Grid Compute Resources and Job Management[1]

Requesting a new certificate with the DOEGrids

- Issue the following command:

```
cert-request -agree -ou p -name "Your name" -email user@umsl.edu \
  -phone "+314-xxx-xxxx" -reason "Grid school" -affiliation osg -vo osgedu \
  -sponsor_name "Alina Bejan" -sponsor_email "abejan@ci.uchicago.edu" \
  -sponsor_phone "+1-773-702-3492"
```

- You will be asked for a secret passphrase. Choose one.
  - Do not forget the passphrase; it cannot be retrieved or rest if lost
- Retrieving a new certificate with the DOEGrids
  - After making the request, wait for a while for the request to be approved (typically a day)
  - You will receive an approval email, containing a serial number

```
Your Personal certificate request has been processed successfully.
SubjectDN= CN=Sanjiv K. Bhatia 599801,OU=People,DC=doegrids,DC=org
notAfter= Sep 16, 2009 4:26:34 PM
notBefore= Sep 16, 2008 4:26:34 PM
Serial Number= 0x6710
```

  - Retrieve your credentials by

```
$ cert-retrieve -serialnum 26384
checking CertLib version, V2-6,  This is the latest version, released 13 Feb 2008.
 using CA doegrids
Using URL https://pki1.doegrids.org/displayBySerial?op=displayBySerial&serialNumber=2638
Checking that the certificate and /home/sanjiv/.globus/2273key.pem match
Enter pass phrase for /home/sanjiv/.globus/2273key.pem:
writing RSA key
/home/sanjiv/.globus/usercert.pem and /home/sanjiv/.globus/userkey.pem now
contain your new credential
```

    - Please enter the passphrase when asked
- Installing your own credentials
  - If you have your own credential, install it into your account by placing two files `usercert.pem` and `userkey.pem` into the directory `~/.globus` in your account by using `scp` or `sftp`
- Creating a proxy
  - Done after installing credential
  - Grid proxy contains everything necessary to authenticate you to grid resources
  - Create proxy using the command `grid-proxy-init`

```
$ grid-proxy-init
Your identity: /DC=org/DC=doegrids/OU=People/CN=Sanjiv K. Bhatia 599801
Enter GRID pass phrase for this identity:
Creating proxy ................................. Done
Your proxy is valid until: Wed Sep 17 06:33:40 2008
```

[1] Most of the material in this set of notes is from the Educational division of Open Science Grid.

        * Again, enter the passphrase that you created

Accessing the grid

- Command line with tools that can be used
  - Simple command

    ```
    globus-job-run osg-grid1 /bin/echo Hello World
    ```
- Specialized applications
  - Write a program to process images that sends data to run on the grid as an inbuilt feature
- Web portals
  - I2U2
    * Interactions In Understanding the Universe
  - SIDGrid
    * Social Informatics Data Grid

Grid middleware

- Glues the grid together
- Software that glues together different clusters into a grid, taking into consideration the socio-political side of things
  - Common policies on who can use what, how much, and what for
- Offers services that couple *users* with *remote resources* through *resource brokers*
- Remote process management
- Co-allocation of resources
- Storage access
- Information
- Security
- QoS

Globus toolkit

- Standard for grid middleware
- Developed at Argonne National Lab and University of Chicago (Globus Alliance)
- Open source
- Adopted by different scientific communities and industries
- Conceived as an open set of architectures, services, and software libraries to support grid and grid applications
- Provides services in major areas of distributed systems
  - Core services

  - Data management
  - Security

Running grid jobs with Globus commands

- Simple command

```
globus-job-run osg-grid1 /bin/echo Hello World
```

- Another command

```
globus-job-run osg-grid1/jobmanager-fork /bin/echo Hello World
```

- Need to specify the complete path for the command
  - Determine the complete path by using the command `which` or `type`
  - Execute the commands `hostname`, `id`, `env`, `ps`, and `uptime` on the grid
  - Do an `ls` on your working directory
  - Use `df` to discover how much storage space exists in the remote `/tmp` directory. Can you create a file on the remote system? Can you remove it?
- Running under a remote shell
  - `globus-job-run` does not start a Unix shell on remote server
  - Need to specify fully qualified path names (no `$PATH` variable)
  - No I/O redirection or `$VAR` substitution
  - Can tell `globus-job-run` to run a shell on remote grid site and pass a command string to that shell

```
$ globus-job-run osg-grid1.umsl.edu /bin/sh -c \
"grep osgedu /etc/passwd | wc -l"
```

  - Common Linux system commands are typically found in the same directory on all Linux systems, so you can expect the path to be the same on every system
  - Application specific software is usually installed in different places on each system
    - Virtual organizations often establish conventions for such things

Globus – Core services

- Basic infrastructure needed to create grid services
- Authorization
- Message level security
- System level services (monitoring)
- Associated data management provides file services
  - GridFTP
    - High performance, secure, and reliable data transfer protocol based on standard FTP
    - Enhances FTP with
      - GSI security
      - Multiple data channels for parallel transfers

