Using Condor

Seminar and Workshop on Grid Computing at UT

UT Knoxville Campus

January 2002

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

Tutorial Outline

- Overview
- The Story of Frieda, the Scientist
  - Using Condor to manage jobs
  - Using Condor to manage resources
  - Condor Architecture and Mechanisms
  - Condor on the Grid
    - Flocking
    - Condor-G

Meet Frieda.

She is a scientist. But she has a big problem.

Frieda's Application ...

Simulate the behavior of **F**(*x,y,z*) for 20 values of *x*, 10 values of *y* and 3 values of *z* (20*10*3 = 600 combinations)

- **F** takes on the average 6 hours to compute on a "typical" workstation (total = 1800 hours)
- **F** requires a "moderate" (128MB) amount of memory
- **F** performs "moderate" I/O - (*x,y,z*) is 5 MB and **F**(*x,y,z*) is 50 MB

I have 600 simulations to run.

Where can I get help?

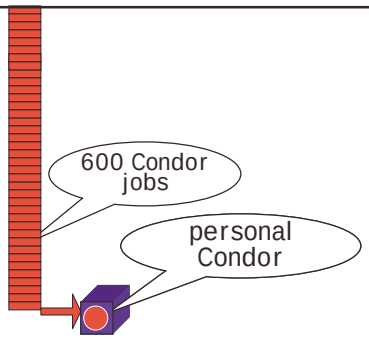
Install a Personal Condor!

Installing Condor

- Download Condor for your operating system
- Available as a free download from
  http://www.cs.wisc.edu/condor
- Stable –vs- Developer Releases
  - Naming scheme similar to the Linux Kernel…
- Available for most Unix platforms and Windows NT

So Frieda Installs Personal Condor on her machine…

- What do we mean by a "Personal" Condor?
  - Condor on your own workstation, no root access required, no system administrator intervention needed
- So after installation, Frieda submits her jobs to her Personal 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

Getting Started: Submitting Jobs to Condor

- Choosing a "Universe" for your job
  - Just use VANILLA for now
- Make your job "batch-ready"
- Creating a *submit description* file
- Run *condor_submit* on your submit description file

Making 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

Creating a Submit Description File

- A plain ASCII text file
- 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 = my_job
Queue
```

Running condor_submit

- You give *condor_submit* the name of the submit file you have created
- *condor_submit* parses the file, checks for errors, and creates a "ClassAd" that describes your job(s)
- Sends your job's ClassAd(s) and executable to the condor_schedd, which stores the job in its queue
  - Atomic operation, two-phase commit
- View the queue with *condor_q*

Running condor_submit

```
% condor_submit my_job.submit-file
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    frieda           6/16 06:52   0+00:00:00 I  0   0.0  my_job

1 jobs; 1 idle, 0 running, 0 held

%
```

Another Submit Description File

```
# Example 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 = /home/wright/condor/my_job.condor
Input      = my_job.stdin
Output     = my_job.stdout
Error      = my_job.stderr
Arguments  = -arg1 -arg2
InitialDir = /home/wright/condor/run_1
Queue
```

"Clusters" and "Processes"

- If your submit file describes multiple jobs, we call this a "cluster"
- Each job within a cluster is called a "process" or "proc"
- If you only specify one job, you still get a cluster, but it has only one process
- A Condor "Job ID" is the cluster number, a period, and the process number ("23.5")
- Process numbers always start at 0

Example Submit Description File for a Cluster

```
# Example condor_submit input file that defines
# a cluster of two jobs with different iwd
Universe   = vanilla
Executable = my_job
Arguments  = -arg1 -arg2

InitialDir = run_0
Queue      ← Becomes job 2.0

InitialDir = run_1
Queue      ← Becomes job 2.1
```

```
% 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> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda           6/16 06:52  0+00:02:11 R  0   0.0  my_job
  2.0    frieda           6/16 06:56  0+00:00:00 I  0   0.0  my_job
  2.1    frieda           6/16 06:56  0+00:00:00 I  0   0.0  my_job

3 jobs; 2 idle, 1 running, 0 held

%
```

Submit Description File for a *BIG* Cluster of Jobs

- The initial directory for each job is specified with the $(Process) macro, and instead of submitting a single job, we use "Queue 600" to submit 600 jobs at once
- $(Process) will be expanded to the process number for each job in the cluster (from 0 up to 599 in this case), so we'll have "run_0", "run_1", … "run_599" directories
- All the input/output files will be in different directories!

Submit Description File for a *BIG* Cluster of Jobs

```
# Example condor_submit input file that defines
# a cluster of 600 jobs with different iwd
Universe   = vanilla
Executable = my_job
Arguments  = -arg1 –arg2
InitialDir = run_$(Process)
Queue 600
```

Using condor_rm

- If you want to remove a job from the Condor queue, you use *condor_rm*
- You can only remove jobs that you own (you can't run *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 "*-a*" option.

Temporarily halt a Job

- Use *condor_hold* to place a job on hold
  - Kills job if currently running
  - Will not attempt to restart job until released
- Use *condor_release* to remove a hold and permit job to be scheduled again

Using condor_history

- Once your job completes, it will no longer show up in *condor_q*
- You can use *condor_history* to view information about a completed job
- The status field ("ST") will have either a "C" for "completed", or an "X" if the job was removed with *condor_rm*

Getting Email from Condor

- By default, Condor will send you email when your jobs completes
  - With lots of information about the run
- If you don't want this email, put this in your submit file:

```
notification = never
```

- If you want email every time something happens to your job (preempt, exit, etc), use this:

```
notification = always
```

Getting Email from Condor (cont'd)

- If you only want email in case of errors, use this:

```
notification = error
```

- By default, the email is sent to your account on the host you submitted from. If you want the email to go to a different address, use this:

```
notify_user = email@address.here
```

A Job's life story: The "User Log" file

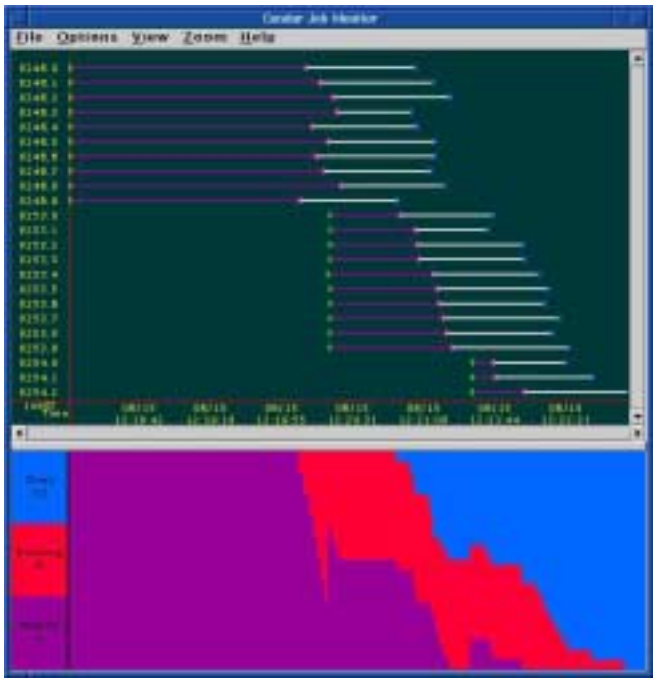
- A UserLog must be specified in your submit file:
  - Log = filename
- You get a log entry for everything that happens to your job:
  - When it was submitted, when it starts executing, preempted, restarted, completes, if there are any problems, etc.
- Very useful! Highly recommended!

Sample Condor User Log

```
000 (8135.000.000) 05/25 19:10:03 Job submitted from host: <128.105.146.14:1816>
...
001 (8135.000.000) 05/25 19:12:17 Job executing on host: <128.105.165.131:1026>
...
005 (8135.000.000) 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
...
```

Uses for the User Log

- Easily read by human or machine
  - C++ library and Perl Module for parsing UserLogs is available
- Event triggers for meta-schedulers
  - Like DagMan…
- Visualizations of job progress
  - Condor JobMonitor Viewer

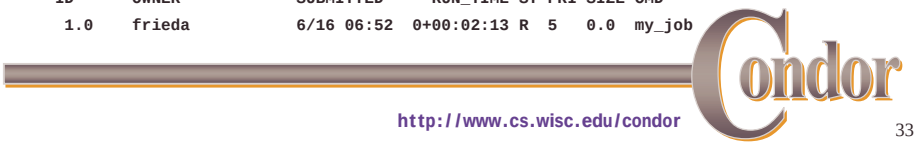


Condor JobMonitor Screenshot

Job Priorities w/ condor_prio

- *condor_prio* allows you to specify the order in which your jobs are started
- Higher the prio #, the earlier the job will start

```
% condor_q
-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda           6/16 06:52  0+00:02:11 R  0   0.0  my_job
% condor_prio +5 1.0
% condor_q
-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda           6/16 06:52  0+00:02:13 R  5   0.0  my_job
```



Want other Scheduling possibilities? Use the Scheduler Universe

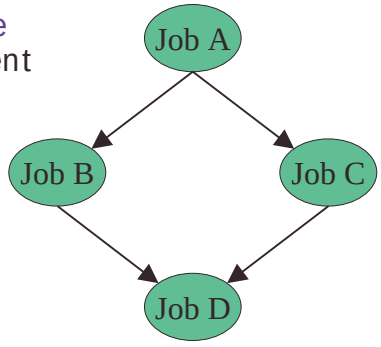
- In addition to VANILLA, another job universe is the *Scheduler Universe*.
- Scheduler Universe jobs run on the submitting machine and serve as a meta-scheduler.
- DAGMan meta-scheduler included

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?

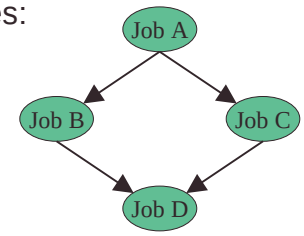
- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a "node" in the DAG.
- Each node can have any number of "parent" or "children" nodes – as long as there are no loops!



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

Submitting a DAG

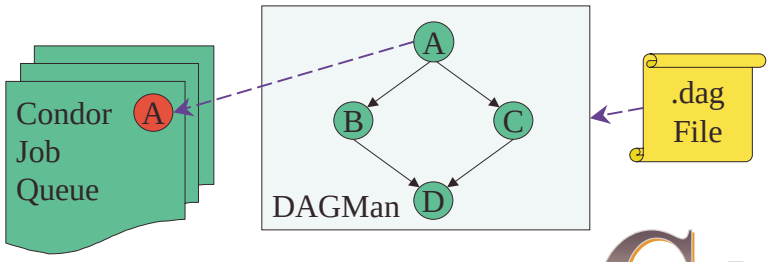
- To start your DAG, just run ***condor_submit_dag*** with your .dag file, and Condor will start a personal DAGMan daemon which to begin running your jobs:

```
% 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.

