Getting Started with OSG Connect

~ an Interactive Tutorial ~

Emelie Harstad <eharstad@unl.edu>, Mats Rynge <rynge@isi.edu>,
Lincoln Bryant <lincolnb@hep.uchicago.edu>, Suchandra Thapa <sthapa@ci.uchicago.edu>,
Balamurugan Desinghu <balamurugan@uchicago.edu>, David Champion <dgc@uchicago.edu>,
Chander S Sehgal <cssehgal@fnal.gov>, Rob Gardner <rwg@hep.uchicago.edu>,
<connect-support@opensciencegrid.org>

Topics

- Properties of DHTC/OSG Jobs
- Getting Started with OSG Connect – Accounts/Logging In/Joining Projects
- Introduction to HTCondor
  - ✧ Exercise: Submit a Simple Job
- Distributed Environment Modules
  - ✧ Exercise: Submit a Batch of R Jobs
- Job Failure Recovery (with short exercise)
- Handling Data: Stash
  - ✧ Exercise: Transfer Data with Globus
  - ✧ Exercise: Access Stash from Job with http
- Workflows with DAGMan
  - ✧ Exercise: DAG NAMD Workflow
- BOSCO – Submit locally, Compute globally
  - ✧ Exercise: Submit Job from Laptop Using BOSCO

Properties of DHTC Jobs

- Run-time: 1-24 hours
- Single-threaded
- Require <2 GB Ram
- Statically compiled executables (transferred with jobs)
- Input and Output files transferred with jobs, and reasonably sized: <10 GB per job (no shared file system on OSG)

Properties of DHTC Jobs

- Run-time: 1-24 hours
- Single-threaded
- Require <2 GB Ram
- Statically compiled executables (transferred with jobs)
- Input and Output files transferred with jobs, and reasonably sized: <10 GB per job (no shared file system on OSG)

These are not hard limits!

- Checkpointing (built-in to application) – for long jobs that are preempted
- Limited support for larger memory jobs
- “Partitionable” slots – for parallel applications using up to 8 cores
- OASIS modules – a collection of pre-installed software packages

Getting Started with OSG Connect

1. Sign up for an account:
   Follow the steps at http://osgconnect.net/signup

1. Add your SSH public key to your account
   a. Sign in at https://portal.osgconnect.net
      (using your campus credentials - InCommon / CILogon)
   b. Managed Identities -> add linked identity -> Add SSH Public Key
   c. Paste contents of ~/.ssh/id_rsa.pub into the text box
      (Help creating a SSH key pair: https://osgconnect.net/keygen)

3. Passwordless login:

```
ssh <username>@login.osgconnect.net
```

4. Join a Project (more info. on next slide)

Projects in OSG Connect

- *Projects* in OSG are used for organizing groups and jobs, granting access to resources, usage accounting.

- Every job submitted through OSG Connect must be associated with a project.

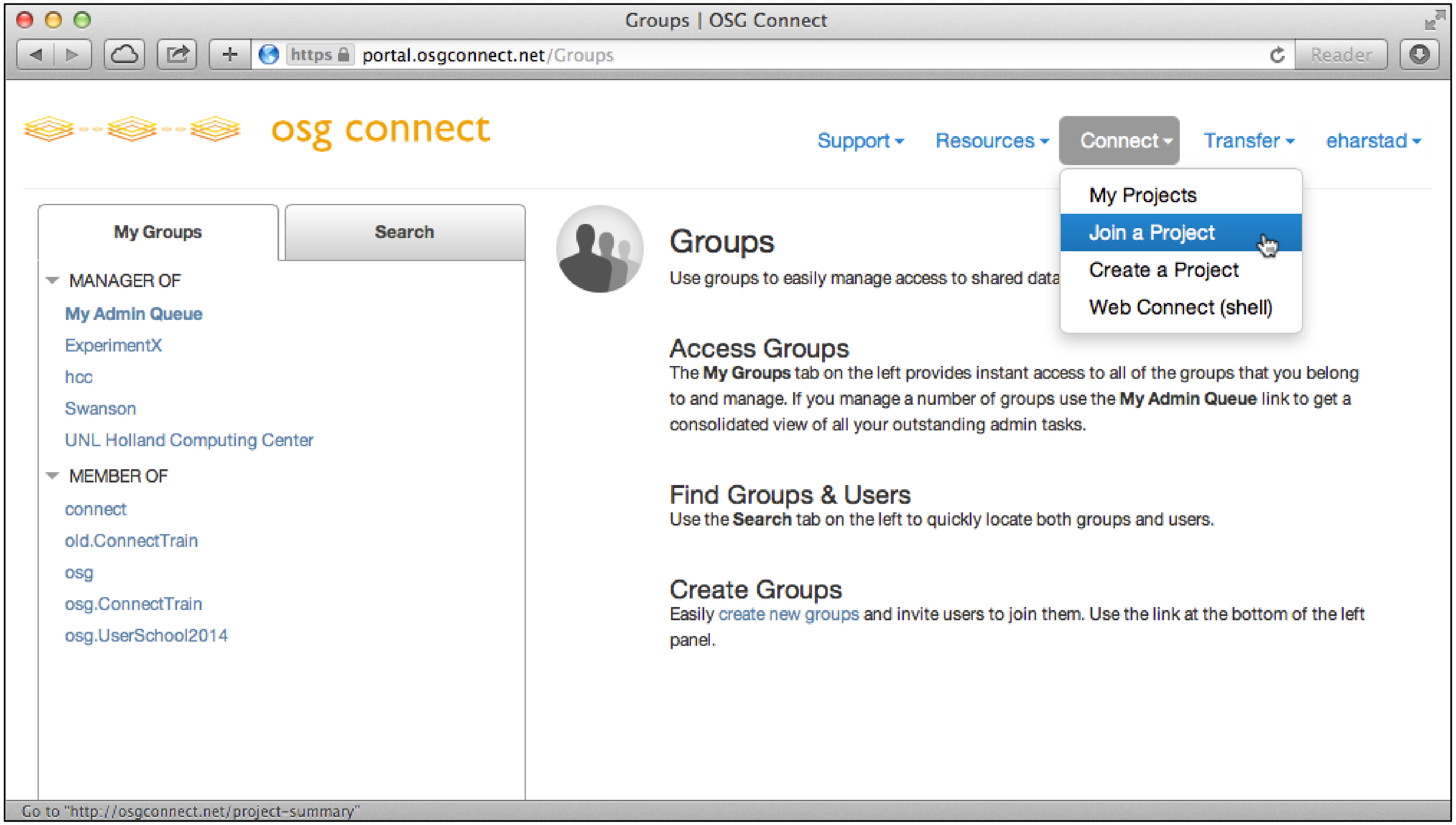
- Principal Investigators or their delegates may create projects and manage project membership.

