User Jamboree May 2013

Contents

1 Introduction
   1.1 Coordinates
   1.2 Registration
   1.3 Workshop Objectives
2 UC3 Overview
3 UC3 Tutorial
   3.1 Session I: Getting started on UC3
   3.2 Session II: Getting a taste of UC3
   3.3 Session III: Software and Data Access Fundamentals
   3.4 Session IV: Specific Application Examples
   3.5 Session V: Higher level tools: Swift and UC3
4 Summary and Closeout

Introduction

UC3 - the University of Chicago Computing Cooperative - is about connecting users to distributed high throughput computing (DHTC) resources whether those resources are on campus, at a remote lab, on the grid, or in a cloud. If your science application requires access to compute cycles, software and data - more than your local environment can handle - then this workshop might be of interest to you. The challenges for
effectively using DHTC resources for compute and data intensive applications are many: distributed identity management, software distribution, local and remote data access are just a few. UC3 aims to simplify these for the class of workloads that are loosely coupled and therefore well-suited to a DHTC environment. One thinks and works locally, but the computation happens globally. This will be a practical, hands on session to introduce potential users to the UC3 environment, and to demonstrate advanced capabilities for software and data access, and (transparent) job scheduling out to the Open Science Grid.

Coordinates

- When: May 23, 2013
- Time: 9 am to 12:30 pm
- Where: Searle 241a

Registration

- To register for the event: http://uc3ujam2013.eventbrite.com/
- All that is needed is a laptop with an ssh client and a CNetID.

Workshop Objectives

- Register to UC3, log in to an interactive submit host, become familiar with the environment.
- Learn a modicum of HTCondor, enough to handle all the basics of job submission, monitoring, and removal
- Scale up a simple application to reach 100s of empty job slots on campus including the UC3 Seeder Cluster, the ITS AppCloud, the Midwest Tier2 center, the ATLAS Tier 3 center, BSD SIRAF Facility, and the RCC Midway cluster (account and allocation required).
- Submit 1000 jobs to the Open Science Grid, opportunistically, free of charge.
- Learn how to submit jobs to the UC3 environment and Midway simultaneously, from your Midway account.
- Use Globus Online to transfer data into the UC3 processing environment.
- Learn how to access software and data from remote locations (i.e. break the CPU-data locality requirement).
- Run practice codes for R, Matlab/Octave, Mathematica and ROOT on UC3.
- Learn how to use Swift, a higher level scripting language for program execution in the UC3 environment.

UC3 Overview

- A brief presentation describing UC3 concepts and principles, what resources are currently available.
  - uc3-presentation-v2.pdf

UC3 Tutorial

Session I: Getting started on UC3
• Registration and login (interactive)
• Quick job submission tutorial
• Scaling up to more resources
• Job Submission from Midway or Laptop

Session II: Getting a taste of UC3

• Using R on UC3
• Using Mathematica on UC3

Session III: Software and Data Access Fundamentals

• Software access in the UC3 environment
• UC3 storage system
• Accessing data and software from anywhere
• Using Globus Online with UC3 Storage

Session IV: Specific Application Examples

• Using Octave on UC3
• Using ROOT on UC3

Session V: Higher level tools: Swift and UC3

• Swift is a parallel scripting language for large scale parallel program executions.
• We'll show how Swift is used in the UC3 environment to manage access to a number of compute resources.

Summary and Closeout

• Attendee profiles and questionnaire
UC3: A Framework for Cooperative Computing at the University of Chicago

THE UC3 SUPPORT TEAM
Computation and Enrico Fermi Institutes

May 23, 2013

Outline

- UC3 background
- Principles
- Implementation
UC Computing Cooperative

- Research project to **connect** users to distributed high throughput computing resources (DHTC)
- A **framework** for resource providers & users
  - Participation entirely voluntary
- **Successes**: 9 publications directly benefiting from UC3 cycles in 2012 (chemistry, particle astrophysics, economics modeling); contributed to Higgs discovery at CERN