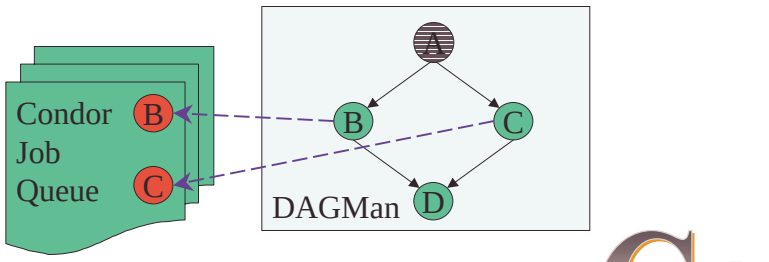
Running a DAG

- DAGMan acts as a “meta-scheduler”, managing the submission of your jobs to Condor based on the DAG dependencies.



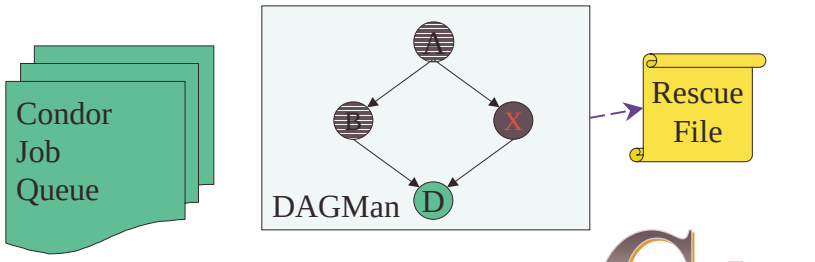
Running a DAG (cont’d)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.



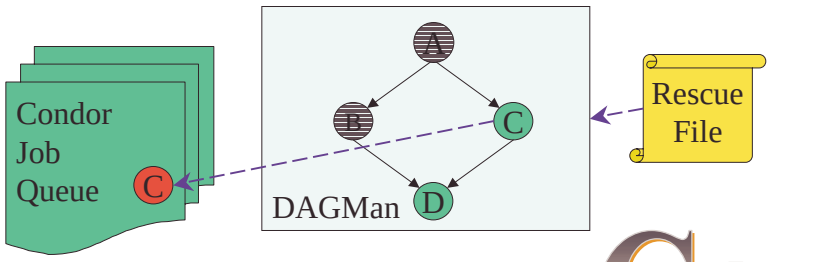
Running a DAG (cont'd)

- 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.



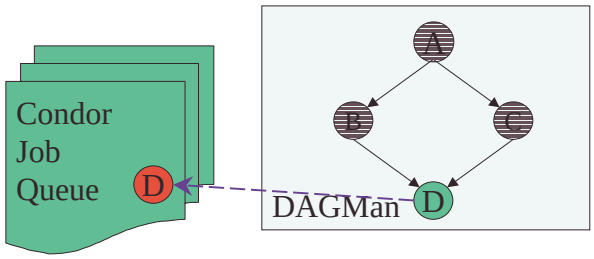
Recovering a DAG

- Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.



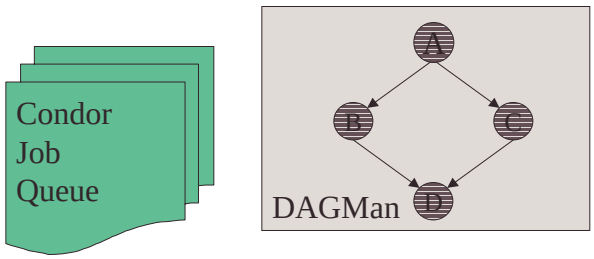
Recovering a DAG (cont'd)

- 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.



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

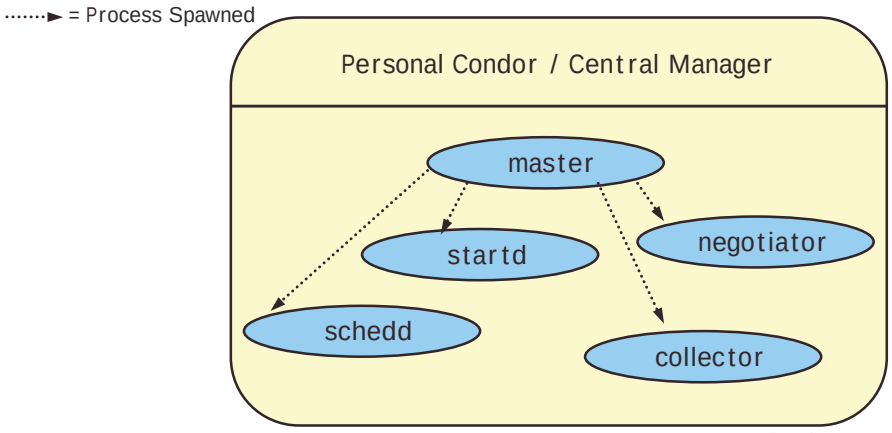
- Condor's checkpointing is not at the kernel level. Thus in the Standard Universe the job may not
  - Fork()
  - Use kernel threads
  - Use some forms of IPC, such as pipes and shared memory
- Many typical scientific jobs are OK

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 Daemon Layout



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

condor_master (cont'd)

- 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

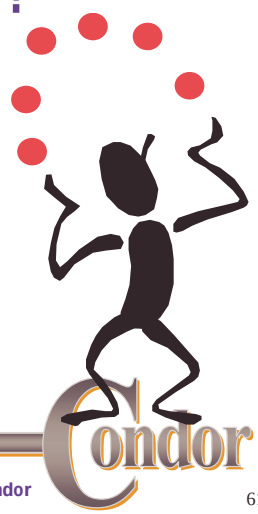
- Collects information from all other Condor daemons in the pool
  - "Directory Service" / Database for a Condor pool
- Each daemon sends a periodic update called a "ClassAd" to the collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (*condor_status*)

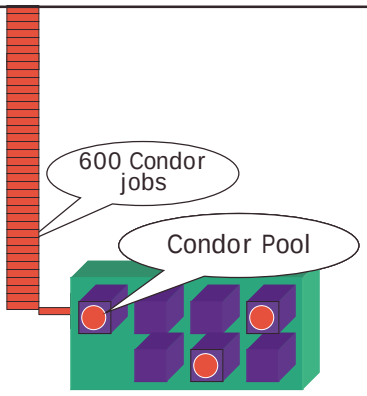
condor_negotiator

- 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

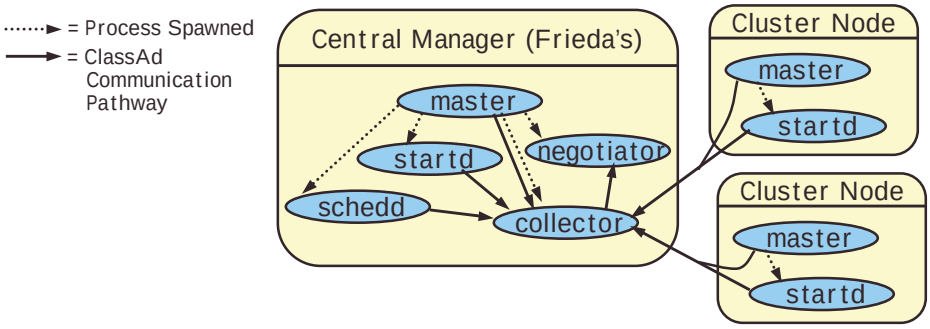
Happy Day! Frieda's organization purchased a Beowulf Cluster!