- To apply for a new project: https://portal.osgconnect.net
  Select: Connect -> Create a Project

- OSG Connect administrator must approve the new project

- To join a pre-existing project: https://portal.osgconnect.net
  Select: Connect -> Join a Project

Projects in OSG Connect


Groups | OSG Connect

https portal.osgconnect.net/Groups

osg connect

MANAGER OF
- My Admin Queue
- ExperimentX
- hcc
- Swanson
- UNL Holland Computing Center

MEMBER OF
- connect
- old.ConnectTrain
- osg
- osg.ConnectTrain
- osg.UserSchool2014

Groups

Use groups to easily manage access to shared data

Access Groups

The **My Groups** tab on the left provides instant access to all of the groups that you belong to and manage. If you manage a number of groups use the **My Admin Queue** link to get a consolidated view of all your outstanding admin tasks.

Find Groups & Users

Use the **Search** tab on the left to quickly locate both groups and users.

Create Groups

Easily create new groups and invite users to join them. Use the link at the bottom of the left panel.

Go to "http://osgconnect.net/project-summary"

Projects in OSG Connect

How to select your project name when submitting a job on OSG Connect

OSG Connect Project Management Commands	
cat .project	# list user's projects (contents of ~/.project file), and show current project.
connect show-projects	# list user's projects
connect project	# allows user to change current project

How to Use the Tutorials

The OSG Connect login node provides a built-in *tutorial* command that provides users with tutorials for many tools and software packages

Commands:

```
$ tutorial #     will print a list tutorials and a brief
                        description for each.

$ tutorial <name>  # will load a specific tutorial.
                              Creates a directory in your
                              current location containing all
                              the files necessary to run the
                              tutorial.
```

Intro to HTCondor

- HTCondor is the OSG Job Scheduler
- Provides an *overlay*: Collection of compute nodes at different OSG sites appears as a single resource to users
- Simplifies job submission: only one submission necessary to access nation-wide pool of resources
- Made possible by *flocking*

Basic procedure:

1) Move all job files to the submit node (or create files directly on the node)
2) Log in to the submit node (ssh <username>@login.osgconnect.net)
3) Create a Condor submit script (contains information for the job scheduler)
4) Submit the job using the 'condor_submit' command.

Intro to HTCondor

Anatomy of a simple condor submit script:

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Intro to HTCondor

Anatomy of a simple condor submit script:

Instructs Condor to use 'vanilla' execution environment

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Intro to HTCondor

Anatomy of a simple condor submit script:

Instructs Condor to use ‘vanilla’ execution environment

Name of the executable file (will automatically be transferred with the job)

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Intro to HTCondor

Anatomy of a simple condor submit script:

```
file: tutorial03.submit
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Instructs Condor to use ‘vanilla’ execution environment

Name of the executable file (will automatically be transferred with the job)

List of input arguments to executable (short.sh)

Intro to HTCondor

Anatomy of a simple condor submit script:

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Instructs Condor to use ‘vanilla’ execution environment

Name of the executable file (will automatically be transferred with the job)

List of input arguments to executable (short.sh)

Error, Output, Log files are created on execute node and transferred back to login node when job finishes.
If subdirectory (e.g. log/) is specified, it must exist.

Intro to HTCondor

Anatomy of a simple condor submit script:

```
file: tutorial03.submit
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Instructs Condor to use 'vanilla' execution environment

Name of the executable file (will automatically be transferred with the job)

List of input arguments to executable (short.sh)

Error, Output, Log files are created on execute node and transferred back to login node when job finishes.
If subdirectory (e.g. log/) is specified, it must exist.

Specify the project name (used for accounting). This line is no longer required. Users can set the project name before submitting the job with the 'connect project' command.

Intro to HTCondor

Anatomy of a simple condor submit script:

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

Instructs Condor to use ‘vanilla’ execution environment

Name of the executable file (will automatically be transferred with the job)

List of input arguments to executable (short.sh)

Error, Output, Log files are created on execute node and transferred back to login node when job finishes.
If subdirectory (e.g. log/) is specified, it must exist.

Specify the project name (used for accounting). This line is no longer required. Users can set the project name before submitting the job with the ‘connect project’ command.

Start 100 identical jobs

Intro to HTCondor

Anatomy of a simple condor submit script:

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh

Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Arguments = 5
Queue 50

Arguments = 10
Queue 20

Arguments = 20
Queue 30
```

Start 50 jobs that sleep for 5 seconds, 20 jobs that sleep for 10 seconds, and 30 jobs that sleep for 20 seconds.

Intro to HTCondor

Anatomy of a simple condor submit script:

file: tutorial03.submit

```
Universe = vanilla
Executable = short.sh
Arguments = 5 # to sleep 5 seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
+ProjectName="ConnectTrain"
Queue 100
```

file: short.sh automatically transferred to execute node

```
#!/bin/bash
# short.sh: a short discovery job

printf "Start time: "; /bin/date
printf "Job is running on node: "; /bin/hostname
printf "Job running as user: "; /usr/bin/id
printf "Job is running in directory: "; /bin/pwd

echo
echo "Working hard..."
sleep ${1-15}
echo "Science complete!"
```

Exercise: Submit a Simple Job

https://confluence.grid.iu.edu/display/CON/OSG+Connect+Quickstart

