Campus Condor Pools

Mats Rynge – rynge@renci.org

Renaissance Computing Institute

University of North Carolina, Chapel Hill

Slide content from *Using Condor: an Introduction* - Condor Week

Outline

- What is a campus Condor pool?
- Example deployments
- Tarheel Grid
- Introduction to Condor

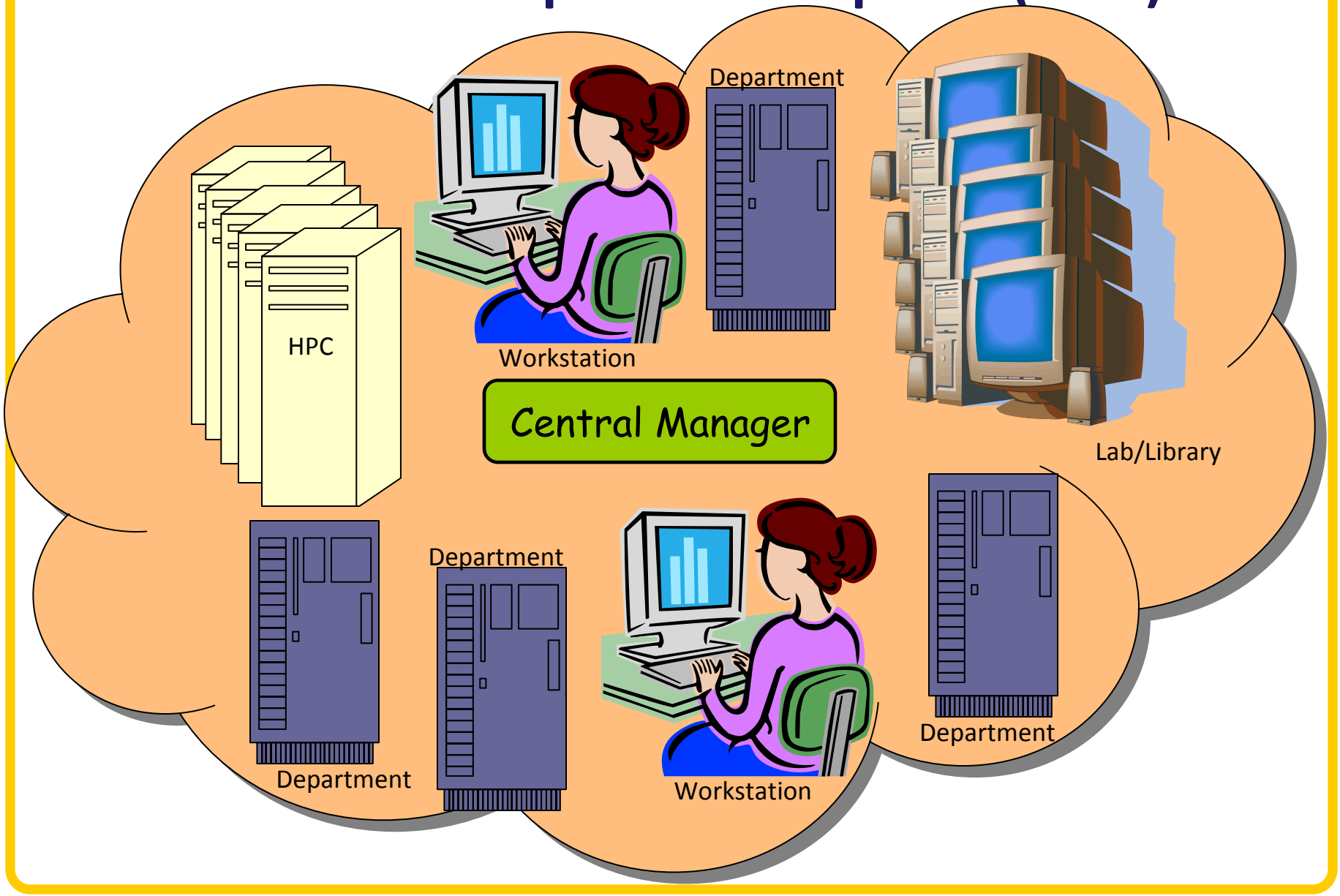
Campus Compute Resources

- There is no place like home!
- HPC (High Performance Computing)
  - Topsail/Emerald (UNC), Sam/HenryN/POWER5 (NCSU), Duke Shared Cluster Resource (Duke)
- HTC (High Throughput Computing)
  - Tarheel Grid, NCSU Condor pool, Duke departmental pools

What is a campus Condor pool?