            - Third party transfers
            - Authenticated reusable channels
    - RFT – Reliable file transfer
        - Protocol to provide reliability and fault tolerance for file transfers
    - RLS – Replica location service
        - Component of data grid architecture
        - Provides access to mapping information from logical names to physical names of items
        - Goals
            - Reduce access latency
            - Improve data locality
            - Improve robustness, scalability, and performance for distributed applications
        - Produces local replica catalogs (LRCs) to represent mappings between logical and physical files scattered across the storage system
            - LRCs may be indexed for performance reasons
- Uses GT4
    - Promotes open HPC

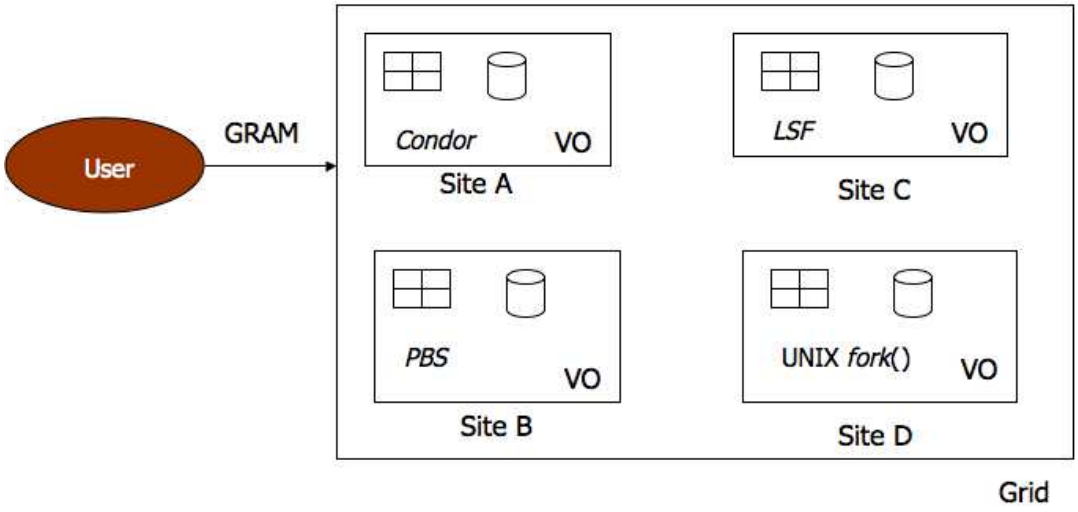
Local Resource Managers – (LRMs)

- Deals with job submission
    - Users want to run their programs – data processing, simulation
    - Run jobs on other computers on the grid to make use of those resources
    - Problem: Sites on the grid may have different platforms
        - Must deploy a uniform way to access the grid
- LRM manages local resources
    - Only one user is allowed to use ecah compute node at a time, with some kind of queue
- Not grid-specific
    - LRM manages jobs on a particular cluster
    - LRM doesn't deal with jobs coming in from elsewhere
    - You need to log in directly to the cluster to submit your jobs
- Every node has an LRM that controls
    - Who is allowed to run jobs?
    - How jobs run on a specific resource?
    - Specific order and location of jobs
- Example policy
    - Each cluster node can run one job
    - If there are more jobs, they must wait in queue
- May allow a set of nodes in the cluster to be reserved for a specific person
- Examples of LRMs are: PBS (Portable Batch System), LSF, Condor

- More advanced LRMs may deal with reservations, prioritising between different users and other parameters

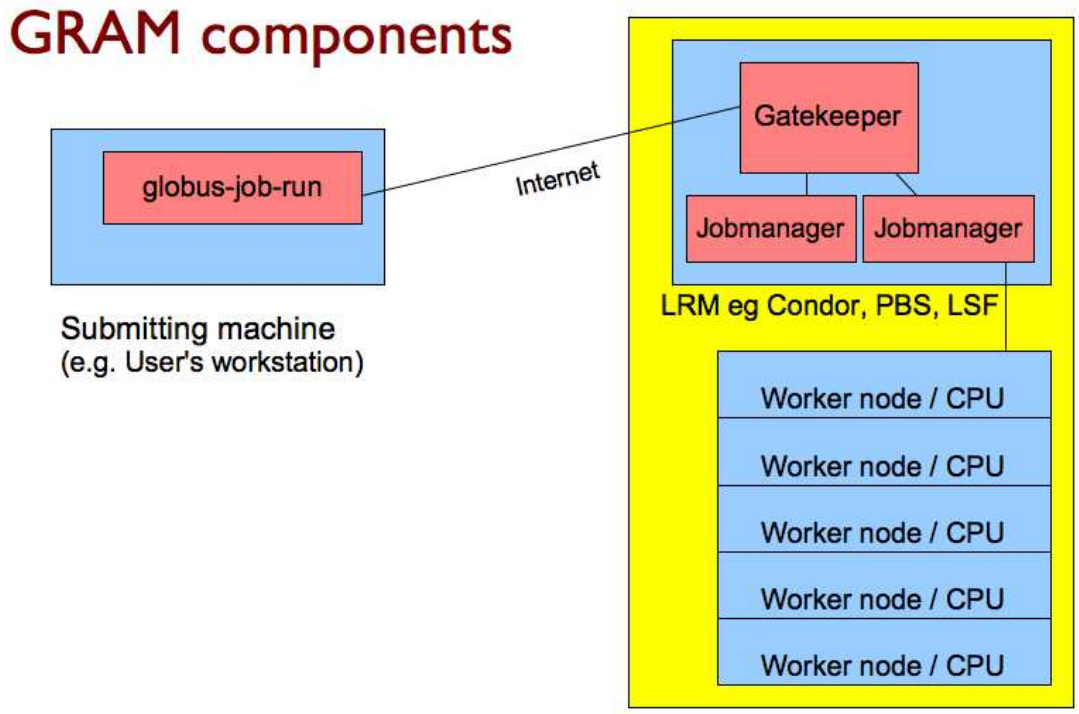
GRAM – Globus Resource Allocation Manager

