Additional DAGMan Features

- Provides other handy features for job management…
  - nodes can have PRE & POST scripts
  - failed nodes can be automatically re-tried a configurable number of times
  - job submission can be “throttled”

We’ve seen how 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?

What if each job needed to run for 20 days? What if I wanted to interrupt a job with a higher priority job?

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 meta-scheduler
  - Standard – Support for transparent process checkpoint and restart

Process Checkpointing

- Condor’s Process Checkpointing mechanism saves all the 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

Relinking Your Job for submission to the 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
```

Limitations in 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
- To free the machine to do a higher priority task (higher priority job, or a job from a user with higher priority)
  - Preemptive-resume scheduling
- When you explicitly run `condor_checkpoint`, `condor_vacate`, `condor_off` or `condor_restart` command

What Condor Daemons are running on my machine, and what do they do?

**condor_master**
- Starts up all other Condor daemons
- If there are any problems and a daemon exits, it restarts the daemon and sends email to the administrator
- Checks the time stamps on the binaries of the other Condor daemons, and if new binaries appear, the master will gracefully shutdown the currently running version and start the new version
- Acts as the server for many Condor remote administration commands:
  - `condor_reconfig`, `condor_restart`, `condor_off`, `condor_on`, `condor_config_val`, etc.

**condor_startd**
- Represents a machine to the Condor system
- Responsible for starting, suspending, and stopping jobs
- Enforces the wishes of the machine owner (the owner's "policy"… more on this soon)

**condor_schedd**
- Represents users to the Condor system
- Maintains the persistent queue of jobs
- Responsible for contacting available machines and sending them jobs
- Services user commands which manipulate the job queue:
  - `condor_submit`, `condor_rm`, `condor_q`, `condor_hold`, `condor_release`, `condor_prio`, ...

**condor_collector**
- Collects information from all other Condor daemons in the pool
  - "Directory Service" / Database for a Condor pool
- Each daemon sends a periodic update called a "ClassAd" to the collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (`condor_status`)

**condor_negotiator**
- Performs "matchmaking" in Condor
- Gets information from the collector about all available machines and all idle jobs
- Tries to match jobs with machines that will serve them
- Both the job and the machine must satisfy each other's requirements

Happy Day! Frieda’s organization purchased a Beowulf Cluster!

- Frieda Installs Condor on all the dedicated Cluster nodes, and configures them with her machine as the central manager…
- Now her Condor Pool can run multiple jobs at once
Frieda tries out parallel jobs…

- MPI Universe
  - MPI (Message Passing Interface): de facto standard for communication among nodes running a parallel program on a distributed memory system.
  - as Fortran, C, C++ libraries

- PVM Universe
  - Parallel Virtual Machine (PVM) is designed to allow a network of heterogeneous machines to be used as a single distributed parallel processor

Schedule and start an MPICH job on dedicated resources

Executable = my-mpi-job
Universe = MPI
Machine_count = 8
queue

(Boss Fat Cat)
The Boss says Frieda can add her co-workers’ desktop machines into her Condor pool as well... but only if they can also submit jobs.
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
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Queue 600

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
  – Remote I/O Socket
  – Standard Universe can use Remote System Calls

Condor File Transfer

• Set ShouldTransferFiles
  – YES: Always transfer files to execution site
  – NO: Rely on a shared filesystem
  – IF_NEEDED: will automatically transfer the files if the submit and execute machine are not in the same FileSystemDomain

Universe = vanilla
Executable = my_job
Arguments = -arg1 -arg2
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 600

Specify Rank!

• All matches which meet the requirements can be sorted by preference with a Rank expression.
• The higher the Rank, the better the match

Universe = vanilla
Executable = my_job
Arguments = -arg1 -arg2
InitialDir = run_$(Process)
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 600
Outcomes of the Study

1. Stork interacted easily and successfully with different underlying systems: SRB, UniTree, GridFTP and Diskrouter.

2. We had the chance to compare different pipeline topologies and configurations:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>End-to-end rate (MB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.0</td>
</tr>
<tr>
<td>2</td>
<td>3.2</td>
</tr>
<tr>
<td>3</td>
<td>5.95</td>
</tr>
</tbody>
</table>

3. Almost all possible network, server, and software failures were recovered automatically.

Policy Configuration

I am adding nodes to the Cluster... but the Engineering Department has priority on these nodes.

(Boss Fat Cat)

The Cluster is fine. But not the desktop machines. Condor can only use the desktops when they would otherwise be idle.

General User Commands

- condor_status View Pool Status
- condor_q View Job Queue
- condor_submit Submit new Jobs
- condor_rm Remove Jobs
- condor_prio Intra-User Prios
- condor_history Completed Job Info
- condor_submit_dag Specify Dependencies
- condor_checkpoint Force a checkpoint
- condor_compile Link Condor library

Administrator Commands

- condor_vacate Leave a machine now
- condor_on Start Condor
- condor_off Stop Condor
- condor_reconfig Reconfig on-the-fly
- condor_config_val View/set config
- condor_userprio User Priorities
- condor_stats View detailed usage accounting stats
## Condor Job Universes

- **Serial Jobs**
  - Vanilla Universe
  - Standard Universe
- **Scheduler Universe**
- **Parallel Jobs**
  - MPI Universe
  - PVM Universe
- **Java Universe**

## Java Universe Job

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

## 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.

## Job Policy Expressions

- User can supply job policy expressions in the submit file.
- Can be used to describe a successful run.
  ```
  on_exit_remove = <expression>
  on_exit_hold = <expression>
  periodic_remove = <expression>
  periodic_hold = <expression>
  ```

## Job Policy Examples

- Do not remove if exits with a signal:
  ```
  on_exit_remove = ExitBySignal == False
  ```
- Place on hold if exits with nonzero status or ran for less than an hour:
  ```
  on_exit_hold = ((ExitBySignal==False) && (ExitSignal != 0)) || ((ServerStartTime – JobStartDate) < 3600)
  ```
- Place on hold if job has spent more than 50% of its time suspended:
  ```
  periodic_hold = CumulativeSuspensionTime > (RemoteWallClockTime / 2.0)
  ```
How Flocking Works

- Add a line to your condor_config:
  
  FLOCK_HOSTS = Pool-Foo, Pool-Bar

---

Condor Flocking

- Remote pools are contacted in the order specified until jobs are satisfied
- The list of remote pools is a property of the Schedd, not the Central Manager
  - So different users can Flock to different pools
  - And remote pools can allow specific users
- User-priority system is “flocking-aware”
  - A pool’s local users can have priority over remote users “flocking” in.

---

Condor Flocking, cont.

- Flocking is “Condor” specific technology...
- Frieda also has access to Globus resources she wants to use
  - She has certificates and access to Globus gatekeepers at remote institutions
- But Frieda wants Condor’s queue management features for her Globus jobs!
- She installs Condor-G so she can submit “Globus Universe” jobs to Condor

---

Condor-G: Globus + Condor

- Condor
  - job scheduling across multiple resources
  - strong fault tolerance with checkpointing and migration
  - layered over Globus as “personal batch system” for the Grid

- Globus
  - middleware deployed across entire Grid
  - remote access to computational resources
  - dependable, robust data transfer