UC3 Principles

- Allow users to think locally, compute globally
  - Access distributed resources (CPU and storage) transparently
- Requires some changes from the traditional model
  - Identity: use a local campus identity when possible
  - Software delivery – cannot rely on installed libraries
  - Data access – data may not be local
  - Resource variability: use of “ClassAds” & requirements
  - Resource access: both allocations vs. “opportunistic”
- TODAY: we’re going to “skim” through things in introductory fashion. Future tutorials will go deeper.
Inspired by so-called “campus grids” in the Open Science Grid (c.f. http://campusgrids.org)

UC3 IMPLEMENTATION

UC3 Connection Infrastructure

- Identity management, automatic account creation
- Login host + HTCondor services
- Open “application seeder” cluster 544 job slots
- BOSCO multi-user service
  - for accessing non-HTCondor schedulers (PBS, SGE, SLURM)
- Dedicated CVMFS repository for UC3 applications
- Data server backed by 52 TB of HDFS storage
  - Globus Online endpoint integrated with CILogon
  - Data access managed by Chirp, Parrot, and Skeleton Key
- Monitoring
- Latest – user/project engagement service (thanks UW CHTC)
Identity, Accounts & sign-up

- uc3.uchicago.edu

Further Reference
- Job Submission Reference

Integrating with User Engagement Portal from UW CHTC

UC3 Connected Resources

UC3 Submit

UC3 Data

NFS Mounts

UC3 CVMFS

HDFS

UC3 BOSCO

OSG FRONT END

OSG GUIDEIN WMS

PBS

ITB

SIRAF

Midway

SLURM

Midwest Tier 2

Condor

UC3 Seeder

Condor

UCT3

Condor

ITS

OSG SITES
UC3 connected resources + BOSCO + OSG
(not shown)

**Pool Summary**

<table>
<thead>
<tr>
<th>Pool</th>
<th>Total Slots</th>
<th>Running</th>
<th>Idle</th>
<th>Owner</th>
<th>Status</th>
<th>Detailed View</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 machine down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>ITS</td>
<td>240</td>
<td>240</td>
<td>0</td>
<td>0</td>
<td>1 machine down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>MWT2 (EU)</td>
<td>2064</td>
<td>1879</td>
<td>181</td>
<td>4</td>
<td>2 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>MWT2 (UC)</td>
<td>3259</td>
<td>3066</td>
<td>193</td>
<td>8</td>
<td>11 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>MWT2 (UIUC)</td>
<td>480</td>
<td>32</td>
<td>448</td>
<td>8</td>
<td>23 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>MWT2 (Validation)</td>
<td>404</td>
<td>334</td>
<td>5</td>
<td>65</td>
<td>52 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>MWT2 (Cloud)</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>OSG VO Frontend</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>UC3 Bosco</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>28 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>UC3 Seeder</td>
<td>528</td>
<td>528</td>
<td>0</td>
<td>0</td>
<td>All machines up</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>UC3 Tutorial</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>All machines up</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td>UCT3 Condor Overlay</td>
<td>322</td>
<td>319</td>
<td>2</td>
<td>1</td>
<td>42 machines down</td>
<td>Usage Jobs Status</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7299</strong></td>
<td><strong>6128</strong></td>
<td><strong>645</strong></td>
<td><strong>526</strong></td>
<td><strong>221 machines down</strong></td>
<td>Usage Jobs Status</td>
</tr>
</tbody>
</table>

- Jobs by State
- Jobs by Owner
- Slots by State
- Slots by Owner

**Usage last week**

Historical grid usage in all pools

*View at: Area |

Legend:
- Unclaimed
- Lincoln-Bryant
- Engage
- GLOW
- UK ATLAS
- HEP
- otherUS
- otherUK
- input
- output
- PIPHI
- RBI-YP
- AutopsyFactory
- Sam-Mahan
- Karol Kliche
- Action Kaply
- Unclaimed
Open Science Grid Resources

Last year:
>750M transfers
>350 PB

Last year:
>720M CPU-hours

Open Science Grid – scale (total)

Last year:
>750M transfers
>350 PB
Opportunistic estimate: ~ 250K CPU-hours / day

Friendly OSG sites

Help Us Help You

- David will discuss registration & engagement service
Session I
Getting Started on UC3

- Registration and login (interactive)
- Quick job submission tutorial
- Scaling up to more resources
- Job Submission from Midway or laptop
Web Site: http://uc3.uchicago.edu/
UC3 Quickstart

This is a quick start page which should take only a few minutes to complete. For more comprehensive UC3 usage documentation, go to the Job Submission guide or other user guides linked from UC3 Home.