- Provides a standardized interface to submit jobs to LRMs
- Clients submit a job request to GRAM
- GRAM translates the job request to something a[ny] LRM can understand
  - Same job request can be used for many different kinds of LRMs

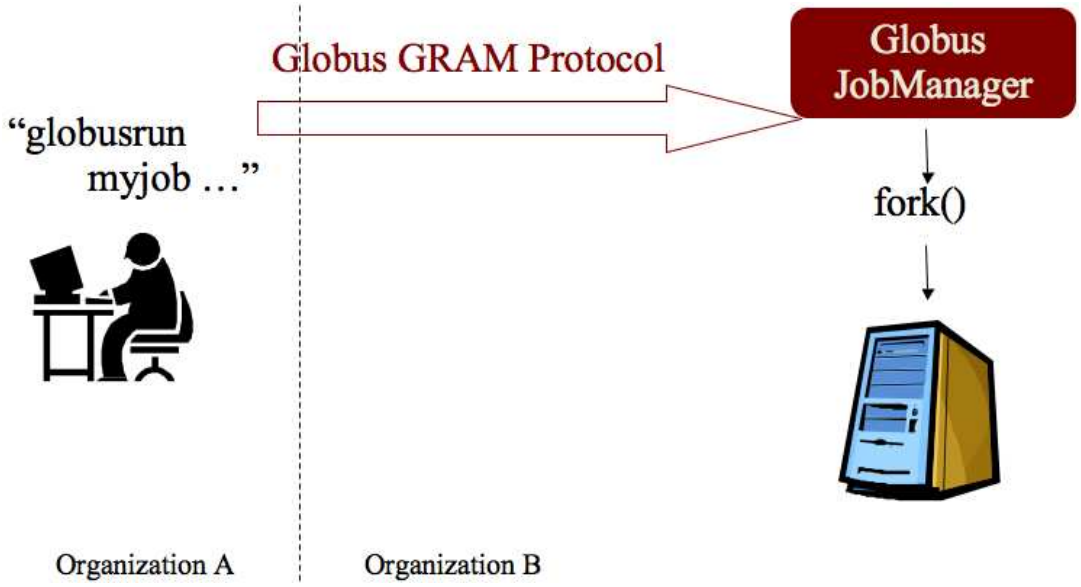


  - Each cluster has a local LRM
  - Each site can have a different LRM, chosen by the site admin
- GRAM does not manage job execution; LRM does that
- GRAM version GT2
  - Own protocols
  - More widely used
  - No longer actively developed
- GRAM version GT4
  - Web services
  - New features go into this version
- GRAM's abilities – Given a job specification
  - Creates an environment for the job
  - Stages files to/from the environment
  - Submits a job to an LRM
  - Monitors a job
  - Sends notification of the job state change
  - Streams a job's stdout/stderr during execution

- Components of GRAM
  - Clients
    - `globus-job-submit`, `globus-run`
  - Gatekeeper
    - Server
    - Accepts job submissions
    - Handles security
  - Job manager
    - Knows how to send a job into the LRM
    - Different job managers for different LRMs



- Remote resource access



- Submitting a job with GRAM
  - `globus-job-run`

    ```
    globus-job-run valhalla.umsl.edu /bin/hostname
    ```
  - We don't care what LRM is used on `valhalla`; this command works with any LRM
- Describing a job with GRAM's rsl (Resource Specification Language)
  - Example file `spec.rsl`

    ```
    &(executable = a.out)
    (directory = /home/nobody)
    (arguments = arg1 "arg 2")
    ```
  - Submit the job with

    ```
    globusrun -f spec.rsl -r valhalla.umsl.edu
    ```
- Generating RSL using other programs
  - RSL job descriptions can become very complicated
  - Use other programs to generate RSL for us, such as Condor-G

Immediate and batch job managers

- GRAM supports the concept of a job manager as an adapter to LRMs
- Each site can support one or more job managers
- Our grid supports two managers
  - `fork` job manager runs a job immediately through the Unix `fork()` interface
  - Condor job manager acts as an interface to the Condor batch scheduling system
- Use the command `time` to find which job manager is faster
- `fork` job manager

  - Very fast
  - Low scheduling latency
  - Runs trivial commands very quickly
  - No compute power; just a single CPU on a cluster-controlling computer called the *head node*
- Write a program in C to find if a given number is a prime. Run it on a remote machine using the grid.