- 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





Layout of the Condor Pool



condor_status

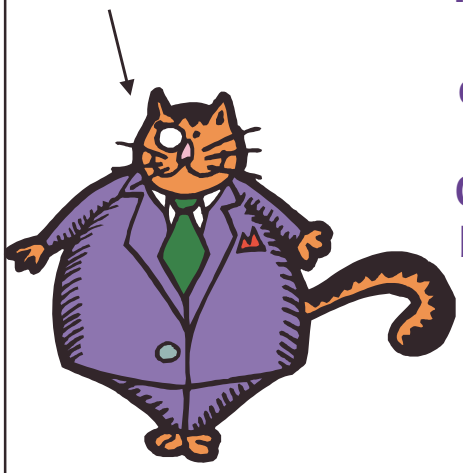
```
% condor_status

Name          OpSys       Arch   State      Activity   LoadAv Mem   ActvtyTime

haha.cs.wisc. IRIX65      SGI    Unclaimed  Idle       0.198   192  0+00:00:04
antipholus.cs LINUX       INTEL  Unclaimed  Idle       0.020   511  0+02:28:42
coral.cs.wisc LINUX       INTEL  Claimed    Busy       0.990   511  0+01:27:21
doc.cs.wisc.e LINUX       INTEL  Unclaimed  Idle       0.260   511  0+00:20:04
dsonokwa.cs.w LINUX       INTEL  Claimed    Busy       0.810   511  0+00:01:45
ferdinand.cs. LINUX       INTEL  Claimed    Suspended  1.130   511  0+00:00:55
vm1@pinguino. LINUX       INTEL  Unclaimed  Idle       0.000   255  0+01:03:28
vm2@pinguino. LINUX       INTEL  Unclaimed  Idle       0.190   255  0+01:03:29
```

Frieda tries out parallel jobs…

- MPI Universe & PVM Universe
- Schedule and start an MPICH job on dedicated resources

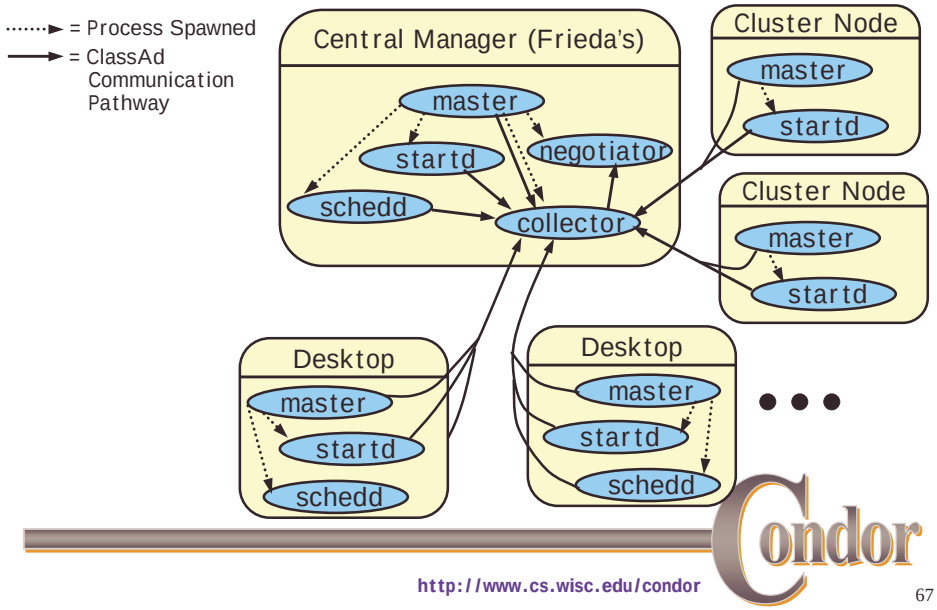
```
Executable = my-mpi-job
Universe = MPI
Machine_count = 8
queue
```

(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.

Layout of the Condor Pool



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
```

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

How can my jobs access their data files?

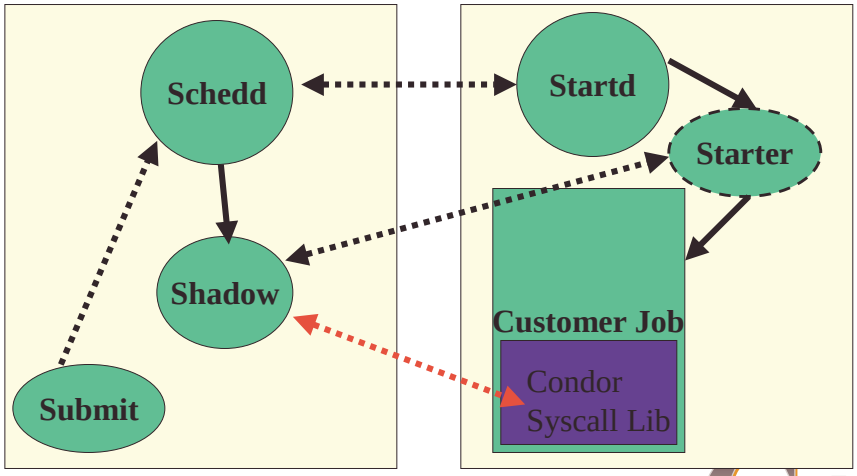
Access to Data in Condor

- Use Shared Filesystem if available
- No shared filesystem?
  - Condor can transfer files
    - Automatically send back changed files
    - Atomic transfer of multiple files
  - Standard Universe can use *Remote System Calls*

Remote System Calls

- I/O System calls 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

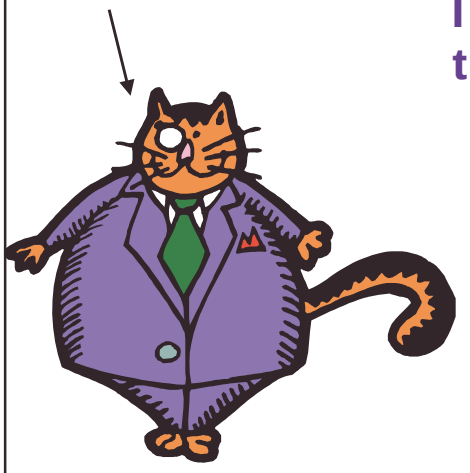


condor_q -io

```
c01(69)% condor_q -io


-- Submitter: c01.cs.wisc.edu : <128.105.146.101:2996> : c01.cs.wisc.edu
ID       OWNER        READ    WRITE     SEEK       XPUT  BUFSIZE  BLKSIZE
  72.3   edayton           [ no i/o data collected yet ]
  72.5   edayton    6.8 MB   0.0 B         0 104.0 KB/s 512.0 KB  32.0 KB
  73.0   edayton    6.4 MB   0.0 B         0 140.3 KB/s 512.0 KB  32.0 KB
  73.2   edayton    6.8 MB   0.0 B         0 112.4 KB/s 512.0 KB  32.0 KB
  73.4   edayton    6.8 MB   0.0 B         0 139.3 KB/s 512.0 KB  32.0 KB
  73.5   edayton    6.8 MB   0.0 B         0 139.3 KB/s 512.0 KB  32.0 KB
  73.7   edayton           [ no i/o data collected yet ]

0 jobs; 0 idle, 0 running, 0 held
```

Policy Configuration

(Boss Fat Cat)



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

Topics for Tomorrow Morning…

The Machine (Startd) Policy Expressions

START – When is this machine willing to start a job

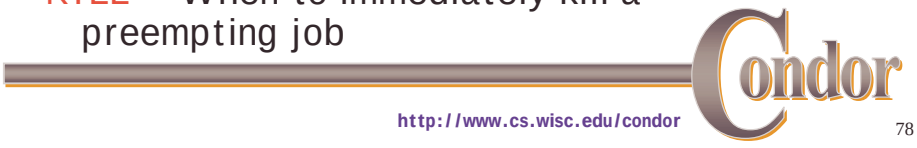
RANK - Job Preferences

SUSPEND - When to suspend a job

CONTINUE - When to continue a suspended job

PREEMPT – When to nicely stop running a job

KILL - When to immediately kill a preempting job



Freida's Current Settings

START = True
RANK =
SUSPEND = False
CONTINUE =
PREEMPT = False
KILL = False

Freida's New Settings for the Chemistry nodes

START = True
RANK = Department == "Chemistry"
SUSPEND = False
CONTINUE =
PREEMPT = False
KILL = False

Submit file with Custom Attribute

```
Executable = charm-run
Universe = standard
+Department = Chemistry
queue
```

What if "Department" not specified?

```
START = True
RANK  = Department =!= UNDEFINED &&
        Department == "Chemistry"
SUSPEND  = False
CONTINUE =
PREEMPT = False
KILL  = False
```

Another example

START = True

RANK = Department =!= UNDEFINED &&
((Department == "Chemistry")*2 +
Department == "Physics")

SUSPEND = False

CONTINUE =

PREEMPT = False

KILL = False

Policy Configuration, cont

(Boss Fat Cat)

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

Topics for Tomorrow Morning…

So Frieda decides she wants the desktops to:

- START jobs when their has been no activity on the keyboard/mouse for 5 minutes and the load average is low
- SUSPEND jobs as soon as activity is detected
- PREEMPT jobs if the activity continues for 5 minutes or more
- KILL jobs if they take more than 5 minutes to preempt

Desktop Machine Policy