Login to the UC3 Environment

- To access UC3 resources one needs a University of Chicago CNetID. If not already registered to UC3, go to the registration site and follow instructions there. Normally it takes a business day for approval.
- Once registered you are authorized to use uc3-sub.uchicago.edu (the Condor submit host) and uc3-data.uchicago.edu (the data host), in each case authenticating with your CNetID (cnetid) and password:
Set up the tutorial

You may perform the examples in the tutorial by typing them in from the text below, or by using tutorial files already on uc3-sub. It’s your choice; the tutorial is the same either way.

**Manual setup**

Create a new directory for the tutorial work:

```
$ cd
$ mkdir uc3-tutorial
$ cd uc3-tutorial
```

**Pretyped setup**

Alternatively, to save some typing, you can install the tutorial into your home directory from uc3-sub. This is *highly recommended* to ensure that you don’t encounter transcription errors during the tutorials.

This will work for all our tutorials - click here for details.

```
$ tutorial quickstart
$ cd ~/uc3-tutorial
```

Click to see full output

**Tutorial jobs**

**Job 1: A simple, nonparallel job**

*Create a workload*

Inside the tutorial directory that you created or installed previously, let's create a test script to execute as your
job:

$ vi short.sh
$ chmod +x short.sh

Here is the content of short.sh:

```bash
#!/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

echo "Environment:"
/bin/env | /bin/sort

echo "Dramatic pause..."
sleep $1-15 # Sleep 15 seconds, or however much we're told to sleep
echo "Et voila!"
```

*If you used the tutorial command, all files are already in your workspace.*

**Run the job locally**

When setting up a new job type, it's important to **test your job outside of Condor** before submitting into the grid. This helps you to catch errors before they have a chance to waste others' computation resources or, worse, freeze up global resources inequitably. This is a harmless example, but let's test it anyway:
Create a condor submit file:

The job looks good, so let’s move ahead into distributed high-throughput execution. The next step is to create a submission file.

$ vi tutorial01.sub

Here is the tutorial01.sub content, configured to use a special tutorial queue. The tutorial queue enqueues jobs at high priority (so that you can see results quickly), but has a limited number of job slots (so it’s not suitable for general use). This is very nearly the minimal content of a submission file.

<table>
<thead>
<tr>
<th>file: tutorial01.sub</th>
</tr>
</thead>
<tbody>
<tr>
<td># Submit description file for short test program</td>
</tr>
<tr>
<td>Universe = vanilla</td>
</tr>
<tr>
<td>Executable = short.sh</td>
</tr>
<tr>
<td>Error = log/job.err.$(Cluster)-$(Process)</td>
</tr>
<tr>
<td>Output = log/job.out.$(Cluster)-$(Process)</td>
</tr>
<tr>
<td>Log = log/job.log.$(Cluster)</td>
</tr>
<tr>
<td>+TutorialJob=true</td>
</tr>
<tr>
<td>+AccountingGroup = &quot;group_friends.cnetid&quot;</td>
</tr>
<tr>
<td>Queue 1</td>
</tr>
</tbody>
</table>

See the fully commented submission file here.
Submit the job using `condor_submit`.

```
$ condor_submit tutorial01.sub
Submitting job(s).
1 job(s) submitted to cluster 86090.
```

Note the "submitted to cluster 86090": this tells you the ID of the job group you've created. You'll use this for monitoring the status of your jobs.