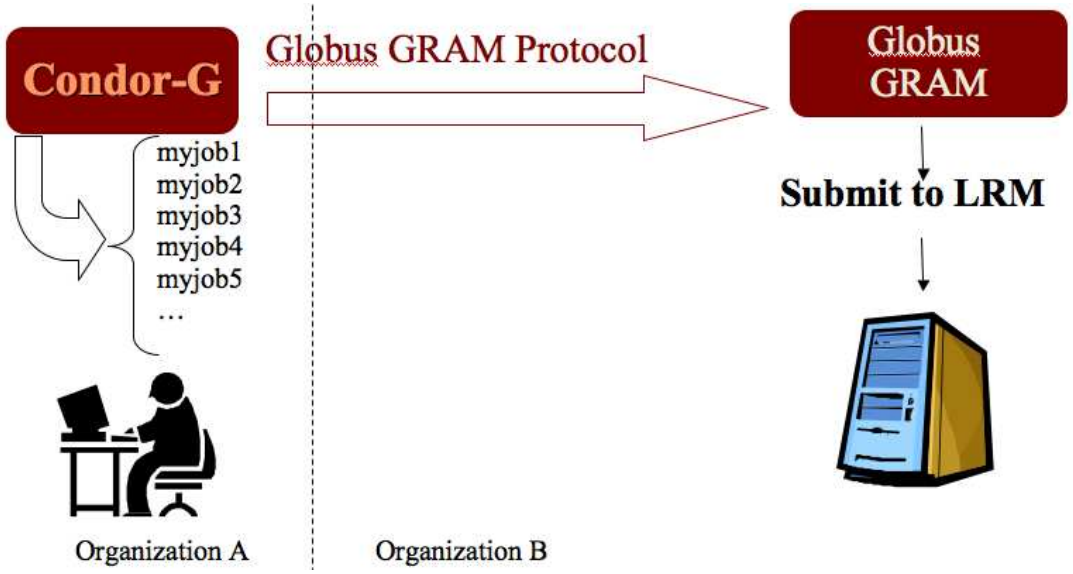
Condor

- Batch job manager
  - Higher scheduling overhead
  - Gives you access to all computers in a cluster; opportunity to do real parallel computing
- Specialized workload management system for compute-intensive jobs
  - Batch system to provide a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management
  - Users submit serial or parallel jobs to Condor
  - Condor places them into a queue
  - Chooses when and where to run the jobs based upon a policy
  - Carefully monitors their progress, and ultimately informs the user upon completion
- Software system to create an HTC (High Throughput Computer) environment
  - Created at UW-Madison
  - Detects machine availability
  - Harnesses available resources
  - Uses remote system calls to send R/W operations over the net
  - Provides powerful resource management by *matching* resource owners with consumers (broker)
- Batch job system
  - Can take advantage of both dedicated and non-dedicated computers to run jobs
  - Focus on high-throughput rather than high-performance
  - Provides a wide variety of features including checkpointing, transparent process migration, remote I/O, parallel programming with MPI, and ability to run large workflows
  - Condor-G
    - Designed to specifically interact with Globus
    - Can provide a resource selection service to different and multiple grid sites
- Condor – features
  - Checkpoint and migration
  - Remote system calls
    - Able to transfer data files and executables across machines
  - Job ordering
    - Job requirements and preferences can be specified via powerful expressions
- Managing a large number of jobs

  - Specify the jobs in a file and submit them to Condor
  - Condor runs the jobs and keeps users notified on their progress
    * Mechanisms to help you manage huge numbers of jobs and corresponding data
    * Handles inter-job dependencies (DAGMan)
  - Submitting jobs to the grid
    * Commonly done as a large number of separate job submissions
    * If you want to use 100 CPUs, make a few hunder submissions, each one intended to use its own CPU
    * Hard to manage without software such as Condor
    * Condor will track the progress, restart jobs if they fail, handle dependencies between different jobs so that some jobs don't start until some other job is finished
  - Users can set Condor's job priorities
  - Condor admninistrators can set user priorities
  - Can do this as
    * LRM on a compute resource
    * Grid client, submitting to GRAM as Condor-G

