII text file
  • Not a ClassAd
  • But condor_submit will make a ClassAd from it

› Condor does **not** care about file extensions

› Tells Condor about your job:
  • Which executable, universe, input, output and error files to use, command-line arguments, environment variables, any special requirements or preferences

http://www.cs.wisc.edu/condor
Simple Submit Description File

# Simple condor_submit input file
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe   = vanilla
Executable = analysis
Log        = my_job.log
Queue
4. Run `condor_submit`

- You give `condor_submit` the name of the submit file you have created:

```
condor_submit my_job.submit
```

- `condor_submit` parses the submit file, checks for it errors, and creates a ClassAd that describes your job.
The Job Queue

- `condor_submit` sends your job’s ClassAd to the schedd
  - Manages the local job queue
  - Stores the job in the job queue
    - Atomic operation, two-phase commit
    - “Like money in the bank”
- View the queue with `condor_q`
An example submission

% condor_submit my_job.submit

Submitting job(s).
1 job(s) submitted to cluster 1.

% condor_q

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>roy</td>
<td>7/6 06:52</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>analysis</td>
</tr>
</tbody>
</table>

1 jobs; 1 idle, 0 running, 0 held

%
Some details

- **Condor sends you email about events**
  - Turn it off: Notification = Never
  - Only on errors: Notification = Error

- **Condor creates a log file (user log)**
  - “The Life Story of a Job”
  - Shows all events in the life of a job
  - Always have a log file
  - Specified with: Log = filename

http://www.cs.wisc.edu/condor
Sample Condor User Log

000 (0001.000.000) 05/25 19:10:03 Job submitted from host: <128.105.146.14:1816>
...

001 (0001.000.000) 05/25 19:12:17 Job executing on host: <128.105.146.14:1026>
...

005 (0001.000.000) 05/25 19:13:06 Job terminated.

(1) Normal termination (return value 0)
    Usr 0 00:00:37, Sys 0 00:00:00  - Run Remote Usage
    Usr 0 00:00:00, Sys 0 00:00:05  - Run Local Usage
    Usr 0 00:00:37, Sys 0 00:00:00  - Total Remote Usage
    Usr 0 00:00:00, Sys 0 00:00:05  - Total Local Usage

9624  - Run Bytes Sent By Job
7146159  - Run Bytes Received By Job
9624  - Total Bytes Sent By Job
7146159  - Total Bytes Received By Job
...

http://www.cs.wisc.edu/condor
More Submit Features

# Example condor_submit input file

Universe     = vanilla
Executable   = /home/roy/condor/my_job.condor
Log          = my_job.log
Input        = my_job.stdin
Output       = my_job.stdout
Error        = my_job.stderr
Arguments    = -arg1 -arg2
InitialDir   = /home/roy/condor/run_1
Queue
Using condor\_rm

➢ If you want to remove a job from the Condor queue, you use \texttt{condor\_rm}

➢ You can only remove jobs that you own (you can’t run \texttt{condor\_rm} on someone else’s jobs unless you are root)

➢ You can give specific job ID’s (cluster or cluster.proc), or you can remove all of your jobs with the “-d’’ option.

    • \texttt{condor\_rm 21.1} \hspace{1cm} \textbullet \hspace{0.5cm} \text{Removes a single job}
    • \texttt{condor\_rm 21} \hspace{1cm} \textbullet \hspace{0.5cm} \text{Removes a whole cluster}
## condor_status

<table>
<thead>
<tr>
<th>Name</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>ActvtyTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>haha.cs.wisc.</td>
<td>IRIX65</td>
<td>SGI</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.198</td>
<td>192</td>
<td>0+00:00:04</td>
</tr>
<tr>
<td>antipholus.cs</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.020</td>
<td>511</td>
<td>0+02:28:42</td>
</tr>
<tr>
<td>coral.cs.wisc</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Claimed</td>
<td>Busy</td>
<td>0.990</td>
<td>511</td>
<td>0+01:27:21</td>
</tr>
<tr>
<td>doc.cs.wisc.e</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.260</td>
<td>511</td>
<td>0+00:20:04</td>
</tr>
<tr>
<td>dsonokwa.cs.w</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Claimed</td>
<td>Busy</td>
<td>0.810</td>
<td>511</td>
<td>0+00:01:45</td>
</tr>
<tr>
<td>ferdinand.cs.</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Claimed</td>
<td>Suspended</td>
<td>1.130</td>
<td>511</td>
<td>0+00:00:55</td>
</tr>
<tr>
<td>vm1@pinguino.</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>255</td>
<td>0+01:03:28</td>
</tr>
<tr>
<td>vm2@pinguino.</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.190</td>
<td>255</td>
<td>0+01:03:29</td>
</tr>
</tbody>
</table>
How can my jobs access their data files?
Access to Data in Condor