```
START = $(CPU_Idle) && KeyboardIdle > 300
SUSPEND = $(MachineBusy)
CONTINUE = $(CPU_Idle) && KeyboardIdle >
  120
PREEMPT  = (Activity == "Suspended") &&
  $(ActivityTimer) > 300
KILL  = $(ActivityTimer) > 300
```

Policy Review

- Users submitting jobs can specify Requirements and Rank expressions
- Administrators can specify Startd Policy expressions individually for each machine (Start,Suspend,etc)
- Expressions can use any job or machine ClassAd attribute
- Custom attributes easily added
- Bottom Line: Enforce almost any policy!

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

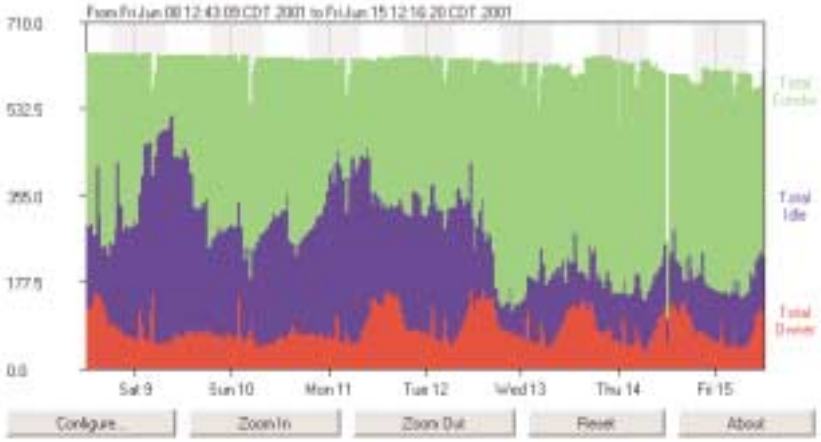
Administrator Commands

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

CondorView Usage Graph

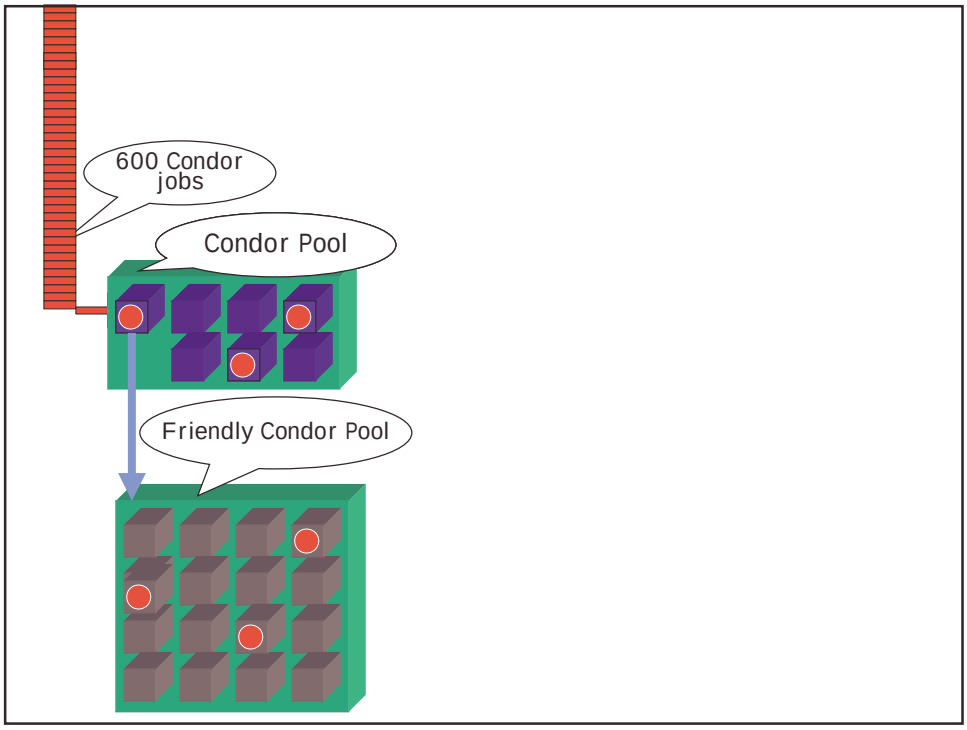


Back to the Story…

Frieda Needs Remote Resources…

Frieda Goes to the Grid!

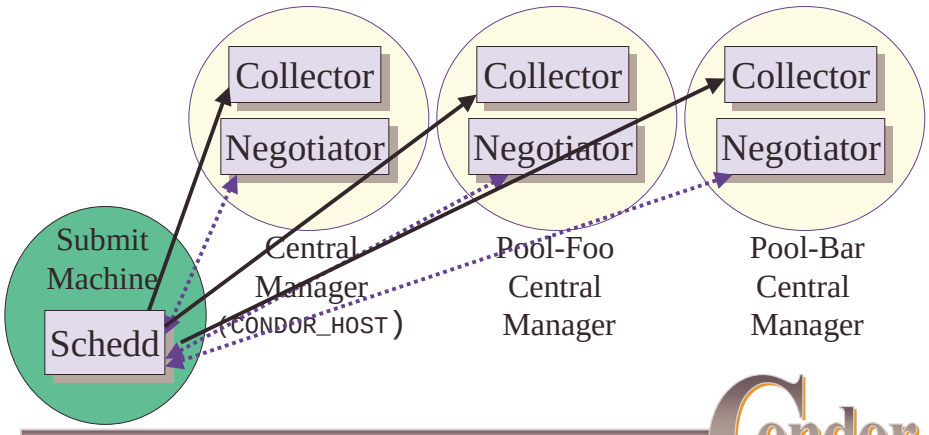
- First Frieda takes advantage of her Condor friends!
- She knows people with their own Condor pools, and gets permission to access their resources
- She then configures her Condor pool to "flock" to these pools



How Flocking Works

› Add a line to your condor_config :