- Condor-G
  - Job management part of Condor
  - Can be used to submit to resources accessible through a Globus interface
  - Does not *create* a grid service
  - Only deals with *using* remote grid services
  - Does whatever it takes to run your jobs, even if
    * Gatekeeper is temporarily unavailable
    * Job manager crashes
    * Local machine crashes
    * Network goes down

- Remote resource access: Condor-G + Globus + Condor



- Access non-Condor grid resources

Globus Project	Condor
Middleware deployed across entire grid	Job scheduling across multiple resource
Remote access to computational resources	Strong fault tolerance with checkpointing and migration
Dependable, robust data transfer	Layered over Globus as “personal batch system” for the grid

- Four steps to run a job with Condor
  - The choices tell Condor
    - **How**
    - **When**
    - **Where** to run the job
    - and describe exactly **what** you want to run

1. Choose a Universe for your job
   - Job Universe lets Condor know how to launch and manage your type of job, as well as what type of runtime environment to set up or expect
   - Many possible choices for Universe
     - Vanilla: Any odd job
     - Grid: Run jobs on grid
     - Standard: Checkpointing and remote I/O
     - Java: For Java jobs
     - MPI: Parallel MPI jobs
     - Virtual machine: Run a virtual machine as job
2. Make your job *batch-ready*
   - Job must be able to run in the background
     - No interactive input, windows, or GUI
   - Condor is designed to run jobs as a batch system, with pre-defined inputs for jobs
   - Can still use `stdin`, `stdout`, and `stderr` but they must be redirected to files
   - Organize data files
3. Create a *submit description* file
   - Plain ASCII text file
     - Syntax is fairly simple as *Key = value*
   - Condor does not care about file extensions
   - Used to tell Condor about the job
     - Which executable to run and where to find it
     - Which universe
     - Location of input, output, and error files
     - Command-line arguments, if any
     - Environment variables
     - Any special requirements or preferences
   - Condor uses this information to build a ClassAd for the job
   - Example file – named `myjob.submit`

```
# myjob.submit file
# Simple condor_submit input file
# (Lines beginning with # are comments
# Note: The words on the left are not case sensitive, but filenames are!
Universe      = vanilla
Executable    = analysis
Log           = my_job.log
Queue
```

        * The command `queue` tells Condor to submit this job into the job queue
    4. Run *condor_submit*
        - Run it by the command
          `condor_submit my_job.submit`
        - `condor_submit` parses the submit file
        - Performs the conversion of submit file into ClassAd, and places it into the job queue

- Another submit description file

```
# Example condor_submit input file
# (Lines beginning with # are comments)
# Note: The words on the left are not case sensitive, but filenames are!
Universe      = vanilla
Executable    = /home/sanjiv/condor/my_job.condor
Input         = my_job.stdin
Output        = myjob.stdout
Error         = my_job.stderr
Arguments     = -arg1 -arg2
InitialDir    = /home/sanjiv/condor/run_1
Queue
```

- Lots of options available in submit file
    - Commands to
        * Watch the queue
        * State of your pool
        * Lots more
- Other Condor commands
    - `condor_q` – Show status of job queue
    - `condor_status` – Show status of compute nodes
    - `condor_rm` – Remove a job
    - `condor_hold` – Hold a job temporarily
    - `condor_release` – Release a job from hold
- Practicing with Condor
    1. Check the Condor queue with `condor_q`
        - Condor is already set up and running on `osg-grid1`
        - Check this by running `condor_q`

```
[sanjiv@osg-grid1 ~]$ condor_q

-- Submitter: osg-grid1.umsl.edu : <134.124.46.21:30157> : osg-grid1.umsl.edu
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

0 jobs; 0 idle, 0 running, 0 held
[sanjiv@osg-grid1 ~]$
```

        - When you submit jobs using Condor, you will see your jobs listed in the output of `condor_q`; you may see the jobs of other users as well
    2. Create your working directories
        - Make some directories to work in

```
$ cd
$ mkdir condor-tutorial
$ cd condor-tutorial
$ mkdir submit
```

- Submit a simple job with Condor-G
  - Create the Submit file

```
$ cd ~/condor-tutorial/submit
$ cat > myjob.submit
executable=/bin/echo
arguments="Hello world"
output=results.out
error=results.err
log=results.log
notification=never
universe=grid
grid_resource=gt2 osg-grid1.umsl.edu/jobmanager-condor
queue
ctrl-D
```

  - Submitting the test job

```
[sanjiv@osg-grid1 ~]$ condor_submit myjob.submit
Submitting job(s).
1 job(s) submitted to cluster 412.
[sanjiv@osg-grid1 ~]$
```

  - Run `condor_q` to see the progress of the job

```
[sanjiv@osg-grid1 ~]$ condor_q

-- Submitter: osg-grid1.umsl.edu : <134.124.46.21:30157> : osg-grid1.umsl.edu
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 412.0   sanjiv          9/30 16:44   0+00:00:00 I  0   0.0  echo
 413.0   osgedu          9/30 16:44   0+00:00:00 I  0   0.0  data Hello world
 414.0   osgedu          9/30 16:44   0+00:01:19 R  0   0.0  data

3 jobs; 2 idle, 1 running, 0 held
[sanjiv@osg-grid1 ~]$
```