- Enable departments and researchers on campus to share their *existing* computers
- The same departments and researchers will have access to a larger pool of resources when they need to do large computations
- Community driven project
  - Enabled by central/department IT
- Mix of resource types, operating systems, configurations such as UID and file system setups

What is a campus Condor pool? (cont)



Clemson Campus Condor Pool

Machines in 50 different locations on Campus

~1,700 job slots

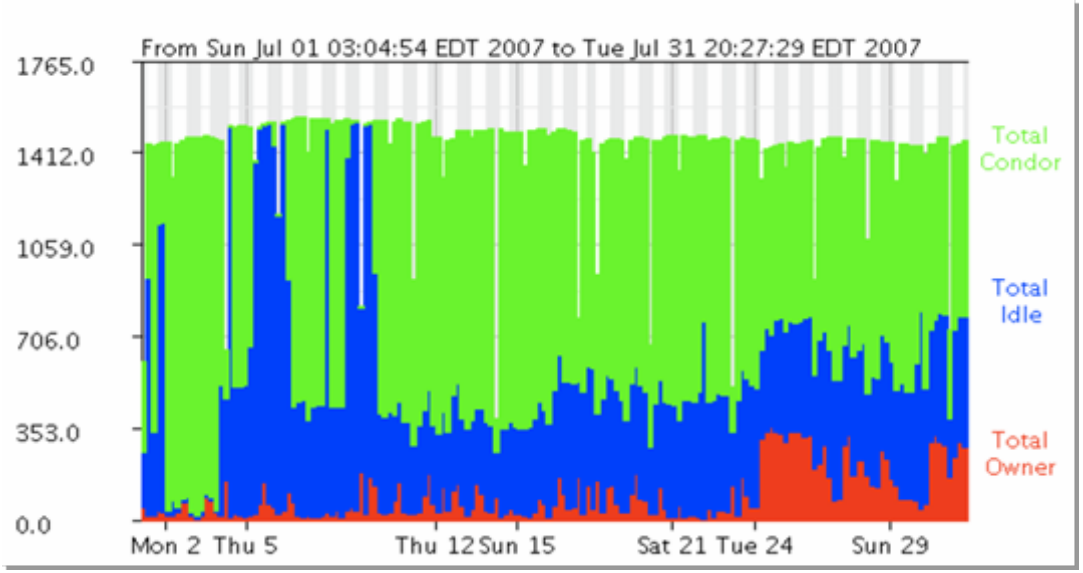
>1.8M hours served in 6 months

	cpu Hours	Users	Jobs Running Avg. #
January	1918	developer	8
February	1332	developer	6
March	217,961	2	334
April	108,119	2	157
May	128,101	3	172
June	174,240	6	248
July	627,102	9	855
August (thru 8/24)	418,238	9	812

Users from Industrial and Chemical engineering, and Economics

Fast ramp up of usage

Accessible to the OSG through a gateway



Clemson (cont.)