```
FLOCK_HOSTS = Pool-Foo, Pool-Bar
```



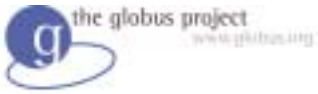
Condor Flocking

- 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



Globus

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

Condor

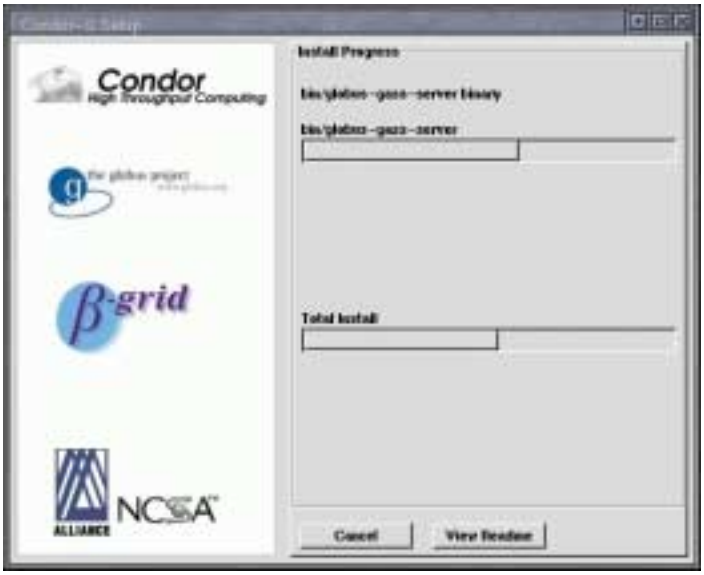
- job scheduling across multiple resources
- strong fault tolerance with checkpointing and migration
- layered over Globus as "personal batch system" for the Grid

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

99

Condor-G Installation: Tell it what you need…

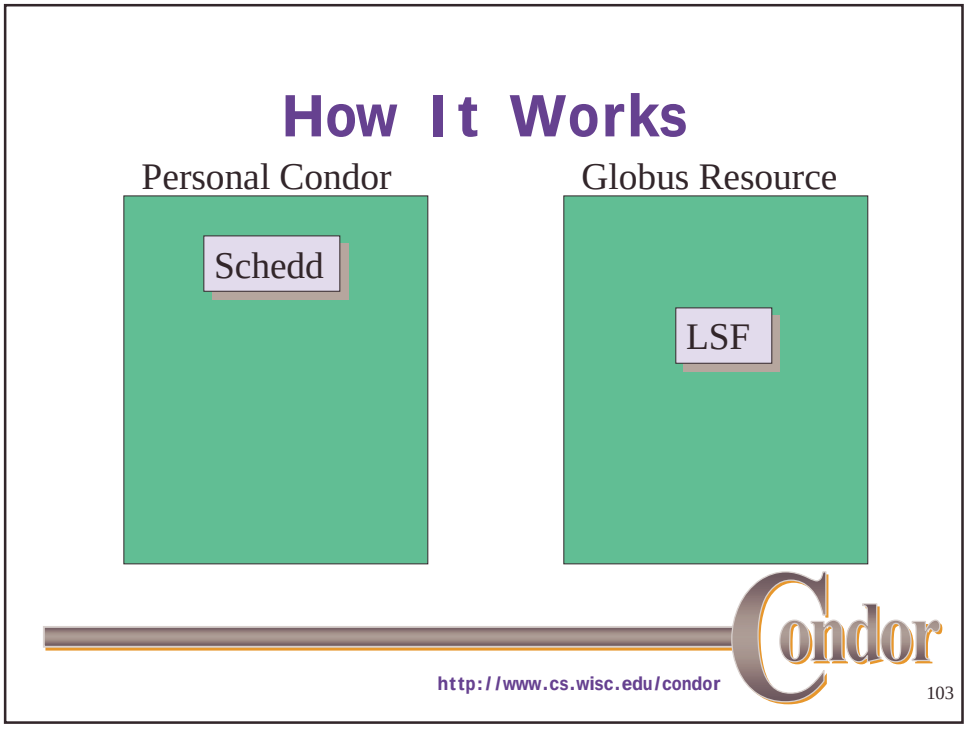
… and watch it go!

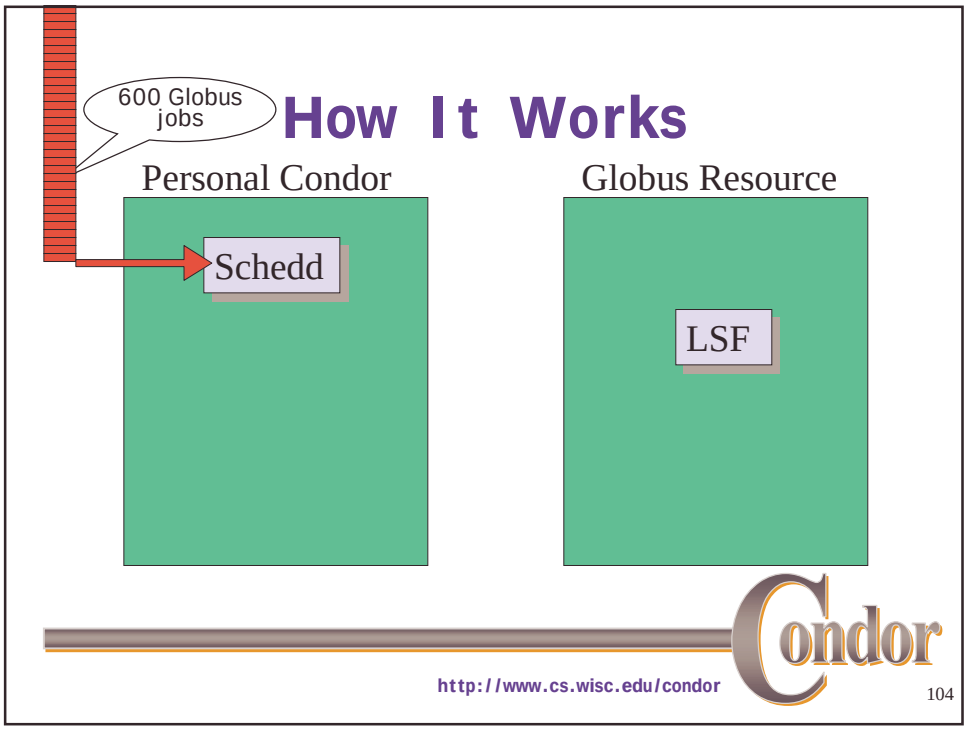


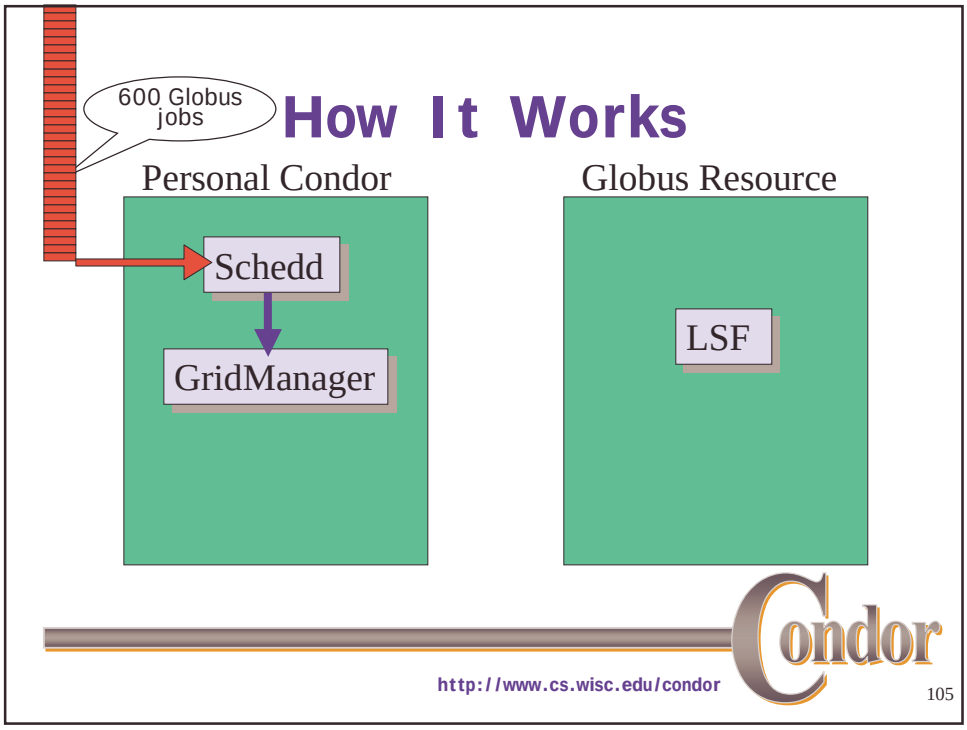
Frieda Submits a Globus Universe Job

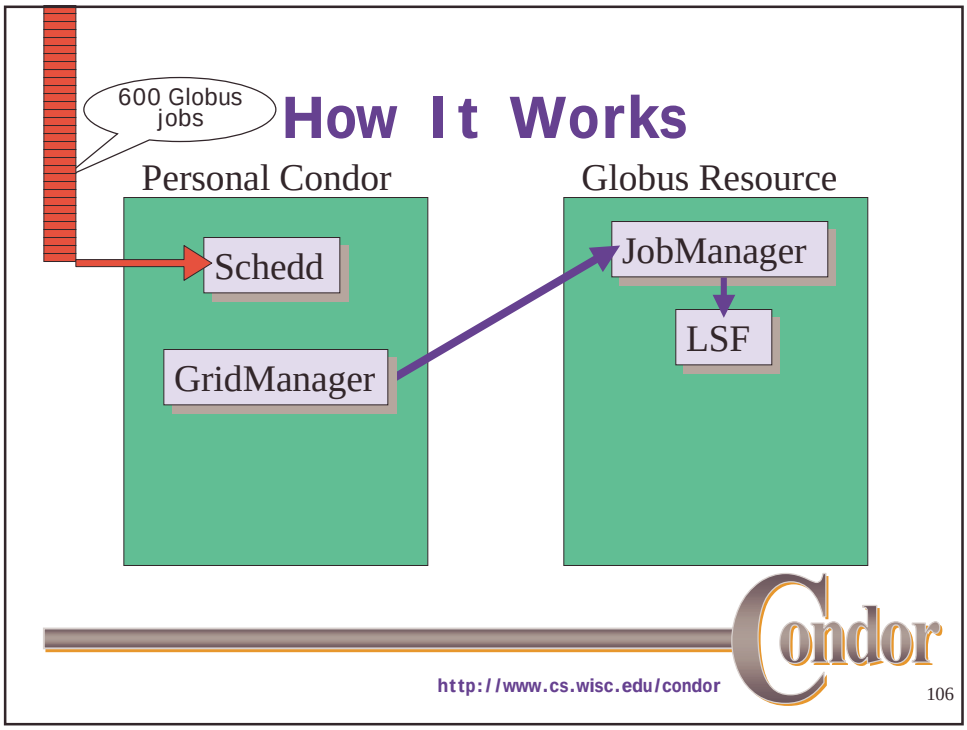
- In her submit description file, she specifies:
  - Universe = Globus
  - Which Globus Gatekeeper to use
  - Optional: Location of file containing your Globus certificate

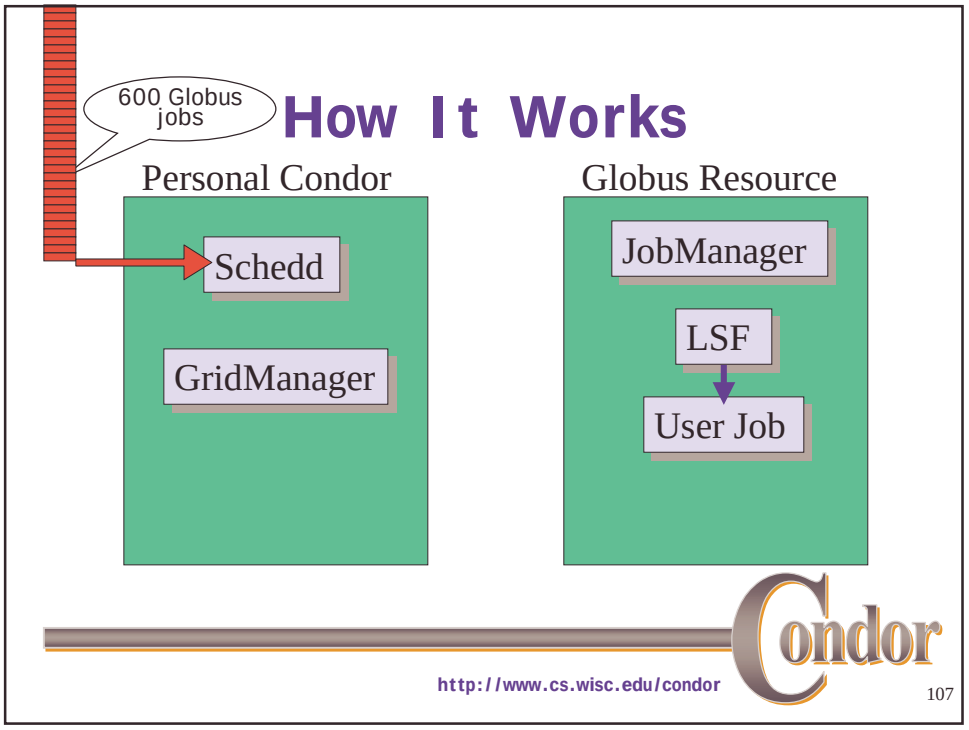
```
universe     = globus
globusscheduler = beak.cs.wisc.edu/jobmanager
executable   = progname
queue
```

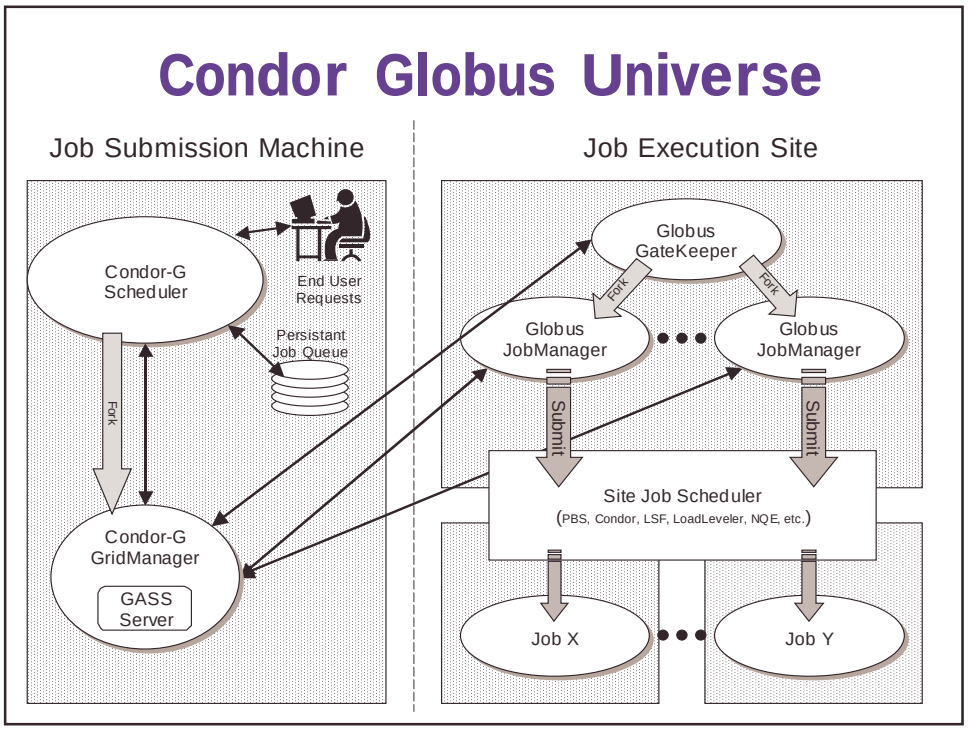

How It Works
Personal Condor
Globus Resource
Schedd
LSF
Condor
http://www.cs.wisc.edu/condor
103


600 Globus jobs
How It Works
Personal Condor
Globus Resource
Schedd
LSF
Condor
http://www.cs.wisc.edu/condor
104


600 Globus jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
LSF
Condor
http://www.cs.wisc.edu/condor
105

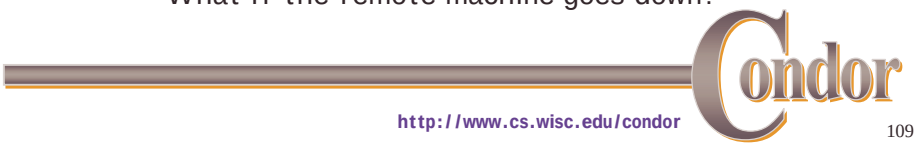

600 Globus jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
JobManager
LSF
Condor
http://www.cs.wisc.edu/condor
106


600 Globus jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
JobManager