  - Run `condor_q -globus` at regular intervals to see Globus-specific status information

```
[sanjiv@osg-grid1 ~]$ condor_q -globus

-- Submitter: osg-grid1.umsl.edu : <134.124.46.21:30157> : osg-grid1.umsl.edu
 ID      OWNER          STATUS  MANAGER  HOST                 EXECUTABLE
 412.0   sanjiv        PENDING condor   osg-grid1.umsl.edu  /bin/echo
[sanjiv@osg-grid1 ~]$
```

  - Monitoring progress with `tail`
    - In another window, run `tail -f` on the log file for your job to monitor progress
  - Verifying completed jobs
    - When the job is no longer listed in `condor_q`, or when the log file reports job terminated, the results can be viewed using `condor_history`
  - When the job completes, verify the output is as expected by looking in output file
  - Held jobs in Condor

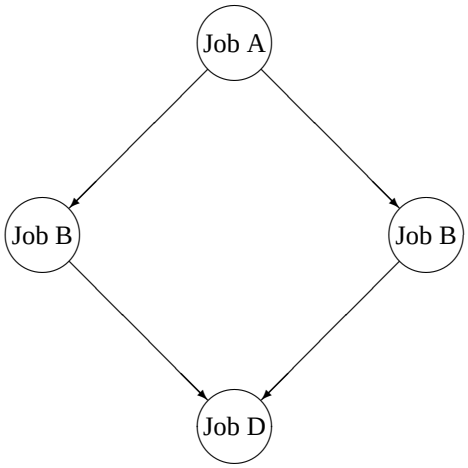
    - When a problem occurs in middleware, Condor-G will *hold* your job
    - Held jobs will remain in the queue, waiting for user intervention
    - When you resolve the problem, you can use `condor_release` to free the job to continue
    - Let us make the output file non-writeable

```
[sanjiv@osg-grid1 test3]$ condor_submit myjob.submit ; chmod a-w results.out
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 509.
[sanjiv@osg-grid1 test3]$
```

    - When the job is done, it goes into the `H` or "held" state
    - Fix the problem by making the file writeable and release the job

```
[sanjiv@osg-grid1 test3]$ chmod u+w results.out
[sanjiv@osg-grid1 test3]$ condor_release -all
All jobs released
```

- Submitting more complex jobs
  - Express dependencies between jobs; Workflows
  - Workflow should be managed even in the case of a failure
  - Scheduler Universe
    - Another job universe in addition to Vanilla
    - Runs on the submitting machine and serves as a meta-scheduler
    - Lets you set up and manage job wokflows
    - DAGMan meta-scheduler manages these jobs; part of Condor
  - DAGMan
    - Directed Acyclic Graph Manager
    - Allows you to specify *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
  - DAG
    - Data structure used by DAGMan to represent dependencies
      - Children may not run until their parents have finished



    - Each job is a *node* in the DAG
    - Each node can have any number of *parent* or *child* nodes, as long as there are no loops
      - Loops may lead to deadlocks
  - Defining a DAG

- Defined by a `.dag` file, listing each of its nodes and their dependencies

```
# diamond.dag
Job A a.sub      # Condor submit file
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
- DAGMan will output the log file for each job and use it to determine what to do next

– Submitting a DAG

- Run the `condor_submit_dag` command with your `.dag` file
- Condor starts a personal DAGMan daemon to begin running your jobs

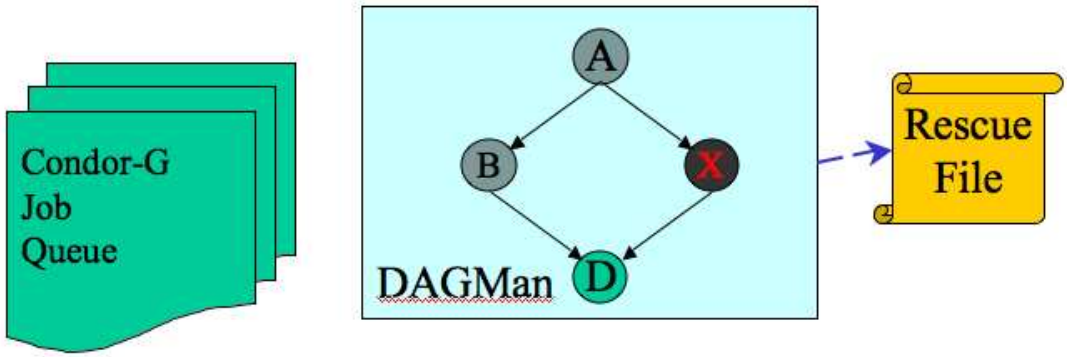
```
% condr_submit_dag diamond.dag
```

- `condor_submit_dag` submits a Scheduler Universe job with DAGMan as the executable
- DAGMan daemon itself runs as a Condor job, so you do not have to baby-sit it
- Just like any other Condor job, you get fault tolerance in case the machine crashes, or reboots, or if there is a network outage
- You are notified when it is done, and whether it succeded or failed