### Check job status

The `condor_q` command tells the status of currently running jobs. Generally you will want to limit it to your own jobs:

```
$ condor_q cnetid
```

```
-- Submitter: uc3-sub.uchicago.edu : <10.1.3.94:9618?sock=25212_0c25_23> :
uc3-sub.uchicago.edu
ID   OWNER  SUBMITTED    RUN_TIME  ST  PRI  SIZE  CMD
86091.0  cnetid  5/21 16:18  0+00:00:10  R  0   0.0  short.sh
1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Note that `condor_q cnetid` lists only your jobs. `condor_q` will list all the jobs on the system.

You can also get status on a specific job cluster:

```
$ condor_q 86091
```

```
-- Submitter: uc3-sub.uchicago.edu : <10.1.3.94:9618?sock=25212_0c25_23> :
uc3-sub.uchicago.edu
ID   OWNER  SUBMITTED    RUN_TIME  ST  PRI  SIZE  CMD
86091.0  cnetid  5/21 16:18  0+00:00:10  R  0   0.0  short.sh
1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Note the `ST` column. Your job will be in the I state (idle) if the queue is busy. If it's currently scheduled and running, it will have state R (running). If it has completed already, will not appear in `condor_q`.

Let's wait for your job to finish – that is, for `condor_q` not to show the job in its output. `Watch` is a useful tool for that; it runs a program repeatedly, letting you see how the output differs at fixed time intervals. Let's submit the job again, and watch `condor_q` output at two-second intervals:
When your job is done, it will stop showing up and you can press **control-C** to exit `watch`.

**Job history**

Once your job has finished, you can get information about its execution from the `condor_history` command. `condor_history` takes similar options to `condor_q`'s options. Most simply, you can look at the job you just ran:

```
$ condor_history 86092
ID    OWNER          SUBMITTED   RUN_TIME     ST COMPLETED   CMD
86092.0  cnetid          5/21 16:22   0+00:01:00 C   5/21 16:23
/share/home/cnetid/uc3-quickstart/short.sh
```

You can see much more information about your job's final status using the `-long` option.

```
$ condor_history -long 86092 | sort
AccountingGroup = "group_friends.cnetid"
Arguments = ""
...।
```

There is a lot of information here, so the output is hidden. **Click to see it all.**

**Check the job output**

Again note your job ID from this job's execution. In the example above it was **86092.0**. (86092 is the *cluster id*, and 86092.0 is the *job id* for the first — and in this case only — job.) You will have 3 new files in your log directory. Their names are specified by the **log**, **output**, and **error** parameters in your submit file:

- a log file from Condor for the job cluster: `log/job.log.86092`
- an output file for each job's output (stdout): `log/job.out.86092-0`
- an error file for each job's errors (stderr): `log/job.err.86092-0`

Read the output file. It should be something like this:
Note that this job ran on uc3-sub. uc3-sub is part of the Tutorial queue, and when available is a first responder for short example jobs like this. In a moment we’ll see ramped-up uses that will execute on other worker nodes.