- 1085 windows machines, 2 linux machines (central and a OSG gatekeeper), condor reporting 1563 slots
- 845 maintained by CCIT
- 241 from other campus depts
- >50 locations
- From 1 to 112 machines in one location
- Student housing, labs, library, coffee shop

- Mary Beth Kurz, first condor user at Clemson:
  - March 215,000 hours, ~110,000 jobs
  - April 110,000 hours, ~44,000 jobs

NWICG
northwest indiana computational grid

14,000 CPUs available

Campus Condor pool backfills idle nodes in PBS clusters - provided 5.5 million CPU-hours in 2006, all from idle nodes in clusters

Use on TeraGrid: 2.4 million hours in 2006 spent Building a database of hypothetical zeolite structures; 2007: 5.5 million hours allocated to TG

PURDUE UNIVERSITY News

OTHER LINKS

News Archive (searchable) | Recent releases | News Service home page

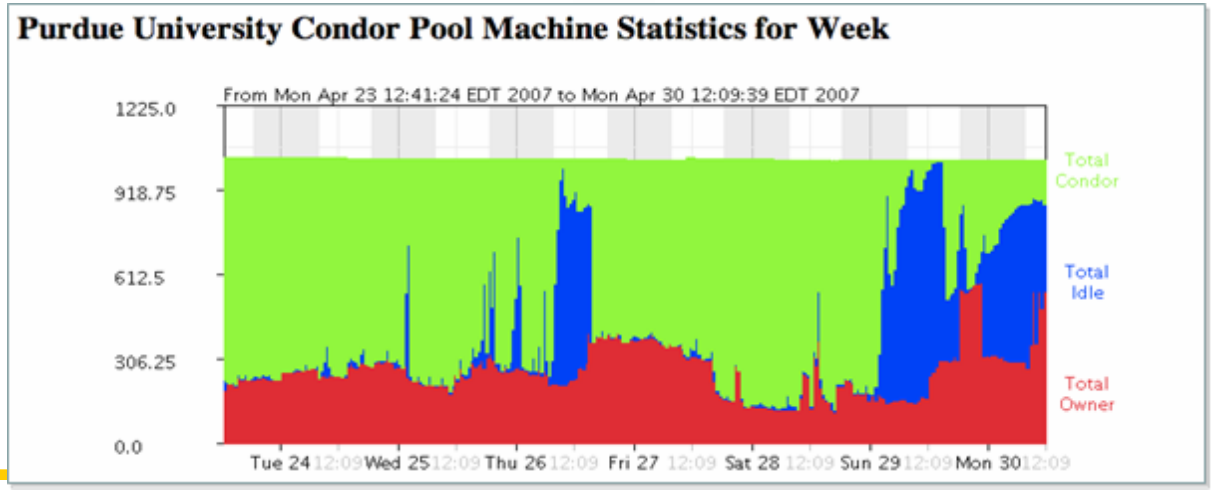
November 1, 2006

Indiana receives funding for regional science computer grid

WEST LAFAYETTE, Ind. — The economy of northwest Indiana will be energized thanks to congressional authorization of $5 million to support a computer grid for the area, officials at Purdue and Notre Dame universities announced Wednesday (Nov. 1).

The U.S. Department of Energy's National Nuclear Security Administration will award the funding for the Northwest Indiana Computational Grid, bringing federal investment in this project to $6.5 million.

Gerry McCartney



http://www.cs.wisc.edu/condor/PCW2007/presentations/cheeseman_Purdue_Condor_Week_2007.ppt

Condor

- Distributed workload manager
  - Central broker to handle distributed submit nodes, and distributed execution nodes
- High Throughput Computing (HTC)
  - Perfect for:
    - Parameter sweeps
    - Monte Carlo simulation
    - Workflows
    - Other serial applications
- Well proven technology

Condor Features

- Flexible scheduling policy engine via ClassAds
  - Preemption, suspension, requirements, preferences, groups, quotas, settable fair-share, system hold…
- Facilities to manage BOTH dedicated CPUs (clusters) and non-dedicated resources (desktops)
- Transparent Checkpoint/Migration for many types of serial jobs
- No shared file-system required
- Workflow management (inter-dependencies)
- Fault-tolerant: can survive crashes, network outages
- Platforms: Linux i386/IA64, Windows 2k/XP, MacOS, Solaris, IRIX, HP-UX, Compaq Tru64, … lots.