LSF
User Job
Condor
http://www.cs.wisc.edu/condor
107


Condor Globus Universe
Job Submission Machine
Job Execution Site
Condor-G Scheduler
End User Requests
Persistant Job Queue
Fork
Condor-G GridManager
GASS Server
Globus GateKeeper
Fork
Fork
Globus JobManager
Globus JobManager
Submit
Submit
Site Job Scheduler
(PBS, Condor, LSF, LoadLeveler, NQE, etc.)
Job X
Job Y

Globus Universe Concerns

- What about Fault Tolerance?
  - Local Crashes
    - What if the submit machine goes down?
  - Network Outages
    - What if the connection to the remote Globus jobmanager is lost?
  - Remote Crashes
    - What if the remote Globus jobmanager crashes?
    - What if the remote machine goes down?



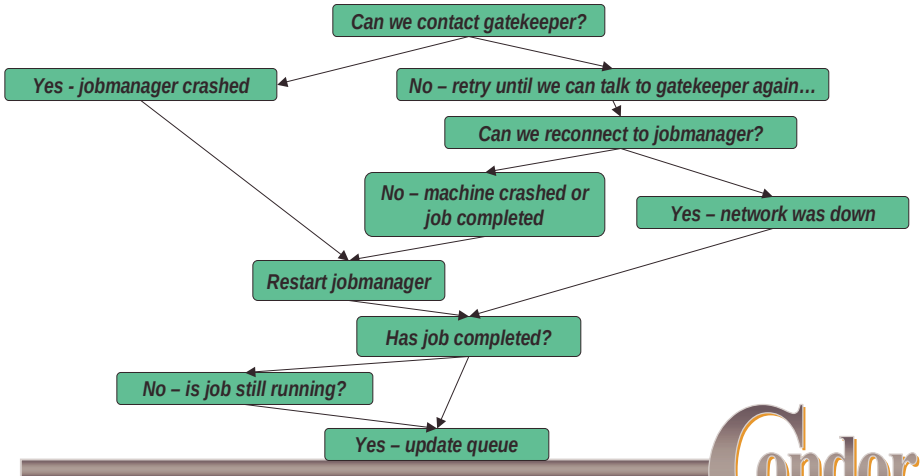
Changes to the Globus JobManager for Fault Tolerance

- Ability to restart a JobManager
- Enhanced two-phase commit submit protocol

Globus Universe Fault-Tolerance: Submit-side Failures

- All relevant state for each submitted job is stored persistently in the Condor job queue.
- This persistent information allows the Condor GridManager upon restart to read the state information and reconnect to JobManagers that were running at the time of the crash.
- If a JobManager fails to respond...

Globus Universe Fault-Tolerance: Lost Contact with Remote Jobmanager



Globus Universe Fault-Tolerance: Credential Management

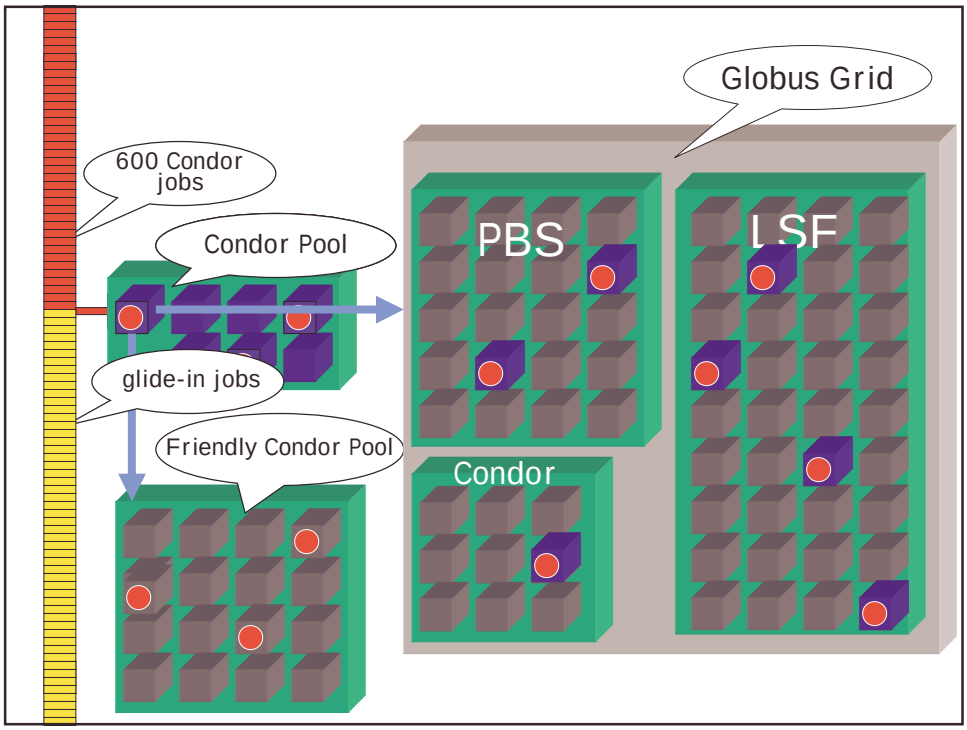
- Authentication in Globus is done with limited-lifetime X509 proxies
- Proxy may expire before jobs finish executing
- Condor can put jobs on hold and email user to refresh proxy
- Todo: Interface with MyProxy…

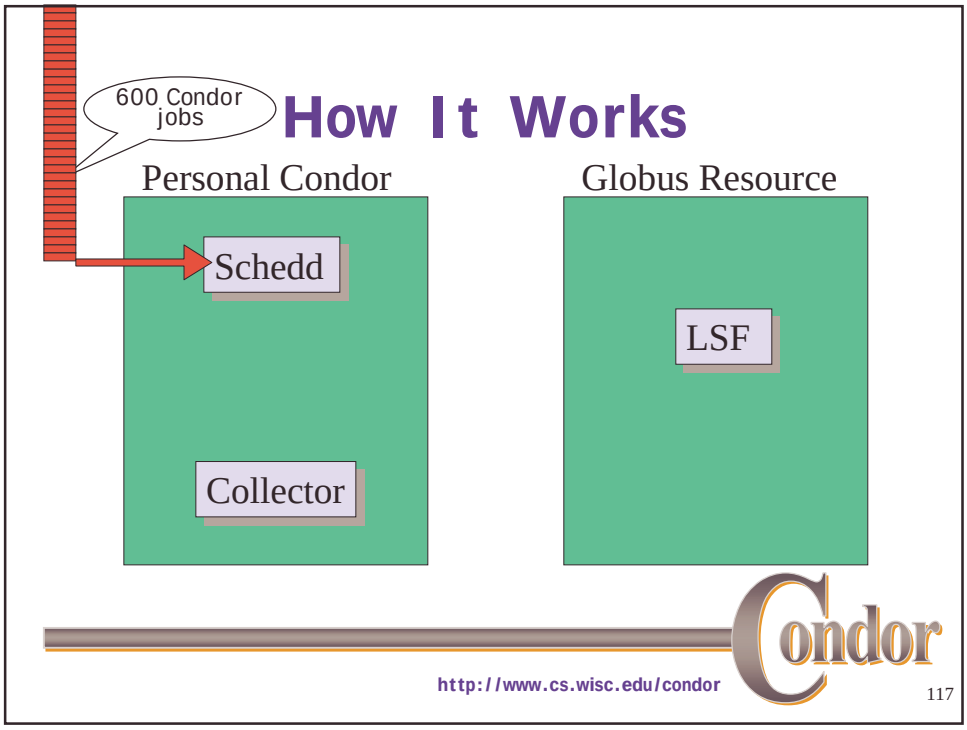
But Frieda Wants More…

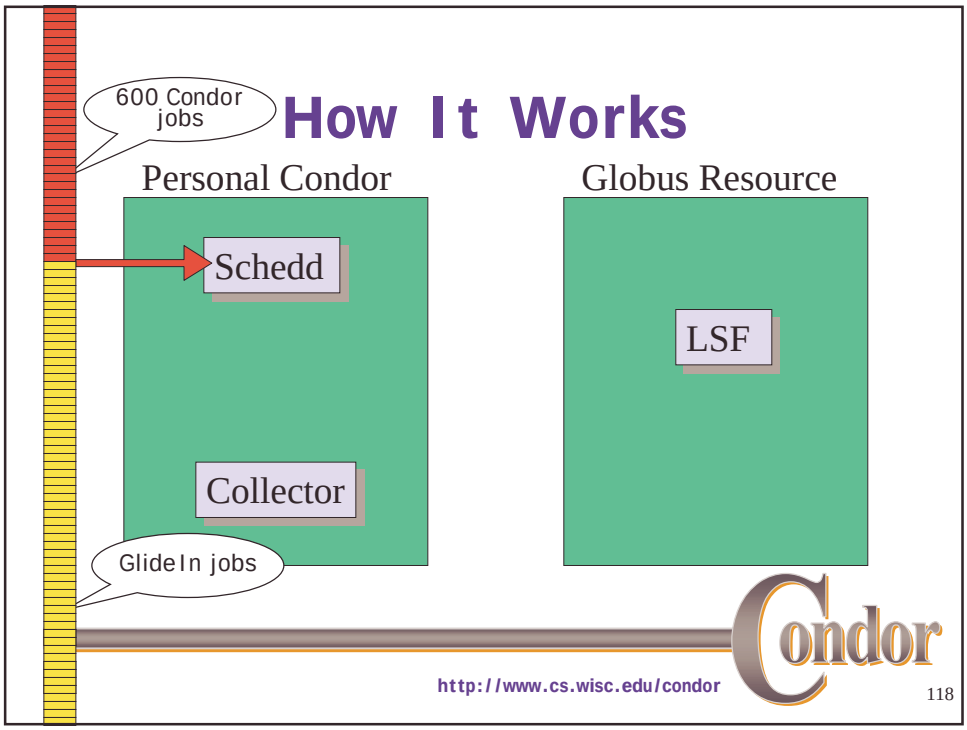