- Use shared filesystem if available
- No shared filesystem?
  - **Condor can transfer files**
    - Can automatically send back changed files
    - Atomic transfer of multiple files
    - Can be encrypted over the wire
  - Remote I/O Socket
  - Standard Universe can use remote system calls (more on this later)

http://www.cs.wisc.edu/condor
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 the submit and execute machine are not in the same FileSystemDomain

Universe = vanilla
Executable = my_job
Log = my_job.log
ShouldTransferFiles = IF_NEEDED
Transfer_input_files = dataset$(Process), common.data
Transfer_output_files = TheAnswer.dat
Queue 600

http://www.cs.wisc.edu/condor
Some of the machines in the Pool do not have enough memory or scratch disk space to run my job!
Specify Requirements!

- An expression (syntax similar to C or Java)
- Must evaluate to True for a match to be made

Universe = vanilla
Executable = my_job
Log = my_job.log
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Queue 600
Specify Rank!

- All matches which meet the requirements can be sorted by preference with a Rank expression.
- Higher the Rank, the better the match

Universe = vanilla
Executable = my_job
Log = my_job.log
Arguments = -arg1 -arg2
InitialDir = run_$\$(Process)
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 600
We’ve seen how Condor can:

... keeps an eye on your jobs and will keep you posted on their progress
... implements your policy on the execution order of the jobs
... keeps a log of your job activities
My jobs run for 20 days…

▷ What happens when they get pre-empted?
▷ How can I add fault tolerance to my jobs?
Condor’s **Standard Universe** to the rescue!

- Condor can support various combinations of features/environments in different “Universes”
- Different Universes provide different functionality for your job:
  - *Vanilla* - Run any Serial Job
  - *Scheduler* - Plug in a scheduler
  - **Standard** - Support for transparent process checkpoint and restart

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

- Condor’s Process Checkpointing mechanism saves the entire state of a process into a checkpoint file
  - Memory, CPU, I/O, etc.
- The process can then be restarted from right where it left off
- Typically no changes to your job’s source code needed - however, your job must be relinked with Condor’s Standard Universe support library

http://www.cs.wisc.edu/condor
Relinking Your Job for Standard Universe
To do this, just place "condor_compile" in front of the command you normally use to link your job:

% condor_compile gcc -o myjob myjob.c
- OR -
% condor_compile f77 -o myjob filea.f fileb.f
- OR -
% condor_compile make -f MyMakefile

http://www.cs.wisc.edu/condor
Limitations of the Standard Universe

- Condor’s checkpointing is not at the kernel level. Thus in the Standard Universe the job may not:
  - Fork()
  - Use kernel threads
  - Use some forms of IPC, such as pipes and shared memory

- Many typical scientific jobs are OK
When will Condor checkpoint your job?

- Periodically, if desired
  - For fault tolerance
- When your job is preempted by a higher priority job
- When your job is vacated because the execution machine becomes busy
- When you explicitly run `condor_checkpoint`, `condor_vacate`, `condor_off` or `condor_restart` command

http://www.cs.wisc.edu/condor
Remote I/O Socket

› Job can request that the condor_starter process on the execute machine create a **Remote I/O Socket**

› Used for online access of file on submit machine – *without Standard Universe.*
  • Use in Vanilla, Java, ...

› Libraries provided for Java and for C, e.g.:
  Java: FileInputStream -> ChirpInputStream
     C: open() -> chirp_open()
Job
Fork
starter
shadow
Home
File
System
I/O Library
I/O Server
Local System Calls
Secure Remote I/O
I/O Proxy
Submission Host
Execution Host
Local I/O (Chirp)
Remote System Calls

- I/O System calls are trapped and sent back to submit machine
- Allows Transparent Migration Across Administrative Domains
  - Checkpoint on machine A, restart on B
- No Source Code changes required
- Language Independent
- Opportunities for Application Steering
  - Example: Condor tells customer process “how” to open files
Job Startup