Tarheel Grid

- ITS provides central services (collector, negotiator, one submit node, …)
- Campus departments / researches can join their existing compute resources to the pool, and share with the community
- Resource owners have full control over the policy for their resources

Machines currently in the Tarheel Grid pool

- Old Baobab nodes
- A set of ISIS machines
- Odum Institute lab machines
- Open Science Grid cluster at RENCI
- Workstations / dedicated processing machines

- 158 slots Linux X86
- 41 slots Linux X86_64
- 2 slots Linux PPC64
- 54 slots Windows

Using Condor: an Introduction

Definitions

- Job
  - The Condor representation of your work
- Machine
  - The Condor representation of computers and that can perform the work
- Match Making
  - Matching a job with a machine “Resource”
- Central Manager
  - Central repository for the whole pool
  - Performs job / machine matching, etc.

Job

Jobs state their requirements and preferences:

- I **require** a Linux/x86 platform
- I **prefer** the machine with the most memory
- I **prefer** a machine in the chemistry department

Machine

Machines state their requirements and preferences:

- **Require** that jobs run only when there is no keyboard activity
- I **prefer** to run Albert's jobs
- I am a machine in the physics department
- **Never** run jobs belonging to Dr. Heisenberg

The Magic of Matchmaking

- Jobs and machines state their requirements and preferences
- Condor matches *jobs* with *machines*
  - Based on **requirements**
  - *And* preferences "**Rank**"

Getting Started: Submitting Jobs to Condor

- Overview:
  - Choose a "Universe" for your job
  - Make your job "batch-ready"
  - Create a *submit description* file
  - Run *condor_submit* to put your job in the queue

1. Choose the "Universe"

- Controls how Condor handles jobs
- Choices include:
  - Vanilla
  - Standard
  - Grid
  - Java
  - Parallel
  - VM

2. Make your job batch-ready

Must be able to run in the background

- No interactive input
- No GUI/window clicks
  - Who needs ‘em anyway

Make your job batch-ready (details)…

– Job can still use **STDIN**, **STDOUT**, and **STDERR** (the keyboard and the screen), but files are used for these instead of the actual devices

– Similar to UNIX shell:

- **`$ ./myprogram <input.txt >output.txt`**

3. Create a Submit Description File

- A plain ASCII text file
- 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 (more on this later)
- Can describe many jobs at once (a “cluster”), each with different input, arguments, output, etc.

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 = cosmos
Output     = cosmos.out
Queue
```

4. Run condor_submit

- You give *condor_submit* the name of the submit file you have created:
  - **`condor_submit cosmos.submit`**
- *condor_submit*:
  - Parses the submit file, checks for errors
  - Creates a "ClassAd" that describes your job(s)
  - Puts job(s) in the Job Queue

ClassAd ?

- Condor's internal data representation
  - Similar to classified ads (as the name implies)
  - Represent an object & its attributes
    - Usually many attributes
  - Can also describe what an object matches with

ClassAd Details

- ClassAds can contain a lot of details
  - The job's executable is "**`cosmos`**"
  - The machine's load average is 5.6
- ClassAds can specify requirements
  - I require a machine with Linux
- ClassAds can specify preferences
  - This machine prefers to run jobs from the physics group

ClassAd Details (continued)

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

ClassAd Example

Example:

```
MyType        = "Job"
TargetType    = "Machine"
ClusterId     = 1377
Owner         = “einstein"
Cmd           = “cosmos"
Requirements =
   (
     (Arch == "INTEL")   &&
     (OpSys == "LINUX")  &&
     (Disk >= DiskUsage) &&
     ( (Memory * 1024)>=ImageSize )
   )
```

String

Number

Boolean

…

The Dog

ClassAd

```
Type  = “Dog”
Color = “Brown”
Price = 12
```

ClassAd for the “Job”

```
. . .
Requirements =
 (type == “Dog”)    &&
 (color == “Brown”) &&
 (price <= 15)
. . .
```

Example: condor_submit and condor_q

```
% condor_submit cosmos.submit
Submitting job(s).
1 job(s) submitted to cluster 1.