```
$ ssh username@login.osgconnect.net
$ tutorial quickstart
$ cd ~/tutorial-quickstart
$ nano short.sh
$ chmod +x short.sh
$ ./short.sh
$ nano tutorial03.submit              #Can also use vi/vim
$ condor_submit tutorial03.submit
$ condor_q <username>
$ watch -n2 condor_q <username>  #Ctrl-c to exit
$ condor_history <jobID>
$ condor_history -long <jobID>
$ condor_history -format ‘%s\n’ LastRemoteHost <jobID>
```

Intro to HTCondor

Summary of Useful Condor Commands	
condor_submit <filename.submit>	# Submit job(s) using specified condor submit script
condor_q <username>	# List status of all uncompleted jobs submitted by user
condor_rm <username>	# Remove all jobs submitted by user
condor_rm <jobID>	# Remove job <jobID>
condor_history <jobID> condor_history -long <jobID>	# Provide detailed information about running jobs
condor_ssh_to_job <jobID>	# ssh to the node where specified job is running (useful for debugging)

HTCondor Manual: http://research.cs.wisc.edu/htcondor/manual/

Distributed Environment Modules

- Modules give users access to a collection of software, libraries, and compilers at OSG compute sites.
- Provides consistent environment across sites for testing and running workflows.
- Modules are published via the distributed file system OASIS, which is available on most sites running OSG Connect jobs.
- Usage: `module load python/2.7`
- More information and list of available modules: https://confluence.grid.iu.edu/display/CON/Distributed+Environment+Modules

Distributed Environment Modules

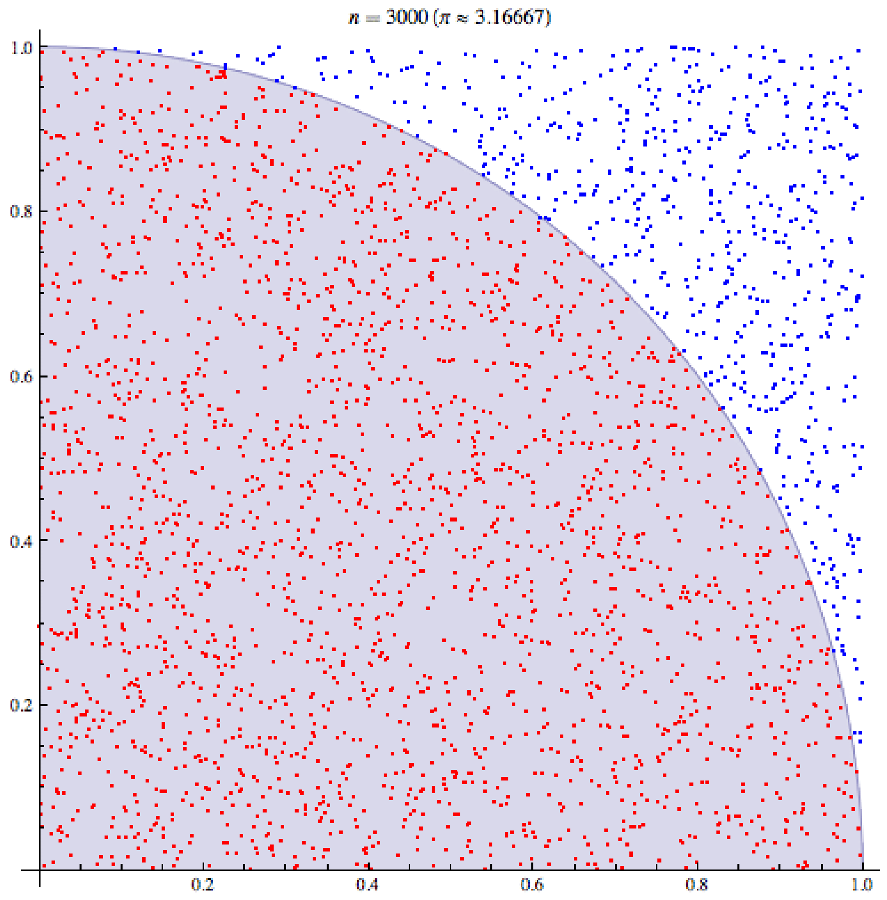
Useful Module Commands	
module avail	# List all available modules/versions
module load <module_name>	# Load a module (sets up environment for using software or libraries)
module unload <module_name>	# Unload a module
module list	# List all loaded modules
module spider <module_name>	# List module dependencies
module keyword <key1> <key2> …	# List modules matching any of the specified keywords

✧ DON'T FORGET!!!! To use modules, first issue the following command:

```
source /cvmfs/oasis.opensciencegrid.org/osg/modules/lmod/5.6.2/init/bash
```

Distributed Environment Modules

Exercise: A Monte Carlo method for estimating the value of Pi.



Take a random sampling of n points on the square inscribed by a unit circle.

Ratio of number of points inside the circle to the total number of trials, n, approaches pi/4 as n increases.

The key is to have a large number of samples, n.

Break the problem down into smaller jobs (smaller n), and take the average of the results.

(Source: http://en.wikipedia.org/wiki/Monte_Carlo_method)

Distributed Environment Modules

Submit Script:

file: R.submit

```
universe = vanilla

Executable = R-wrapper.sh
arguments = mcpi.R
transfer_input_files = mcpi.R

output = job.out.$(Process)
error = job.error.$(Process)
log = job.log.$(Process)

requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org \
        =?= TRUE)
queue 100
```

Distributed Environment Modules

Submit Script:

```
file: R.submit
universe = vanilla

Executable = R-wrapper.sh
arguments = mcpi.R
transfer_input_files = mcpi.R

output = job.out.$(Process)
error = job.error.$(Process)
log = job.log.$(Process)

requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org \
        =?= TRUE)
queue 100
```

Wrapper Script: sets up the environment, loads R module, and invokes R script.

```
file: R-wrapper.sh
#!/bin/bash
source /cvmfs/oasis.open \
        sciencegrid.org/osg/modules/ \
        lmod/5.6.2/init/bash
module load R
Rscript $1
```

Distributed Environment Modules

Submit Script:

```
file: R.submit
universe = vanilla

Executable = R-wrapper.sh
arguments = mcpi.R
transfer_input_files = mcpi.R

output = job.out.$(Process)
error = job.error.$(Process)
log = job.log.$(Process)

requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org \
        =?= TRUE)
queue 100
```

Wrapper Script: sets up the environment, loads R module, and invokes R script.

```
file: R-wrapper.sh
#!/bin/bash
source /cvmfs/oasis.open \
        sciencegrid.org/osg/modules/ \
        lmod/5.6.2/init/bash
module load R
Rscript $1
```

```
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
```

Distributed Environment Modules

Submit Script:

```
file: R.submit
universe = vanilla

Executable = R-wrapper.sh
arguments = mcpi.R
transfer_input_files = mcpi.R

output = job.out.$(Process)
error = job.error.$(Process)
log = job.log.$(Process)

requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org \
        =?= TRUE)
queue 100
```

Wrapper Script: sets up the environment, loads R module, and invokes R script.

```
file: R-wrapper.sh
#!/bin/bash
source /cvmfs/oasis.open \
        sciencegrid.org/osg/modules/ \
        lmod/5.6.2/init/bash
module load R
Rscript $1
```

```
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
```

```
source /cvmfs/oasis.opensciencegrid.org/osg/modules/lmod/5.6.2/init/bash
```

Distributed Environment Modules

Submit Script:

```
file: R.submit
universe = vanilla

Executable = R-wrapper.sh
arguments = mcpi.R
transfer_input_files = mcpi.R

output = job.out.$(Process)
error = job.error.$(Process)
log = job.log.$(Process)

requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org \
         =?= TRUE)
queue 100
```

Wrapper Script: sets up the environment, loads R module, and invokes R script.