- Schedd
- Shadow
- Submit
- Startd
- Starter
- Customer Job
  - Condor Syscall Lib
condor_q -io

c01(69)% condor_q -io

-- Submitter: c01.cs.wisc.edu : <128.105.146.101:2996> : c01.cs.wisc.edu

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>READ</th>
<th>WRITE</th>
<th>SEEK</th>
<th>XPUT</th>
<th>BUFSIZE</th>
<th>BLKSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>72.3</td>
<td>edayton</td>
<td>[ no i/o data collected yet ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72.5</td>
<td>edayton</td>
<td>6.8 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>104.0 KB/s</td>
<td>512.0 KB</td>
<td>32.0 KB</td>
</tr>
<tr>
<td>73.0</td>
<td>edayton</td>
<td>6.4 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>140.3 KB/s</td>
<td>512.0 KB</td>
<td>32.0 KB</td>
</tr>
<tr>
<td>73.2</td>
<td>edayton</td>
<td>6.8 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>112.4 KB/s</td>
<td>512.0 KB</td>
<td>32.0 KB</td>
</tr>
<tr>
<td>73.4</td>
<td>edayton</td>
<td>6.8 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>139.3 KB/s</td>
<td>512.0 KB</td>
<td>32.0 KB</td>
</tr>
<tr>
<td>73.5</td>
<td>edayton</td>
<td>6.8 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>139.3 KB/s</td>
<td>512.0 KB</td>
<td>32.0 KB</td>
</tr>
<tr>
<td>73.7</td>
<td>edayton</td>
<td>[ no i/o data collected yet ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 jobs; 0 idle, 0 running, 0 held

http://www.cs.wisc.edu/condor
Condor Job Universes

- Serial Jobs
  - Vanilla Universe
  - Standard Universe
- Scheduler Universe
- Parallel Jobs
  - MPI Universe (soon the Parallel Universe)
  - PVM Universe
- Java Universe

http://www.cs.wisc.edu/condor
Java Universe Job

```
universe = java
executable = Main.class
jar_files = MyLibrary.jar
input = infile
output = outfile
arguments = Main 1 2 3
queue
```

condor_submit
Why not use Vanilla Universe for Java jobs?

- Java Universe provides more than just inserting “java” at the start of the execute line
  - Knows which machines have a JVM installed
  - Knows the location, version, and performance of JVM on each machine
  - Provides more information about Java job completion than just JVM exit code
    - Program runs in a Java wrapper, allowing Condor to report Java exceptions, etc.
Java support, cont.

condor_status -java

<table>
<thead>
<tr>
<th>Name</th>
<th>JavaVendor</th>
<th>Ver</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>aish.cs.wisc.</td>
<td>Sun Microsy</td>
<td>1.2.2</td>
<td>Owner</td>
<td>Idle</td>
<td>0.000</td>
<td>249</td>
</tr>
<tr>
<td>anfrom.cs.wis</td>
<td>Sun Microsy</td>
<td>1.2.2</td>
<td>Owner</td>
<td>Idle</td>
<td>0.030</td>
<td>249</td>
</tr>
<tr>
<td>babe.cs.wisc.</td>
<td>Sun Microsy</td>
<td>1.2.2</td>
<td>Claimed</td>
<td>Busy</td>
<td>1.120</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- **Use:**
  - `condor_submit`
  - `condor_q`
  - `condor_status`

- **Condor can run**
  - Any old program (vanilla)
  - Some jobs with checkpointing & remote I/O (standard)
  - Java jobs with better understanding

- **Files can be accessed via**
  - Shared filesystem
  - File transfer
  - Remote I/O

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

Running a parameter sweep
Clusters and Processes

- If your submit file describes multiple jobs, we call this a “cluster”
- Each cluster has a unique “cluster number”
- Each job in a cluster is called a “process”
  - Process numbers always start at zero
- 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

http://www.cs.wisc.edu/condor
Example Submit Description File for a Cluster

# Example submit description file that defines a cluster of 2 jobs with separate working 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_0
Queue

InitialDir = run_1
Queue

· Becomes job 2.0
· Becomes job 2.1

http://www.cs.wisc.edu/condor
Submitting The Job

% condor_submit my_job.submit-file

Submitting job(s).
2 job(s) submitted to cluster 2.