**Job 2: Submit more concurrent jobs**

Next, let’s submit 10 jobs at the same time. **Tutorial02.sub** is the same as **tutorial01.sub**, but replaces **Queue 1** with **Queue 10**.

```bash
$ cat log/job.out.86092-0
Start time: Tue May 21 16:22:35 CDT 2013
Job is running on node: uc3-sub.uchicago.edu
Job running as user: uid=200(cnetid) gid=200(uc3jamboree) groups=200(uc3jamboree)
Environment:
  PWD=/share/home/cnetid/uc3-quickstart
  SHLVL=1
  TMP=/var/lib/condor/execute/dir_17738
  TMPDIR=/var/lib/condor/execute/dir_17738
  _=/bin/env
  _CONDOR_ANCESTOR_17738=17740:1369171355:65080133
  _CONDOR_ANCESTOR_25212=25218:1367942563:4021420750
  _CONDOR_ANCESTOR_25218=17738:1369171355:426986731
  _CONDOR_JOB_AD=/var/lib/condor/execute/dir_17738/.job.ad
  _CONDOR_JOB_IWD=/share/home/cnetid/uc3-quickstart
  _CONDOR_JOB_PIDS=
  _CONDOR_MACHINE_AD=/var/lib/condor/execute/dir_17738/.machine.ad
  _CONDOR_SCRATCH_DIR=/var/lib/condor/execute/dir_17738
  _CONDOR_SLOT=slot4
Dramatic pause...
Et voila!

See the fully commented submission file here.

You’ll see something like the following upon submission:
$ condor_submit tutorial02.sub
Submitting job(s)...........
10 job(s) submitted to cluster 86093.

The number of periods is the number of jobs that condor_submit enqueued.

## Waiting for job completion

Using `watch -n2 condor_q cnetid` as before, watch these short jobs execute. You'll observe their state changes as they go from Idle to Running, then disappear from the queue as they complete.

For a different twist, we can tell Condor simply to wait until our jobs finish, using the `condor_wait` command. First submit `tutorial02.sub` again, then run `condor_wait` with your log file as an argument. By also using `time`, we can easily see approximately how long the whole job cluster takes.

$ condor_submit tutorial02.sub
Submitting job(s)...........
$ time condor_wait log/job.log.86094

All jobs done.

real  0m39.047s
user  0m0.003s
sys   0m0.007s

$ condor_history 86094

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>COMPLETED</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>86094.9</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:32 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.8</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:32 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.7</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:32 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.6</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:32 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.5</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:30 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.4</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:30 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.3</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:30 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.2</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:30 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.1</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:30 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
<tr>
<td>86094.0</td>
<td>cnetid</td>
<td>5/21 16:36</td>
<td>0+00:00:30 C</td>
<td>5/21</td>
<td>16:36</td>
<td>/share/home/cnetid/uc3-quickstart/short.sh</td>
</tr>
</tbody>
</table>

All 10 jobs now have the “C” state, indicating that they have completed. Compare the RUN_TIME for jobs in `condor_history` to the real time from `time`'s output. **This job cluster had good parallelism.**
Job 3: Passing arguments to executables

Sometimes it's useful to pass arguments to your executable from your submit file. For example, you might want to use the same job script for more than one run, varying only the parameters. You can do that by adding Arguments to your submission file. Let's try that with tutorial03.sub.

We want to run many more instances for this example: 100 instead of only 10. To ensure that we don't collectively overwhelm the scheduler let's also dial down our sleep time from 15 seconds to 5.

```
file: tutorial03.sub

# Submit description file for short test program

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)
+TutorialJob=true
+AccountingGroup = "group_friends.cnetid"
Queue 100
```

See the fully commented submission file here.

Nothing interesting is going to happen yet, so we'll just wait for this job to finish. Let's use the `time` `condor_wait` method, so we can compare wall time again.

```
$ condor_submit tutorial03.sub
Submitting
job(s)....................................................................................................
100 job(s) submitted to cluster 86097.
$ time condor_wait log/job.log.86097
All jobs done.
real 0m20.040s
user 0m0.010s
sys 0m0.007s
```

Despite having more jobs, with more aggregate clock time required (500s vs 300s), this can actually run faster because it's more parallel. This cluster ran in approx. 20s of real time, but did 500s of “work”. Previously, we could eyeball efficiency by looking at `condor_history`'s output, but with 100 jobs that's harder, and gets into some more advanced command usage. Click for some analysis of this job.

Where did jobs run?

With 100 jobs in the queue, we can begin to examine where our jobs are actually executing and get a sense of
how distributed this busywork is. To get that information, we’ll use a couple of `condor_history` commands. First, run `condor_history -long jobid` for your first job. Again the output is quite long:

```bash
$ condor_history -long 86097.0 | sort | less
```

```
AccountingGroup = "group_friends.cnetid"
Args = "5"
...
```

Click for the full output.

Looking through here for a hostname, we can see that the parameter that we want to know is `LastRemoteHost`. That’s what job slot our job ran on. With that detail, we can construct a shell command to get the execution node for each of our 100 jobs, and we can plot the spread. `LastRemoteHost` normally combines a slot name and a host name, separated by an `@` symbol, so we’ll use the UNIX `cut` command to slice off the slot name and look only at hostnames.

```bash
$ condor_history -format '%s
' LastRemoteHost 86097 | cut -d@ -f2 | distribution --height=100
```