% condor_q

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED   RUN_TIME  ST PRI SIZE CMD
  1.0    einstein         4/15 06:52  0+00:00:00 I 0   0.0  cosmos

1 jobs; 1 idle, 0 running, 0 held

%
```

Email about your job

- Condor sends email about job events to the submitting user

- Specify “notification” in your submit file to control which events:

```
Notification = complete
Notification = never
Notification = error
Notification = always
```

"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 (i.e. 2.1)
  - A cluster can have a single process
    - Job ID = 20.0 ·**Cluster 20, process 0**
  - Or, a cluster can have more than one process
    - Job ID: 21.0, 21.1, 21.2 ·**Cluster 21, process 0, 1, 2**

Submit File for a Cluster

```
# Example submit file for a cluster of 2 jobs
# with separate input, output, error and log files
Universe   = vanilla
Executable = cosmos
Arguments  = -f cosmos_0.dat
log        = cosmos_0.log
Input      = cosmos_0.in
Output     = cosmos_0.out
Error      = cosmos_0.err
Queue                    ·Job 2.0 (cluster 2, process 0)
Arguments = -f cosmos_1.dat
log        = cosmos_1.log
Input      = cosmos_1.in
Output     = cosmos_1.out
Error      = cosmos_1.err
Queue                    ·Job 2.1 (cluster 2, process 1)
```

Submit Description File for 1,000,000 Jobs

```
# Cluster of 1,000,000 jobs with different directories
Universe   = vanilla
Executable = cosmos
Log        = cosmos.log
Output     = cosmos.out
Input      = cosmos.in
Arguments  = –f cosmos.dat
...
InitialDir = run_0
Queue
InitialDir = run_1
Queue
```

- Log, input, output & error files -> run_0
- Job 3.0 (Cluster 3, Process 0)
- Log, input, output & error files -> run_1
- Job 3.1 (Cluster 3, Process 1)

- Do this 999,998 more times……

Submit File for a Big Cluster of Jobs

- We just submitted 1 cluster with 1,000,000 processes
- All the input/output files will be in different directories
- The submit file is pretty unwieldy (over 5,000,000 lines!)
- Isn't there a better way?

Submit File for a Big Cluster of Jobs (the better way)

- We can queue all 1,000,000 in 1 "Queue" command
  - **`Queue 1000000`**
- Condor provides **`$(Process)`** and **`$(Cluster)`**
  - **`$(Process)`** will be expanded to the process number for each job in the cluster
    - 0, 1, … 999999
  - **`$(Cluster)`** will be expanded to the cluster number
    - Will be 4 for all jobs in this cluster

$(Process)

- The initial directory for each job can be specified using **`$(Process)`**
  - **`InitialDir = run_$(Process)`**
  - Condor will expand these to:
    - "**`run_0`**", "**`run_1`**", … "**`run_999999`**" directories
- Similarly, arguments can be variable
  - **`Arguments = -n $(Process)`**
  - Condor will expand these to:
    - **"`-n 0`", "`-n 1`", … "`-n 999999`"**

Better Submit File for 1,000,000 Jobs

```
# Example condor_submit input file that defines
# a cluster of 100000 jobs with different directories
Universe   = vanilla
Executable = cosmos
Log        = cosmos.log
Input      = cosmos.in
Output     = cosmos.out
Error      = cosmos.err
Arguments  = –f cosmos.dat
InitialDir = run_$(Process)
Queue 1000000
```

- All share arguments
- run_0 … run_999999
- Jobs 0 … 999999

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 (parrot)

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

```
# Example submit file using file transfer
Universe   = vanilla
Executable = cosmos
Log        = cosmos.log
ShouldTransferFiles   = IF_NEEDED
Transfer_input_files  = dataset.$(Process), common.data
Transfer_output_files = TheAnswer.dat
Queue 600
```

Now what?

- Some of the machines in the pool can't run my jobs
  - Not the right operating system for my executable
  - Not enough RAM
  - Not enough scratch disk space
  - Required software not installed
  - Etc.

Specify Requirements

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