% condor_q

--- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>frieda</td>
<td>4/15 06:52</td>
<td>0+00:02:11</td>
<td>R</td>
<td>0</td>
<td>0.0</td>
<td>my_job</td>
</tr>
<tr>
<td>2.0</td>
<td>frieda</td>
<td>4/15 06:56</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>my_job</td>
</tr>
<tr>
<td>2.1</td>
<td>frieda</td>
<td>4/15 06:56</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>my_job</td>
</tr>
</tbody>
</table>

3 jobs; 2 idle, 1 running, 0 held

http://www.cs.wisc.edu/condor
Submit Description File for a \textit{BIG} Cluster of Jobs

\begin{itemize}
\item The initial directory for each job can be specified as \verb|run_$(Process)|, and instead of submitting a single job, we use \textquote{Queue 600} to submit 600 jobs at once.
\item The \verb|$(Process)| macro will be expanded to the process number for each job in the cluster (0 - 599), so we'll have \textquote{run_0}, \textquote{run_1}, \ldots \textquote{run_599} directories.
\item All the input/output files will be in different directories!
\end{itemize}
Submit Description File for a BIG Cluster of Jobs

# 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)       #run_0 ... run_599
Queue 600   #Becomes job 3.0 ... 3.599
More \textdollar{(Process)}

\textgreater{} You can use \textdollar{(Process)} anywhere.

\begin{itemize}
\item Universe = vanilla
\item Executable = my\_job
\item Log = my\_job.$(Process).log
\item Arguments = -randomseed $(Process)$
\item Input = my\_job.stdin
\item Output = my\_job.stdout
\item Error = my\_job.stderr
\item InitialDir = run\_$\{(Process)\}$ \cdot run\_0 \ldots run\_599
\item Queue 600 \cdot Becomes job 3.0 \ldots 3.599
\end{itemize}
Sharing a directory

- You don’t have to use separate directories.
- $(Cluster)$ will help distinguish runs

```
Universe       = vanilla
Executable     = my_job
Log            = my_job.$(Cluster).$(Process).log
Arguments      = -randomseed $(Process)
Input          = my_job.input.$(Process)
Output         = my_job.stdout.$(Cluster).$(Process)
Error          = my_job.stderr.$(Cluster).$(Process)
Queue          = 600
```

http://www.cs.wisc.edu/condor
Difficulties with $(Process)$

› Some people want to pass their program $(Process) + 10$ (or another number)

› You can’t do this. 😞

› You can do things like:

  Universe    = vanilla
  Executable  = my_job
  Arguments   = -randomseed 10$(Process)$

  ...
  Queue 600

› 10 is pre-pended to each $(Process)$ Argument
Job Priorities

- Are some of the jobs in your sweep more interesting than others?

- `condor_prio` lets you set the job priority
  - Priority relative to your jobs, not other peoples
  - Condor 6.6: priority can be -20 to +20
  - Condor 6.7: priority can be any integer

- Can be set in submit file:
  - `Priority = 14`
What if you have **LOTS** of jobs?

- **System resources**
  - Each job requires a shadow process
  - Each shadow requires file descriptors and sockets
  - Each shadow requires ports/sockets
  - Set system limits for these to be large

- **Each condor_schedd limits max number of jobs running**
  - Default is 200
  - Configurable

- **Consider multiple submit hosts**
  - You can submit jobs from multiple computers
  - Immediate increase in scalability & complexity

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

› You submit 10 parameter sweeps

› You have five classes of parameters sweeps
  • Call them A, B, C, D, E

› How can you look at the status of jobs that are part of Type B parameter sweeps?
Advanced Trickery cont.

› In your job file:
  • +SweepType = "B"

› You can see this in your job ClassAd
  • condor_q -l

› You can show jobs of a certain type:
  • condor_q -constraint 'SweepType == "B"

› Very useful when you have a complex variety of jobs

› Try this during the exercises!

› Be careful with the quoting...
Part Four

Managing Job Dependencies
DAGMan

Directed Acyclic Graph

Manager

- DAGMan allows you to specify the dependencies between your Condor jobs, so it can manage them automatically for you.

- Example: “Don’t run job B until job A has completed successfully.”
What is a DAG?

- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a node in the DAG.
- Each node can have any number of “parent” or “children” nodes - as long as there are no loops!
Defining a DAG

- A DAG is defined by a `.dag file`, listing each of its nodes and their dependencies:

  Job A a.sub
  Job B b.sub
  Job C c.sub
  Job D d.sub

  Parent A Child B C
  Parent B C Child D
DAG Files….

The complete DAG is five files

One DAG File:

- Job A a.sub
- Job B b.sub
- Job C c.sub
- Job D d.sub

Four Submit Files:

- Universe = Vanilla
- Executable = analysis…

Parent A Child B C
Parent B C Child D
Submitting a DAG

› To start your DAG, just run `condor_submit_dag` with your `.dag` file, and Condor will start a personal DAGMan process which to begin running your jobs:

```shell
% condor_submit_dag diamond.dag
```

› `condor_submit_dag` submits a Scheduler Universe job with DAGMan as the executable.