- She wants to run standard universe jobs on Globus-managed resources
  - For matchmaking and dynamic scheduling of jobs
  - For job checkpointing and migration
  - For remote system calls

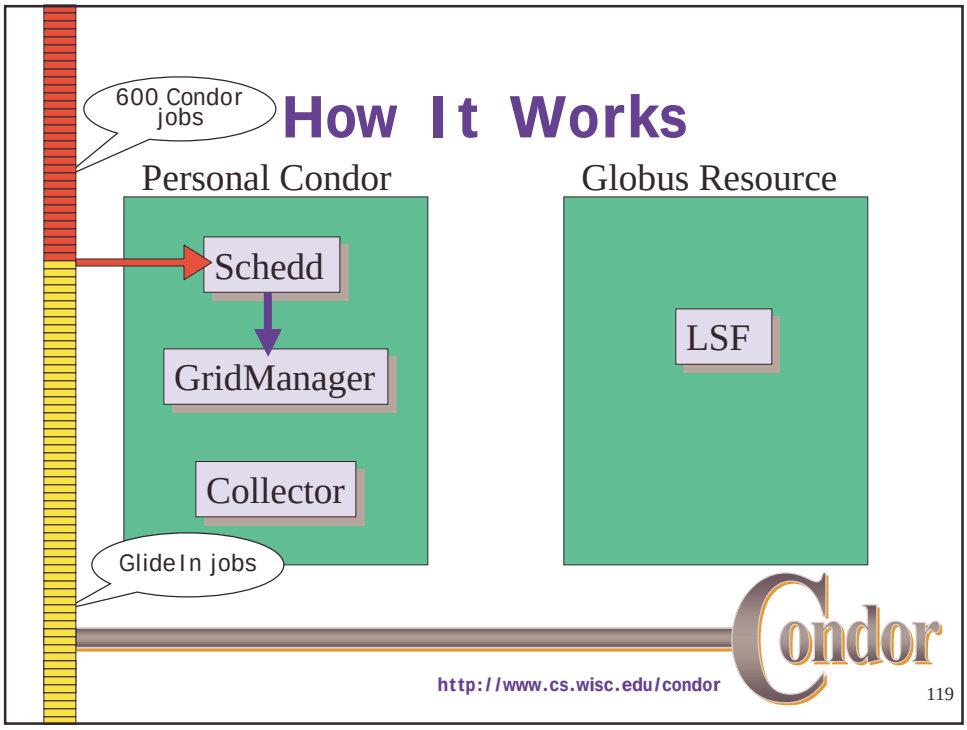
Solution: Condor GlideIn

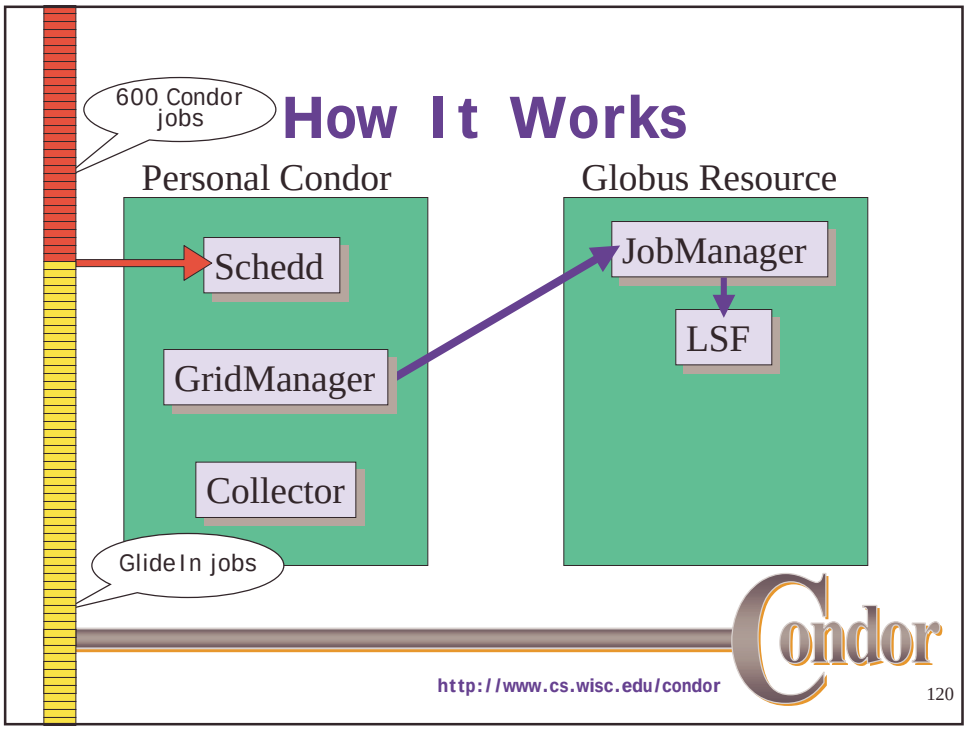
- Frieda can use the Globus Universe to run Condor daemons on Globus resources
- When the resources run these GlideIn jobs, they will temporarily join her Condor Pool
- She can then submit Standard, Vanilla, PVM, or MPI Universe jobs and they will be matched and run on the Globus resources

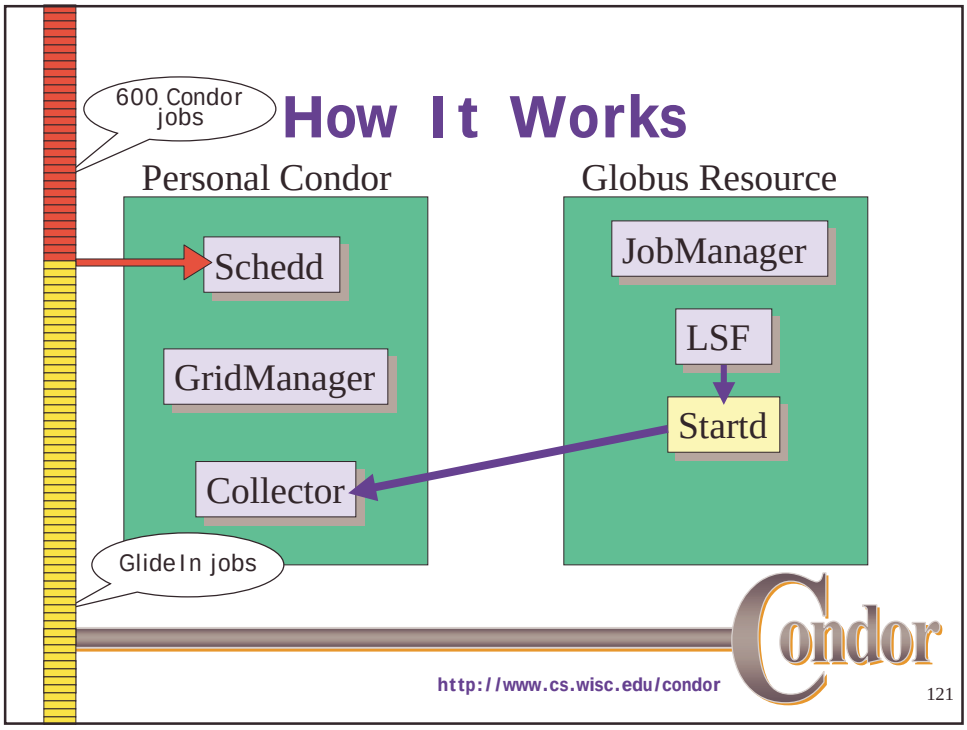


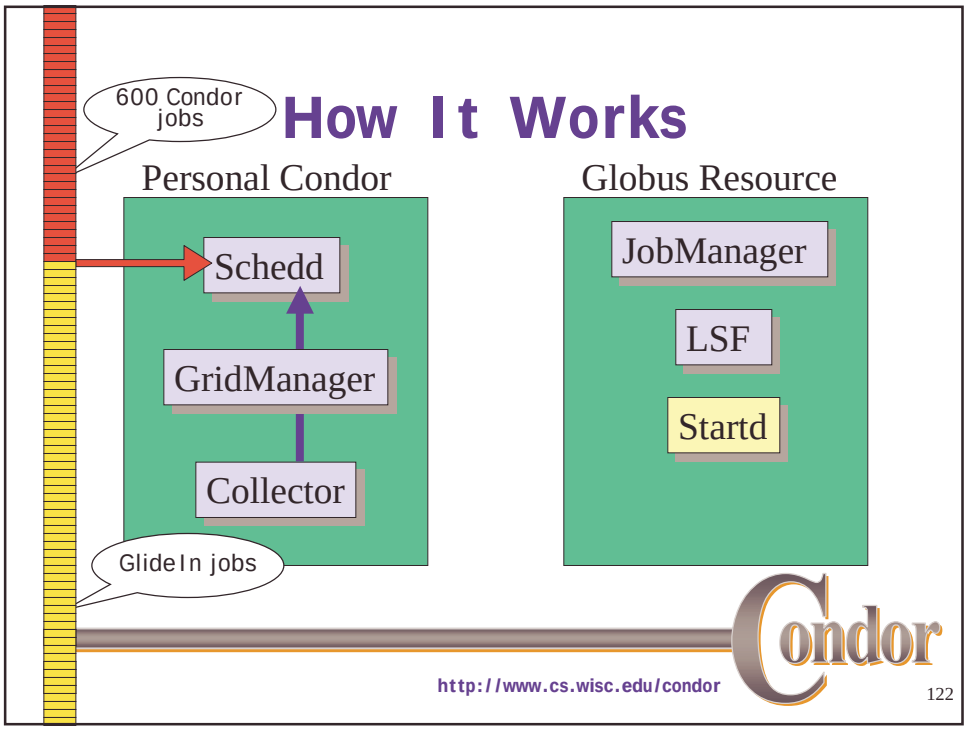

600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
LSF
Collector
Condor
http://www.cs.wisc.edu/condor
117

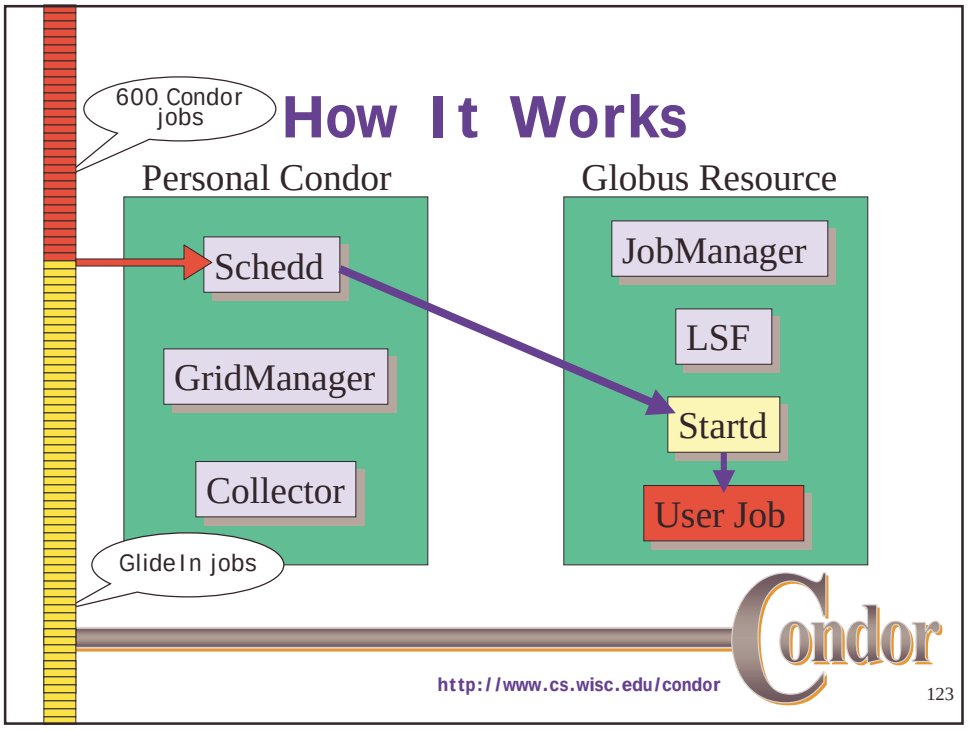

600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
LSF
Collector
GlideIn jobs
Condor
http://www.cs.wisc.edu/condor
118

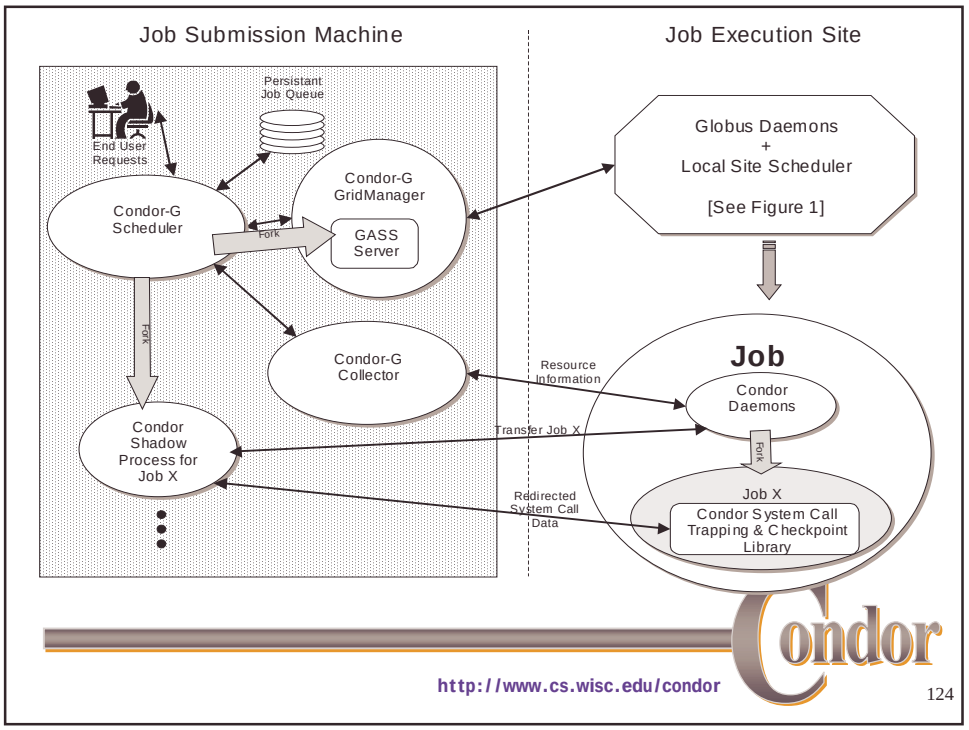

600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
Collector
LSF
GlideIn jobs
Condor
http://www.cs.wisc.edu/condor
119


600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
JobManager
LSF
GridManager
Collector
GlideIn jobs
Condor
http://www.cs.wisc.edu/condor
120


600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
Collector
JobManager
LSF
Startd
GlideIn jobs
Condor
http://www.cs.wisc.edu/condor
121


600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
Collector
JobManager
LSF
Startd
GlideIn jobs
Condor
http://www.cs.wisc.edu/condor
122


600 Condor jobs
How It Works
Personal Condor
Globus Resource
Schedd
GridManager
Collector
JobManager
LSF
Startd