```
Universe   = vanilla
Executable = cosmos
Log        = cosmos.log
InitialDir = run_$(Process)
Requirements = Memory >= 4096 && Disk > 10000
Queue 1000000
```

Advanced Requirements

- Requirements can match custom attributes in your Machine Ad
  - Can be added by hand to each machine
  - Or, automatically using the scripts

```
Universe   = vanilla
Executable = cosmos
Log        = cosmos.log
InitialDir = run_$(Process)
Requirements = ( (HasCosmosData =!= UNDEFINED) && \
                 (HasCosmosData == TRUE)  )
Queue 1000000
```

And, 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 = cosmos
Log        = cosmos.log
Arguments  = -arg1 –arg2
InitialDir = run_$(Process)
Requirements = Memory >= 4096 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 1000000
```

My jobs aren't running!!

Check the queue

- Check the queue with condor_q:

```
bash-2.05a$ condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510> :x.cs.wisc.edu
ID  OWNER   SUBMITTED    RUN_TIME ST PRI SIZE CMD
5.0 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 0
5.1 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 1
5.2 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 2
5.3 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 3
5.4 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 4
5.5 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 5
5.6 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 6
5.7 einstein  4/20 12:23  0+00:00:00 I 0   9.8  cosmos -arg1 –n 7
6.0 einstein  4/20 13:22  0+00:00:00 H 0   9.8  cosmos -arg1 –arg2
8 jobs; 8 idle, 0 running, 1 held
```

Look at jobs on hold

```
% condor_q –hold
-- Submiter: x.cs.wisc.edu : <128.105.121.53:510>
   :x.cs.wisc.edu
 ID      OWNER            HELD_SINCE HOLD_REASON
 6.0   einstein             4/20 13:23 Error from starter on
   skywalker.cs.wisc.edu

9 jobs; 8 idle, 0 running, 1 held
```

Or, See full details for a job

```
% condor_q –l 6.0
```

Check machine status

- Verify that there are idle machines with condor_status:

```
bash-2.05a$ condor_status
Name          OpSys   Arch   State     Activity LoadAv Mem   ActvtyTime
vm1@tonic.c LINUX   INTEL Claimed Busy      0.000   501  0+00:00:20
vm2@tonic.c LINUX   INTEL Claimed Busy      0.000   501  0+00:00:19
vm3@tonic.c LINUX   INTEL Claimed Busy      0.040   501  0+00:00:17
vm4@tonic.c LINUX   INTEL Claimed Busy      0.000   501  0+00:00:05

               Total Owner Claimed Unclaimed Matched Preempting
INTEL/LINUX        4     0       4         0       0          0
      Total        4     0       4         0       0          0
```

Look in Job Log

- Look in your job log for clues:

```
bash-2.05a$ cat cosmos.log
000 (031.000.000) 04/20 14:47:31 Job submitted from host:
   <128.105.121.53:48740>
...
007 (031.000.000) 04/20 15:02:00 Shadow exception!
        Error from starter on gig06.stat.wisc.edu: Failed
   to open '/scratch.1/einstein/workspace/v67/condor-
   test/test3/run_0/cosmos.in' as standard input: No such
   file or directory (errno 2)
        0   -   Run Bytes Sent By Job
        0   -   Run Bytes Received By Job
...
```

Better analysis:
condor_q –better-analyze

```
bash-2.05a$ condor_q -better-ana 29
The Requirements expression for your job is:

( ( target.Memory > 8192 ) ) && ( target.Arch == "INTEL" ) &&
( target.OpSys == "LINUX" ) && ( target.Disk >= DiskUsage ) &&
( TARGET.FileSystemDomain == MY.FileSystemDomain )
Condition                          Machines Matched  Suggestion
---------                          ----------------  ----------
1 ( ( target.Memory > 8192 ) )     0                 MODIFY TO 4000
2 ( TARGET.FileSystemDomain == "cs.wisc.edu" )584
3   ( target.Arch == "INTEL" )     1078
4   ( target.OpSys == "LINUX" )    1100
5   ( target.Disk >= 13 )          1243
```