› Thus the DAGMan daemon itself runs as a Condor job, so you don’t have to baby-sit it.
Running a DAG

- DAGMan acts as a scheduler, managing the submission of your jobs to Condor based on the DAG dependencies.
Running a DAG (cont’d)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.
Running a DAG (cont’d)

In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a "rescue" file with the current state of the DAG.
Recovering a DAG

Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.
Recovering a DAG (cont’d)

Once that job completes, DAGMan will continue the DAG as if the failure never happened.
Finishing a DAG

- Once the DAG is complete, the DAGMan job itself is finished, and exits.
DAGMan & Log Files

› For each job, Condor generates a log file
› DAGMan reads this log to see what has happened
› If DAGMan dies (crash, power failure, etc...)
   • Condor will restart DAGMan
   • DAGMan re-reads log file
   • DAGMan knows everything it needs to know

http://www.cs.wisc.edu/condor
Advanced DAGMan Tricks

- Throttles and degenerative DAGs
- Recursive DAGs: Loops and more
- Pre and Post scripts: editing your DAG
Throttles

- Failed nodes can be automatically retried a configurable number of times
  - Can retry N times
  - Can retry N times, unless a node returns specific exit code

- Throttles to control job submissions
  - Max jobs submitted
  - Max scripts running
Degenerative DAG

- Submit DAG with:
  - 200,000 nodes
  - No dependencies

- Use DAGMan to throttle the jobs
  - Condor is scalable, but it will have problems if you submit 200,000 jobs simultaneously
  - DAGMan can help you get scalability even if you don’t have dependencies
Recursive DAGs

- Idea: any given DAG node can be a script that does:
  1. Make decision
  2. Create DAG file
  3. Call condor_submit_dag
  4. Wait for DAG to exit

- DAG node will not complete until recursive DAG finishes,

- Why?
  - Implement a fixed-length loop
  - Modify behavior on the fly
Recursive DAG

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z
DAGMan scripts

> DAGMan allows pre & post scripts
  • Don’t have to be scripts: any executable
  • Run before (pre) or after (post) job
  • Run on the same computer you submitted from

> Syntax:

  JOB A a.sub
  SCRIPT PRE A before-script $JOB
  SCRIPT POST A after-script $JOB $RETURN
So What?

▷ Pre script can make decisions
  • Where should my job run? (Particularly useful to make job run in same place as last job.)
  • Should I pass different arguments to the job?
  • Lazy decision making

▷ Post script can change return value
  • DAGMan decides job failed in non-zero return value
  • Post-script can look at {error code, output files, etc} and return zero or non-zero based on deeper knowledge.

http://www.cs.wisc.edu/condor
Part Five
Master Worker Applications

(Slides adapted from Condor Week 2005 presentation by Jeff Linderoth)
Why Master Worker?

- An alternative to DAGMan
- DAGMan
  - Create a bunch of Condor jobs
  - Run them in parallel
- Master Worker (MW)
  - Write a bunch of tasks in C++
  - MW uses Condor to run your tasks
  - Don’t worry about the jobs
  - But rewrite your application to fit MW

http://www.cs.wisc.edu/condor
Master Worker Basics

- Master assigns tasks to workers
- Workers perform tasks and report results
- Workers do not communicate (except via master)

- Simple
- Fault Tolerant
- Dynamic

http://www.cs.wisc.edu/condor
Master Worker Toolkit

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- **MW** is a software package that encapsulates these abstractions
  - **API**: C++ abstract classes
  - User writes 10 methods
  - The MWized code will transparently adapt to the dynamic and heterogeneous computing environment
- **MW** also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/\{PVM, Sockets, Files\}
  - Single processor

http://www.cs.wisc.edu/condor
MW’s Layered Architecture

Application classes

MW abstract classes

Resource Mgr  Communication Layer

Underlying infrastructure

API

IPI

Infrastructure Provider’s Interface

http://www.cs.wisc.edu/condor
MW’s Runtime Structure

1. User code adds tasks to the master’s Todo list;
2. Each task is sent to a worker (Todo -> Running);
3. The task is executed by the worker;
4. The result is sent back to the master;
5. User code processes the result (can add/remove tasks).