```
% condor_q
-- Submitter: foo.bar.edu : <128.105.175.133:1027> : foo.bar.edu
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 2456.0   user            3/8  19:22   0+00:00:02 R  0   3.2 condor_dagman -f -
```

– Running a DAG

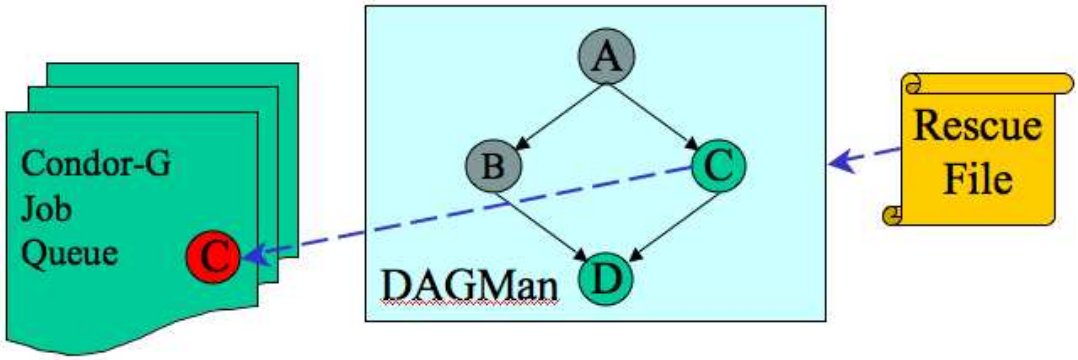
- DAGMan acts as a meta-scheduler
- Manages submission of jobs to Condor-G based on DAG dependencies
- DAGMan holds and submits jobs to the Condor-G queue at appropriate times
  - Once job A completes successfully, jobs B and C will be submitted at the same time
- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a *rescue file* with the current state of the DAG



  - If job C fails, DAGman will wait until job B completes, and then will exit creating a rescue file
  - Job D will not run
  - In its log, DAGman will provide additional details of which node failed and why

– Recovering a DAG – fault tolerance

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



  - Since A and B have already completed, DAGman will start by resubmitting C
  - On completion of C, DAGman will continue as if the failure never happened
- Finishing a DAG
  - Once the DAG is complete, the DAGman job itself is finished and exits

- Working with DAGs – A simple DAG
  - DAGman helps us run several grid job at once
  - Create a small shell script to monitor the Condor-G queue

```
#!/bin/sh
while :
do
    condor_q ${USER}
    condor_q -globus ${USER}
    sleep 10
done
```

  - Create a minimal DAG for DAGman, with a single node
  - The file `mydag.dag` contains only one line as follows

```
Job HelloWorld myjob.submit
```

  - Submit the DAG

```
[sanjiv@osg-grid1 test4]$ condor_submit_dag mydag.dag

Checking all your submit files for log file names.
This might take a while...
Done.
------------------------------------------------------------------------
File for submitting this DAG to Condor           : mydag.dag.condor.sub
Log of DAGMan debugging messages                 : mydag.dag.dagman.out
Log of Condor library output                     : mydag.dag.lib.out
Log of Condor library error messages             : mydag.dag.lib.err
Log of the life of condor_dagman itself          : mydag.dag.dagman.log

Condor Log file for all jobs of this DAG         : /home/sanjiv/test4/results.log
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 512.
------------------------------------------------------------------------
[sanjiv@osg-grid1 test4]$
```

  - Monitor the job using the watch script and using `tail -f` on the file `mydag.dag.dagman.out`

  - DAGman itself runs as a Condor job, specifically a scheduler universe job

- Running a job with more complex DAGs
  - Typically, each node in a DAG will have its own Condor submit file
  - Create more submit files by copying the existing file
  - Copy `myjob.submit` to `job.setup.submit`, `job.work1.submit`, `job.work2.submit`, `job.workfinal.submi` and `job.finalize.submit`
  - Edit the different submit files by changing `results` to `results.NODE` where `NODE` is the middle extension of filename; do not change the `log` entry
    * This avoids different nodes from overwriting each other's output
    * DAGman requires that all nodes output their logs in the same location; Condor will ensure that different jobs will not overwrite each other's entries in the log
  - Change the arguments in submit files
  - Add new nodes to your DAG; new `mydag.dag` is:

```
Job Setup             job.setup.submit
Job WorkerNode_1      job.work1.submit
Job WorkerNode_2      job.work2.submit
Job CollectResults    job.workfinal.submit
Job LastNode          job.finalize.submit
PARENT SetUp          CHILD WorkerNode_1 WorkerNode_2
PARENT WorkerNode_1 WorkerNode_2 CHILD CollectResults
PARENT CollectResults CHILD LastNode
```

  - Change the watch script