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

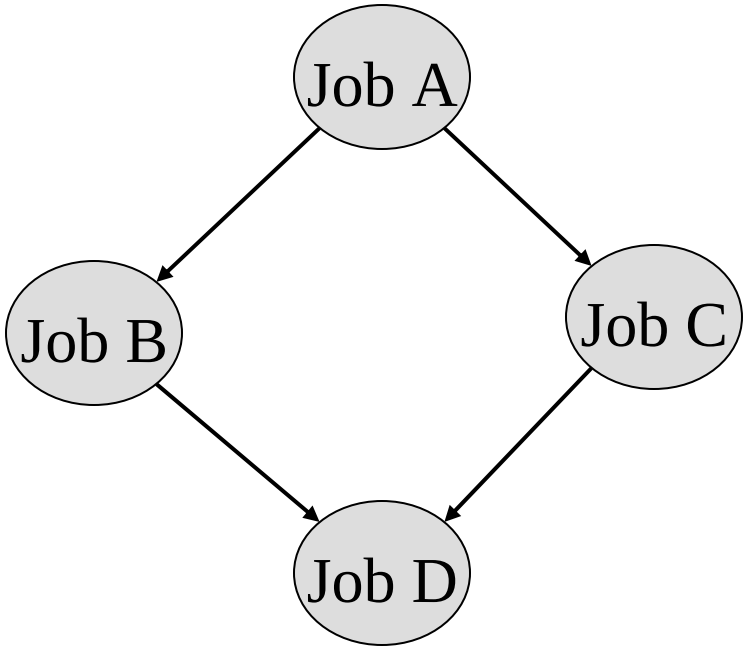
- Do not remove if exits with a signal:
  - `on_exit_remove = ExitBySignal == False`
- Place on hold if exits with nonzero status or ran for less than an hour:
  - `on_exit_hold = ( (ExitBySignal==False) && (ExitSignal != 0) ) || ( (ServerStartTime - JobStartDate) < 3600)`
- Place on hold if job has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

DAGMan

- **Directed Acyclic Graph Manager**
- DAGMan allows you to specify the *dependencies* between your Condor jobs, so it can *manage* them automatically for you.
  - E.G.: “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!



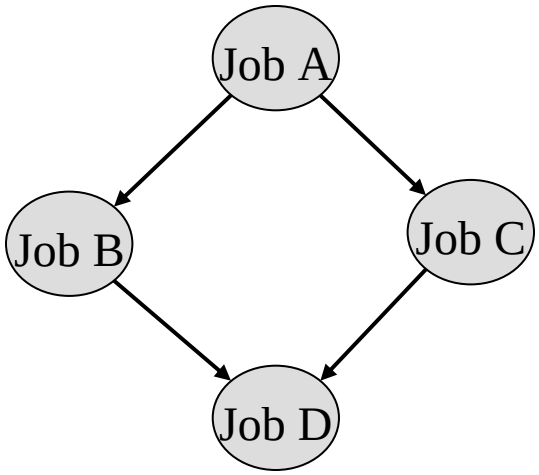
DAGs large and small

- DAGMan can scale up to huge DAGs
  - LIGO runs 500,000 node DAGs
- DAGMan is useful even in the small cases!
  - What about a simple 2 job DAG?
    - Run database query
    - Process output
    - Can all be run on one machine--no Condor pool necessary
- DAGMan can be used as a Condor job throttle
  - Lots of nodes with no dependencies

Defining a DAG

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

```
# diamond.dag
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
```



- each node will run the Condor job specified by its accompanying Condor submit file

Condor Commands Cheat Sheet

- condor_status
- condor_submit
- condor_q
- condor_rm
- condor_submit_dag

E N D