User Job
GlideIn jobs
Condor
http://www.cs.wisc.edu/condor
123


Job Submission Machine
Job Execution Site
Persistant Job Queue
End User Requests
Condor-G Scheduler
Condor-G GridManager
GASS Server
Fork
Condor-G Collector
Condor Shadow Process for Job X
Globus Daemons + Local Site Scheduler
[See Figure 1]
Job
Resource Information
Condor Daemons
Transfer Job X
Job X
Redirected System Call Data
Condor System Call Trapping & Checkpoint Library
Condor
http://www.cs.wisc.edu/condor
124

GlideIn Concerns

- What if a Globus resource kills my GlideIn job?
  - That resource will disappear from your pool and your jobs will be rescheduled on other machines
  - Standard universe jobs will resume from their last checkpoint like usual
- What if all my jobs are completed before a GlideIn job runs?
  - If a GlideIn Condor daemon is not matched with a job in 10 minutes, it terminates, freeing the resource

A Common Question

My Personal Condor is flocking with a bunch of Solaris machines, and also doing a GlideIn to a Silicon Graphics O2K. I do not want to statically partition my jobs.

Solution: In your submit file, say:

```
Executable = myjob.$$(OpSys).$$(Arch)
```

The "$$(xxx)" notation is replaced with attributes from the machine ClassAd which was matched with your job.

In Review

With Condor Frieda can…

- … manage her compute job workload
- … access local machines
- … access remote Condor Pools via flocking
- … access remote compute resources on the Grid via Globus Universe jobs
- … carve out her own personal Condor Pool from the Grid with GlideIn technology

Thank you!

Check us out on the Web:

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

Email:

condor-admin@cs.wisc.edu