```
file: R-wrapper.sh
#!/bin/bash
source /cvmfs/oasis.open \
         sciencegrid.org/osg/modules/ \
         lmod/5.6.2/init/bash
module load R
Rscript $1
```

Prepares environment for running R

Equivalent to: 'R --slave'
(accepts R script as argument)

Distributed Environment Modules

Submit Script:

```
file: R.submit
universe = vanilla

Executable = R-wrapper.sh
arguments = mcpi.R
transfer_input_files = mcpi.R

output = job.out.$(Process)
error = job.error.$(Process)
log = job.log.$(Process)

requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org \
         =?= TRUE)
queue 100
```

R Script: performs the actual analysis

```
file: mcpi.R
montecarloPi <- function(trials) {
  count = 0
  for(i in 1:trials) {
    if((runif(1,0,1)^2 + \
        runif(1,0,1)^2)<1) {
      count = count + 1
    }
  }
  return((count*4)/trials)
}

montecarloPi(1000)
```

✧ mcpi.R is not the executable for this Condor job (the wrapper script is the executable). So mcpi.R must be specified on the 'transfer_input_files' line, along with any other input files.

Exercise: Submit a Batch of R Jobs

```
$ ssh <username>@login.osgconnect.net
$ tutorial R
$ cd ~/tutorial-R
$ nano mcpi.R
$ nano R-wrapper.sh
$ nano R.submit
$ ./R-wrapper.sh mcpi.R
$ condor_submit R.submit
$ condor_q <username>
$ watch -n2 condor_q <username>  #Ctrl-c to exit
$ condor_history <cluster>
$ condor_history  -long <cluster>
$   grep "\[1\]" job.out.* | awk '{sum += $2} END { print "Average =", sum/NR}'
```

Exercise: Troubleshooting Job Failure

Submit Script:

file: error101_job.submit

```
Universe = vanilla

Executable = error101.sh
Arguments = 3600 # to sleep an hour

Requirements = (Memory >= 51200)

Error = job.err
Output = job.out
Log = job.log
Queue 1
```

file: error101.sh automatically transferred to execute node

```
#!/bin/bash
# error101.sh: a short discovery job

printf "Start time: "; /bin/date
printf "Job is running on node: "; /bin/hostname
printf "Job running as user: "; /usr/bin/id
printf "Job is running in directory: "; /bin/pwd

echo
echo " Working hard... "
sleep ${1-15}
echo " Science complete! "
```

Exercise: Troubleshooting Job Failure

Submit Script:

file: error101_job.submit
```
Universe = vanilla

Executable = error101.sh
Arguments = 3600 # to sleep an hour

Requirements = (Memory >= 51200)

Error = job.err
Output = job.out
Log = job.log
Queue 1
```

Note the additional requirement for 51200 MB of memory!

file: error101.sh automatically transferred to execute node
```
#!/bin/bash
# error101.sh: a short discovery job

printf "Start time: "; /bin/date
printf "Job is running on node: "; /bin/hostname
printf "Job running as user: "; /usr/bin/id
printf "Job is running in directory: "; /bin/pwd

echo
echo " Working hard... "
sleep ${1-15}
echo " Science complete! "
```

Exercise: Troubleshooting Job Failure

```
$ ssh username@login.osgconnect.net
$ tutorial error101
$ nano error101_job.submit
$ nano error101.sh
$ condor_submit error101_job.submit
$ condor_q <username>
$ condor_q -analyze <jobID>
$ condor_q -better-analyze <jobID>

$ condor_qedit <jobID> Requirements ‘Memory >= 512’

OR

$ condor_rm <jobID>                          # Cancel the job
$ nano error101_job.submit                   # Edit the submit file
$ condor_submit error101_job.submit     # Re-submit job
```

Troubleshooting Job Failure

Condor Commands for Troubleshooting	
condor_q -analyze <jobID>	# Print detailed information about job status
condor_q -better-analyze <jobID>	# Print (longer) detailed information about job status
condor_qedit <jobID> \ <attribute_name> <attribute_value>	# Edit attributes of job in idle state
condor_release <jobID>	# Release job from 'held' state
condor_ssh_to_job <jobID>	# ssh to the node where specified job is running (useful for debugging)

✧ Also, don't forget to check the job log and error files!!

Handling Data - Stash

Stash

- Distributed filesystem for staging data for OSG Connect jobs

- Temporary storage of job I/O files

- Accessible on OSG Connect login node

  Your stash directory is: ~/data

  Can use scp/sftp to transfer to and from stash:

  ```
  scp input_data.tar.gz username@login.osgconnect.net:~/data/.
  scp username@login.osgconnect.net:~/data/outputdata.tar.gz ./
  ```

- Accessible through Globus Connect (or the OSG Connect Web Portal: https://portal.osgconnect.net)

- Publically available on the web

  Data located in ~/data/public can be accessed online at:

  http://stash.osgconnect.net/+username

  Access stash from a compute node:

  ```
  wget http://stash.osgconnect.net/+username/input.dat
  ```

Handling Data - Stash

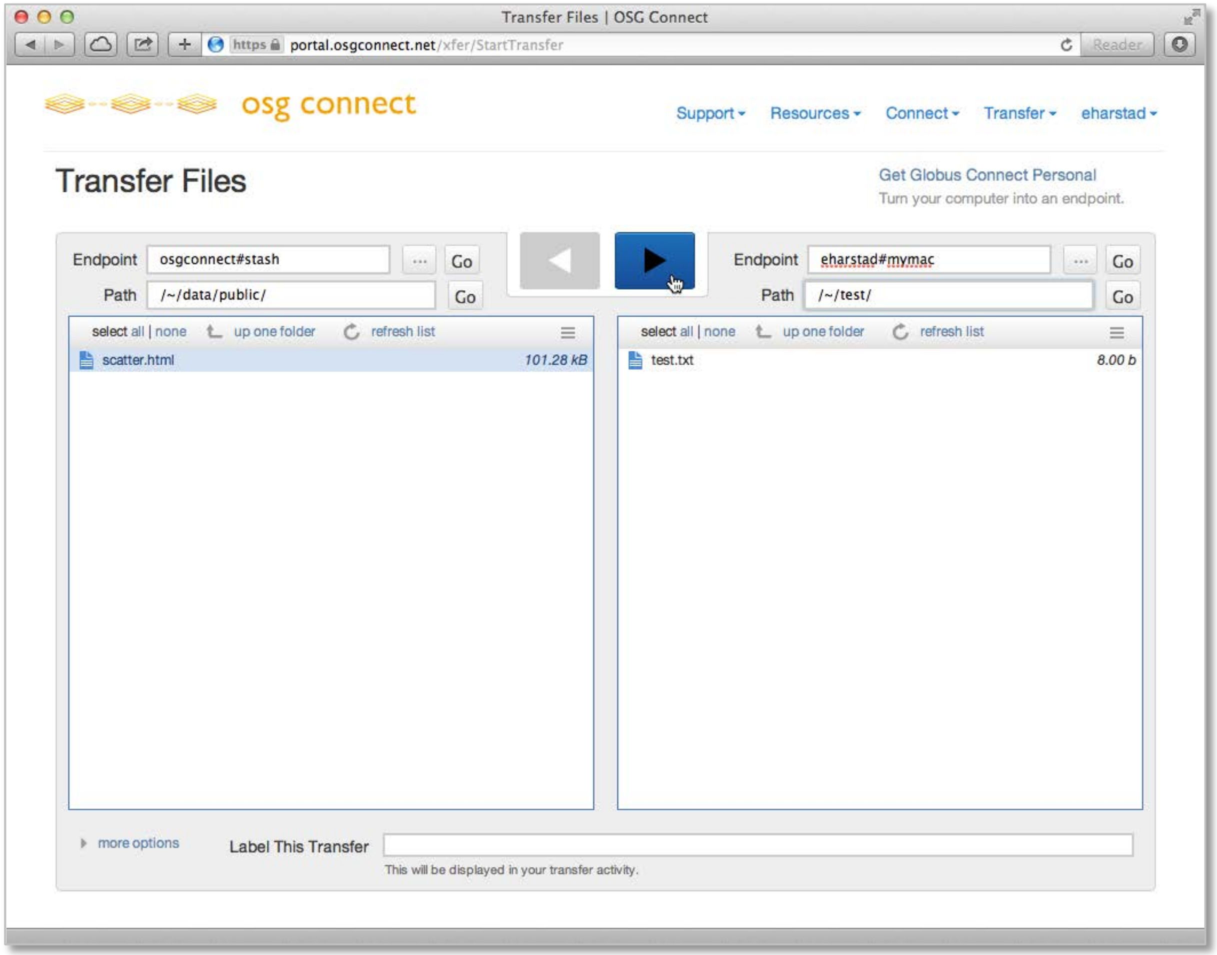
Accessing Stash through Globus

1) Login at
http://portal.osgconnect.net

2) Select:
Transfer -> Start Transfer

3) Enter endpoint names and navigate to your file(s)
The stash endpoint is "osgconnect#stash"

4) "Get Globus Connect Personal" to use your own computer as an endpoint.


Transfer Files | OSG Connect

https portal.osgconnect.net/xfer/StartTransfer

osg connect

Support Resources Connect Transfer eharstad

Transfer Files

Get Globus Connect Personal
Turn your computer into an endpoint.

Endpoint osgconnect#stash Go
Path /~/data/public/ Go

select all | none up one folder refresh list

scatter.html 101.28 kB

Endpoint eharstad#mymac Go
Path /~/test/ Go

select all | none up one folder refresh list

test.txt 8.00 b

more options

Label This Transfer

This will be displayed in your transfer activity.

Handling Data - Stash

Submit Script:

file: namd_stash_run.submit

```
Universe = vanilla
Executable = namd_stash_run.sh

transfer_input_files = ubq_gbis_eq.conf, ubq.pdb, ubq.psf
should_transfer_files=Yes
Transfer_Output_Files = namdoutput_using_stash.dat
when_to_transfer_output = ON_EXIT
output        = job.out
error         = job.error
log           = job.log
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
Queue 1
```

Handling Data - Stash

Submit Script:

```
file: namd_stash_run.submit
Universe = vanilla
Executable = namd_stash_run.sh

transfer_input_files = ubq_gbis_eq.conf, ubq.pdb, ubq.psf
should_transfer_files=Yes
Transfer_Output_Files = namdoutput_using_stash.dat
when_to_transfer_output = ON_EXIT
output        = job.out
error         = job.error
log           = job.log
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
Queue 1
```

List of input files to transfer

Specify:

- whether to transfer output files
- name of output file(s)
- when to transfer

Handling Data - Stash

Submit Script:

file: namd_stash_run.submit

```
Universe = vanilla
Executable = namd_stash_run.sh

transfer_input_files = ubq_gbis_eq.conf, ubq.pdb, ubq.psf
should_transfer_files=Yes
Transfer_Output_Files = namdoutput_using_stash.dat
when_to_transfer_output = ON_EXIT
output       = job.out
error        = job.error
log          = job.log
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
Queue 1
```

Executable: Prepares environment, launches namd with specified config file

file: namd_stash_run.sh

```
#!/bin/bash
source /cvmfs/oasis.opensciencegrid.org/osg/modules/lmod/5.6.2/init/bash
module load namd/2.9
wget http://stash.osgconnect.net/+username/Namd_param/par_all27_prot_lipid.inp
namd2 ubq_gbis_eq.conf > namdoutput_using_stash.dat
```

Handling Data - Stash

Submit Script:

```
file: namd_stash_run.submit
Universe = vanilla
Executable = namd_stash_run.sh

transfer_input_files = ubq_gbis_eq.conf, ubq.pdb, ubq.psf
should_transfer_files=Yes
Transfer_Output_Files = namdoutput_using_stash.dat
when_to_transfer_output = ON_EXIT
output       = job.out
error        = job.error
log          = job.log
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
Queue 1
```

Download input data from stash

Executable: Prepares environment, launches namd with specified config file

```
file: namd_stash_run.sh
#!/bin/bash
source /cvmfs/oasis.opensciencegrid.org/osg/modules/lmod/5.6.2/init/bash
module load namd/2.9
wget http://stash.osgconnect.net/+username/Namd_param/par_all27_prot_lipid.inp
namd2 ubq_gbis_eq.conf > namdoutput_using_stash.dat
```

Handling Data - Stash

Submit Script:

```
file: namd_stash_run.submit
Universe = vanilla
Executable = namd_stash_run.sh

transfer_input_files = ubq_gbis_eq.conf, ubq.pdb, ubq.psf
should_transfer_files=Yes
Transfer_Output_Files = namdoutput_using_stash.dat
when_to_transfer_output = ON_EXIT
output        = job.out
error         = job.error
log           = job.log
requirements = (HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE)
Queue 1
```

Redirect namd output to a file.

Executable: Prepares environment, launches namd with specified config file