```
#!/bin/sh

# wcq: Watch condor queue

while :
do
    echo ....
    echo .... Output from condor_q
    echo ....

    condor_q ${USER}

    echo ....
    echo .... Output from condor_q -globus
    echo ....

    condor_q -globus ${USER}

    echo ....
    echo .... Output from condor_q -dag
    echo ....

    condor_q -dag ${USER}

    sleep 10
done
```

  - Submit your new DAG using `condor_submit_dag` and monitor it

  - Examine your results and log
  - Examine the DAGman log

- Multiple Globus schedulers
  - Schedule multiple dependency jobs to different grid sites
  - Easily done by changing the grid resource in submit files
  - A single DAG can send jobs to a variety of sites
  - Condor-G is capable of managing jobs being distributed to many different sites simultaneously

- Handling failed jobs with DAGman
  - DAGman can handle a situation where some of the nodes in the DAG fail
  - DAGman will run as many nodes as possible and then, create a rescue DAG making it easy when the problem is fixed
  - Set up a script (`dagman.sh`) for failure as follows

```
#!/bin/bash

echo "I am process id $$ on " ‘hostname‘
echo "This is sent to stderr" 1>&2

date

echo "Running as binary $0" "$@"
echo "My name (argument 1) is $1"
echo "My sleep duration (argument 2) is $2"
sleep $2

echo "Sleep of $2 seconds finished.  Exiting."
echo "RESULT: 1 FAILURE"
exit 1
```

  - Modify `job.work2.submit` to include the above job as script and submit the file to run
  - Watch the progress till all jobs are completed
    * DAGman will show that remaining nodes ran based on bad data from the failed node
    * DAGman checks the return code and considers non-zero a failure
    * We are using Condor-G which relies on Globus and Globus does not return error codes
  - Adding a POST script
    * To make DAGman notice the problem and stop the DAG at that point, use a POST script to detect the problem
    * Wrap the executable in a script that will output the executable's return code to stdout and have the POST script scan the stdout for the status, or may be the executable's normal output contains enough information to make the decision
    * In this case, the executable is emitting a well known message; add a POST script
    * Clean up your results; delete the derived files (`mydag.dag.*` and `results.*`); make sure that you do not delete the DAG file `mydag.dag`
    * Create the script to check output: `postscript_checker`

```
#!/bin/bash

grep ’RESULT: 0 SUCCESS’ $1 > /dev/null 2>&1
```

    * Append the following lines to your `mydag.dag`

```
Script POST Setup         postscript_checker results.setup.output
Script POST WorkerNode_1  postscript_checker results.work1.output
Script POST WorkerNode_2  postscript_checker results.work2.output
Script POST CollectResults  postscript_checker results.workfinal.output
Script POST LastNode      postscript_checker results.finalize.output
```

  - Resubmit the DAG
    - Submit the DAG again with the new POST scripts in place using the command `condor_submit_dag`
    - After execution, examine the file `mydag.dag.dagman.out` and check if any of the jobs failed
    - If a job failed, DAGman runs as much of the DAG as possible and logs enough information to continue the run when the situation is resolved.
  - Examine `mydag.dag.rescue`
    - Rescue DAG is structurally the same as original DAG
    - Completed jobs are marked with DONE
    - Upon resubmission, the DONE nodes will be skipped
  - Fix the problem and submit the rescue DAG
    - If you submit rescue DAG without fixing the problem, DAGman will generate another rescue file: `mydag.dag.rescue.r`

- Condor properties
  - Monitors submitted jobs and reports progress
  - Implements your policy on the execution order of jobs
  - Keeps a log of your job activities
- Long jobs
  - What happens to a job when
    - A machine is shut down
    - There is a network outage
    - Another job with higher priority preempts it
  - Do I lose all these hours or days of computation time?
  - What happens when the jobs get preempted?
  - How can I add fault tolerance to my jobs?
- Standard Universe in Condor
  - Condor supports various combinations of features/environments in different *Universes*
  - Different universes provide different functionalities for jobs

    Vanilla runs any serial job

    Scheduler allows a plug in scheduler

    Standard provides support for *transparent process checkpoint* and *restart*
      - Provides two important services to your job: process checkpoint and remote system calls
- Process checkpointing
  - Condor's process checkpointing mechanism saves the entire state of a process (snapshot) into a checkpoint file
    - State includes memory, CPU, and I/O
  - Process can be *restarted* from the point it left off
  - Typically, there is no change to the job's source code; however, the job must be relinked with Condor's Standard Universe support library

OSG and job submissions

- OSG sites present interfaces allowing remotely submitted jobs to be accepted, queued, and executed locally
- OSG supports Condor-G job submission client which interfaces to either the pre-web service or web services GRAM Globus interface at the executing site
- Job managers at the back end of the GRAM gatekeeper support job execution by local Condor, LSF, PBS, or SGE batch systems