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

- MWMaster
  - getuserinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()

- MWTask
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()

- MWWorker
  - unpack_worker_init_data()
  - execute_task()
Other MW Utilities

- **MWprintf**
  to print progress, result, debug info, etc;

- **MWDriver**
  to get information, set control policies, etc;

- **RMC:**
  to specify resource requirements, prepare for communication, etc.
Real MW Applications

- **MWFATCOP** (Chen, Ferris, Linderoth)
  
  A branch and cut code for linear integer programming

- **MWMINLP** (Goux, Leyffer, Nocedal)
  
  A branch and bound code for nonlinear integer programming

- **MWQPBB** (Linderoth)
  
  A (simplicial) branch and bound code for solving quadratically constrained quadratic programs

- **MWAND** (Linderoth, Shen)
  
  A nested decomposition based solver for multistage stochastic linear programming

- **MWATR** (Linderoth, Shapiro, Wright)
  
  A trust-region-enhanced cutting plane code for linear stochastic programming and statistical verification of solution quality.

- **MWQAP** (Anstreicher, Brixius, Goux, Linderoth)
  
  A branch and bound code for solving the quadratic assignment problem

http://www.cs.wisc.edu/condor
Example: Nug30

- nug30 (a Quadratic Assignment Problem instance of size 30) had been the “holy grail” of computational QAP research for > 30 years
- In 2000, Anstreicher, Brixius, Goux, & Linderoth set out to solve this problem
- Using a mathematically sophisticated and well-engineered algorithm, they still estimated that we would require **11 CPU years** to solve the problem.

http://www.cs.wisc.edu/condor
# Nug 30 Computational Grid

<table>
<thead>
<tr>
<th>Number</th>
<th>Arch/OS</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>414</td>
<td>Intel/Linux</td>
<td>Argonne</td>
</tr>
<tr>
<td>96</td>
<td>SGI/Irix</td>
<td>Argonne</td>
</tr>
<tr>
<td>1024</td>
<td>SGI/Irix</td>
<td>NCSA</td>
</tr>
<tr>
<td>16</td>
<td>Intel/Linux</td>
<td>NCSA</td>
</tr>
<tr>
<td>45</td>
<td>SGI/Irix</td>
<td>NCSA</td>
</tr>
<tr>
<td>246</td>
<td>Intel/Linux</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>146</td>
<td>Intel/Solaris</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>133</td>
<td>Sun/Solaris</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>190</td>
<td>Intel/Linux</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>94</td>
<td>Intel/Solaris</td>
<td>Georgia Tech</td>
</tr>
<tr>
<td>54</td>
<td>Intel/Linux</td>
<td>Italy (INFN)</td>
</tr>
<tr>
<td>25</td>
<td>Intel/Linux</td>
<td>New Mexico</td>
</tr>
<tr>
<td>12</td>
<td>Sun/Solaris</td>
<td>Northwestern</td>
</tr>
<tr>
<td>5</td>
<td>Intel/Linux</td>
<td>Columbia U.</td>
</tr>
<tr>
<td>10</td>
<td>Sun/Solaris</td>
<td>Columbia U.</td>
</tr>
</tbody>
</table>

- Used tricks to make it look like one Condor pool
  - Flocking
  - Glide-in
- 2510 CPUs total

http://www.cs.wisc.edu/condor
Workers Over Time
Nug30 solved

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wall Clock Time</strong></td>
<td>6 days</td>
</tr>
<tr>
<td></td>
<td>22:04:31 hours</td>
</tr>
<tr>
<td><strong>Avg # Machines</strong></td>
<td>653</td>
</tr>
<tr>
<td><strong>CPU Time</strong></td>
<td>11 years</td>
</tr>
<tr>
<td><strong>Parallel Efficiency</strong></td>
<td>93%</td>
</tr>
</tbody>
</table>
More on MW

- [http://www.cs.wisc.edu/condor/mw](http://www.cs.wisc.edu/condor/mw)
- Version 0.2 is the latest
  - It’s more stable than the version number suggests!
- Mailing list available for discussion
- Active development by the Condor team
I could also tell you about…

› **Condor-G**: Condor’s ability to talk to other Grid systems
  • Globus 2, 3, 4
  • NorduGrid
  • Oracle
  • Condor…

› **Stork**: Treating data placement like computational jobs

› **Nest**: File server with space allocations

› **GCB**: Living with firewalls & private networks

http://www.cs.wisc.edu/condor
But I won’t

› After lunch: Exercises
› Please ask me questions, now or later