```
file: namd_stash_run.sh
#!/bin/bash
source /cvmfs/oasis.opensciencegrid.org/osg/modules/lmod/5.6.2/init/bash
module load namd/2.9
wget http://stash.osgconnect.net/+username/Namd_param/par_all27_prot_lipid.inp
namd2 ubq_gbis_eq.conf > namdoutput_using_stash.dat
```

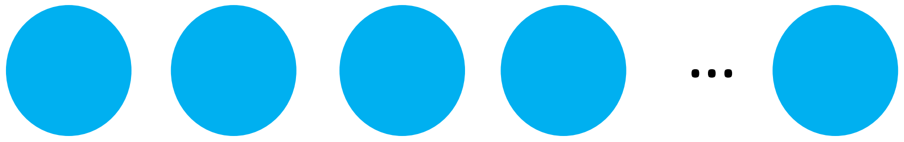
Exercise: Access Stash from Job with http

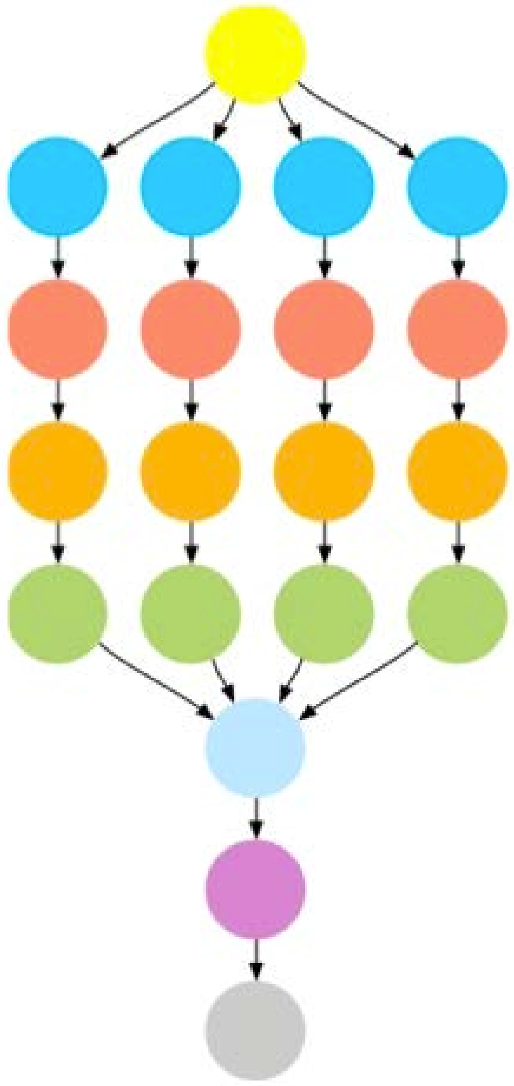
```
$ ssh <username>@login.osgconnect.net
$ tutorial stash-namd
$ cd ~/tutorial-stash-namd
$ nano namd_stash_run.submit
$ nano namd_stash_run.sh                    # Edit “username”
$ cp par_all27_prot_lipid.inp ~/data/public/.
$ ./namd_stash_run.sh
$ condor_submit namd_stash_run.submit
$ condor_q <username>
$ watch -n2 condor_q <username>  #Ctrl-c to exit
$ condor_q -analyze <jobID>
```

Job Workflows with DAG

DAGMan is recommended for all production style workloads, even if there is no structure to your jobs

- Good job retry mechanism (try jobs N times, check success with post scripts, ..)
- Can throttle the number of submitted jobs
- Provides a workload "checkpointing" mechanism

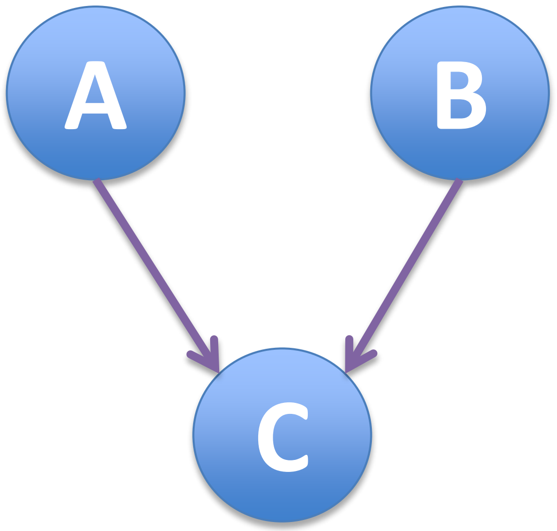

Independent jobs



Job Workflows with DAG

DAG file points to regular HTCondor job submit files, and allows you to specify relationships

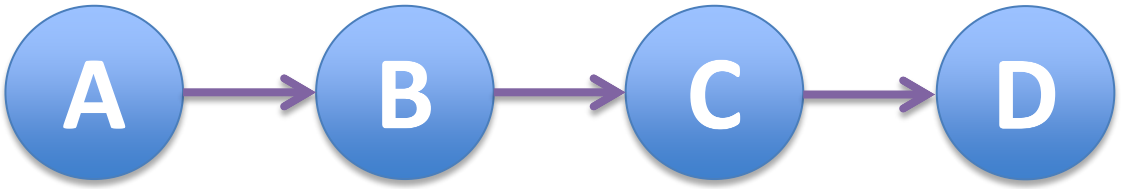
```
JOB A job_a.submit
RETRY A 3

JOB B job_b.submit
RETRY B 3

JOB C job_c.submit
RETRY C 3

PARENT A CHILD C
PARENT B CHILD C
```


$$\begin{tikzcd} A \arrow[dr] & & B \arrow[dl] \\ & C & \end{tikzcd}$$

Job Workflows with DAG

Today's Exercise: Simple Linear DAG

- Each step depends on successful completion of previous step.
- For relatively short jobs, monitoring this without a DAG is tedious and inefficient.

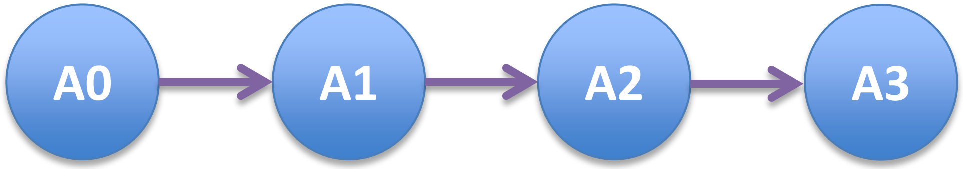


Job Workflows with DAG

DAG file:

file: linear.dag

```
#######DAG file#######
Job A0 namd_run_job0.submit
Job A1 namd_run_job1.submit
Job A2 namd_run_job2.submit
Job A3 namd_run_job3.submit
PARENT A0 CHILD A1
PARENT A1 CHILD A2
PARENT A2 CHILD A3
```

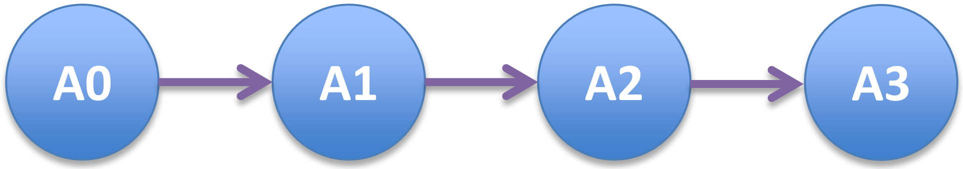

$$\begin{tikzcd} A0 \arrow[r] & A1 \arrow[r] & A2 \arrow[r] & A3 \end{tikzcd}$$

Job Workflows with DAG

DAG file:

```
file: linear.dag
######DAG file#######
Job A0 namd_run_job0.submit
Job A1 namd_run_job1.submit
Job A2 namd_run_job2.submit
Job A3 namd_run_job3.submit
PARENT A0 CHILD A1
PARENT A1 CHILD A2
PARENT A2 CHILD A3
```

Job keyword, Job Name, Condor Job submission script


$$\begin{tikzcd} A0 \arrow[r] & A1 \arrow[r] & A2 \arrow[r] & A3 \end{tikzcd}$$

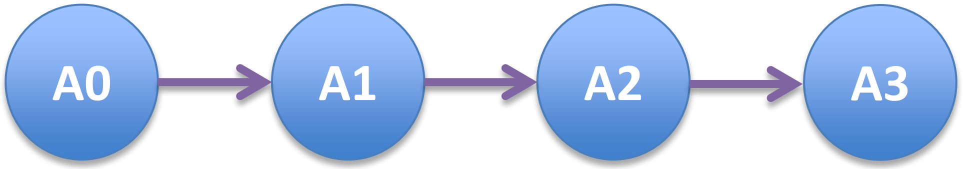
Job Workflows with DAG

DAG file:

```
file: linear.dag
######DAG file######
Job A0 namd_run_job0.submit
Job A1 namd_run_job1.submit
Job A2 namd_run_job2.submit
Job A3 namd_run_job3.submit
PARENT A0 CHILD A1
PARENT A1 CHILD A2
PARENT A2 CHILD A3
```

Job keyword, Job Name, Condor Job submission script

Job dependency description


$$\begin{tikzcd} A0 \arrow[r] & A1 \arrow[r] & A2 \arrow[r] & A3 \end{tikzcd}$$

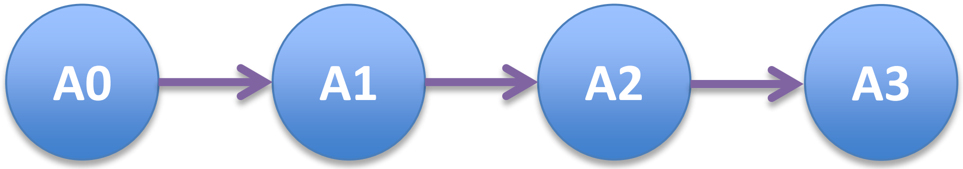
Job Workflows with DAG

DAG file:

```
file: linear.dag
######DAG file######
Job A0 namd_run_job0.submit
Job A1 namd_run_job1.submit
Job A2 namd_run_job2.submit
Job A3 namd_run_job3.submit
PARENT A0 CHILD A1
PARENT A1 CHILD A2
PARENT A2 CHILD A3
```

Submit file:

```
file: namd_run_job1.submit
Universe = vanilla
Executable = namd_run_job1.sh
transfer_input_files = ubq_gbis_eq_job1.conf, \
          ubq.pdb, ubq.psf, \
          par_all27_prot_lipid.inp, \
          OutFilesFromNAMD_job0.tar.gz
should_transfer_files=Yes
when_to_transfer_output = ON_EXIT
output        = job.output.job1
error         = job.error.job1
log           = job.log.job1
requirements = \
(HAS_CVMFS_oasis_opensciencegrid_org =?= TRUE) Queue
1
```



$$\begin{tikzcd} A0 \arrow[r] & A1 \arrow[r] & A2 \arrow[r] & A3 \end{tikzcd}$$

Job Workflows with DAG

DAG file:

```
file: linear.dag
######DAG file######
Job A0 namd_run_job0.submit
Job A1 namd_run_job1.submit
Job A2 namd_run_job2.submit
Job A3 namd_run_job3.submit
PARENT A0 CHILD A1
PARENT A1 CHILD A2
PARENT A2 CHILD A3
```

Submit file:

```
file: namd_run_job1.submit
Universe = vanilla
Executable = namd_run_job1.sh
transfer_input_files = ubq_gbis_eq_job1.conf, \
         ubq.pdb, ubq.psf, \
         par_all27_prot_lipid.inp, \
         OutFilesFromNAMD_job0.tar.gz
should_transfer_files=Yes
when_to_transfer_output = ON_EXIT
output       = job.output.job1
error        = job.error.job1
```

```
file: namd_run_job1.sh
#!/bin/bash
tar xzf OutFilesFromNAMD_job0.tar.gz
mv OutFilesFromNAMD_job0/*job0.restart* .
source /cvmfs/oasis.opensciencegrid.org/osg/modules/lmod/5.6.2/init/bash
module load namd/2.9
namd2 ubq_gbis_eq_job1.conf > ubq_gbis_eq_job1.log
mkdir OutFilesFromNAMD_job1
rm *job0*
cp * OutFilesFromNAMD_job1/.
tar czf OutFilesFromNAMD_job1.tar.gz OutFilesFromNAMD_job1
```

Exerscise: DAG NAMD Workflow

```
$ ssh <username>@login.osgconnect.net
$ tutorial dagman-namd
$ cd ~/tutorial-dagman-namd
$ nano namd_run_job1.submit
$ nano namd_run_job1.sh
$ condor_submit_dag linear.dag
$ watch -n2 condor_q <username>  #Ctrl-c to exit
```

Exerscise: DAG NAMD Workflow

Bonus Exercise: X-DAG

```
$ ssh <username>@login.osgconnect.net
$ tutorial dagman-namd
$ cd ~/tutorial-dagman-namd/X-DAG
$ condor_submit_dag xconfig
```

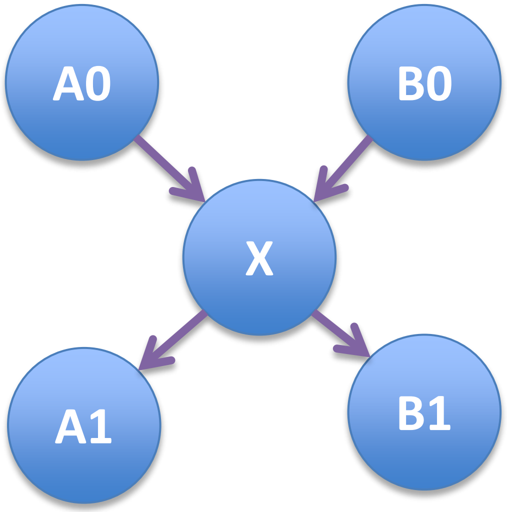
Take a look at the dag file ‘xconfig’, and see if you can draw a picture of the dependency graph.

Exerscise: DAG NAMD Workflow

Bonus Exercise: X-DAG

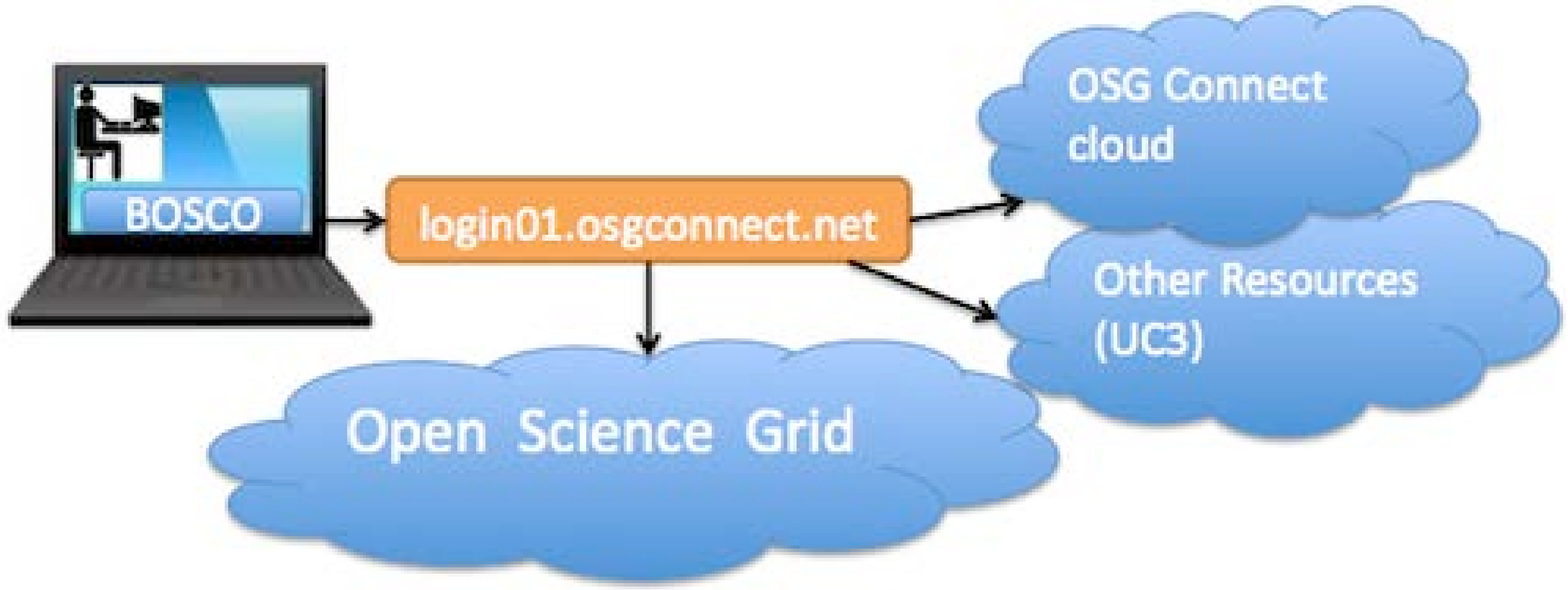
```
$ ssh <username>@login.osgconnect.net
$ tutorial dagman-namd
$ cd ~/tutorial-dagman-namd/X-DAG
$ condor_submit_dag xconfig
```

Take a look at the dag file ‘xconfig’, and see if you can draw a picture of the dependency graph.



$$\begin{tikzcd}
A0 \arrow[dr] & & B0 \arrow[dl] \\
& X \arrow[dl] \arrow[dr] & \\
A1 & & B1
\end{tikzcd}$$

BOSCO – Stage Jobs Locally



https://confluence.grid.iu.edu/pages/viewpage.action?pageId=10944561

BOSCO – Stage Jobs Locally

Download BOSCO to your laptop or workstation: (download: http://bosco.opensciencegrid.org/download/)

```
$ wget -O ./bosco_quickstart.tar.gz \
http://bosco.opensciencegrid.org/download-form/?package=1.2/bosco_quickstart.tar.gz
```

OR

```
$ curl -O \
http://bosco.opensciencegrid.org/download-form/?package=1.2/bosco_quickstart.tar.gz
```

Untar the package and run the quickstart script:

```
$ tar xvzf ./bosco_quickstart.tar.gz
$ ./bosco_quickstart
```

Answer the questions:

- When prompted "Do you want to install Bosco? Select y/n and press [ENTER]:" press "y" and ENTER.
- When prompted "Type the cluster name and press [ENTER]:" type login.osgconnect.net and press ENTER.
- When prompted "Type your name at login.osgconnect.net (default YOUR_USER) and press [ENTER]:" enter your user name on OSG-Connect and press ENTER.
- When prompted "Type the queue manager for login.osgconnect.net (pbs, condor, lsf, sge, slurm) and press [ENTER]:" enter condor and press ENTER.

Remove the installer and its log file:

```
$ rm bosco_quickstart*
```

Exercise: Submit Job from Laptop Using BOSCO

Each time you want to run BOSCO, first set up the environment, then start BOSCO:

```
$ source ~/bosco/bosco_setenv
$ bosco_start
```

Copy the quickstart tutorial from the osgconnect login node to your computer:

```
$ scp -r <username>@login.osgconnect.net:~/tutorial-quickstart ./
$ cd tutorial-quickstart
```

Edit the submit script: Change 'vanilla' to 'grid'

Submit the job:

```
$ condor_submit tutorial03.submit
```

Check the status of your job:

```
$ condor_q
```

- Note that condor_q lists only your jobs even without specifying the user id. There may be other jobs queued on OSG Connect but to see them you'll have to login on login.osgconnect.net and issue condor_q there.