```
<table>
<thead>
<tr>
<th>Val</th>
<th>Ct (Pct)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>uc3-t001.mwt2.org</td>
<td>16 (16.00%)</td>
<td>+++++++++++++++++++++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-sub.uchicago.edu</td>
<td>10 (10.00%)</td>
<td>+++++++++++++++++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c012.mwt2.org</td>
<td>6 (6.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c011.mwt2.org</td>
<td>6 (6.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c003.mwt2.org</td>
<td>6 (6.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c021.mwt2.org</td>
<td>5 (5.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c016.mwt2.org</td>
<td>4 (4.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c020.mwt2.org</td>
<td>4 (4.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c004.mwt2.org</td>
<td>4 (4.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c001.mwt2.org</td>
<td>4 (4.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c009.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c002.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c022.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c015.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c006.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c013.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c014.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c010.mwt2.org</td>
<td>3 (3.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c005.mwt2.org</td>
<td>2 (2.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c018.mwt2.org</td>
<td>2 (2.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c019.mwt2.org</td>
<td>2 (2.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c007.mwt2.org</td>
<td>2 (2.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c008.mwt2.org</td>
<td>2 (2.00%)</td>
<td>+++++++++++++++++++++</td>
</tr>
<tr>
<td>uc3-c017.mwt2.org</td>
<td>1 (1.00%)</td>
<td>+++</td>
</tr>
</tbody>
</table>
```

The `distribution` program reduces a list of hostnames to a set of hostnames with no reduplication – much like `sort | uniq -c` – but additionally plots a distribution curve on your terminal window. This is nice for seeing how Condor selected your execution endpoints. Obviously, job cluster 86097 ran primarily on `uc3-t001.mwt2.org`, which has 16 cores and, therefore, 16 job slots.

You can perform additional distribution analysis by looking at only part of the hostname. For example:
Not very interesting! All these jobs ran on the UC3 seeder. Let's look at another run of tutorial03.sub, done when the cluster was much busier.

That's more interesting. 57% of our workload ran in the UC3 seeder, while 43% ran on the UC ATLAS Tier 3 cluster.

**Job 4: Job requirements**

UC3 is not a cluster per se. UC3 is a platform for requesting available computational capacity from a collection of cooperating computational resources. Some of these are managed locally, while others are managed by other institutes or departments in the University. UC3 even has access to clusters at other institutions via the Open Science Grid. Each of these clusters may have its own distinct operational characteristics – duty cycles, available software, hardware capabilities, etc.

For our last exercise, let's look at how to select execution nodes based on the requirements of your job. Imagine for a moment that our sleep job requires access to the ATLAS Tier 3 data volumes. We can advise Condor of this with a `Requirements` item in the submission file. This will guarantee that the job only runs on nodes that have the indicated properties. In this instance, we're asking for worker nodes that have direct access to data for the University of Chicago ATLAS Tier 3 group. (In a later tutorial, we'll talk about giving Condor access to data that you don't have a direct link to.)

```
Requirements = (HAS_TIER3_DATA == true)
```

For this example, we'll also scale back down to 25 jobs sleeping one second each, because the point is to get a specific site, not to try to get several sites.
See the fully commented submission file here.

$ condor_submit tutorial04.sub
Submitting job(s).........................
25 job(s) submitted to cluster 86128.

This job could take a little longer, just for Condor's flocking mechanism to engage a remote cluster. UCT3 is an independent cluster with its own users and scheduler, so it is not necessarily as available as the seeder is. In fact, this job might not even run for a quite a while, if the Tier 3 is very busy.

When it is complete, we'll look at the job history to see where our 25 jobs ran:

$ condor_wait log/job.log.86128
All jobs completed.
$ condor_history -format '%s\n' LastRemoteHost 86128 | cut -d@ -f2 | distribution --height=100

<table>
<thead>
<tr>
<th>Val</th>
<th>Ct (Pct)</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>uct3-c026.mwt2.org</td>
<td>5 (20.00%)</td>
<td>++++++++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c024.mwt2.org</td>
<td>4 (16.00%)</td>
<td>++++++++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c028.mwt2.org</td>
<td>4 (16.00%)</td>
<td>++++++++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c034.mwt2.org</td>
<td>3 (12.00%)</td>
<td>++++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c032.mwt2.org</td>
<td>3 (12.00%)</td>
<td>++++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c027.mwt2.org</td>
<td>2 (8.00%)</td>
<td>+++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c033.mwt2.org</td>
<td>2 (8.00%)</td>
<td>+++++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c031.mwt2.org</td>
<td>1 (4.00%)</td>
<td>+++++++++++++++++</td>
</tr>
<tr>
<td>uct3-c023.mwt2.org</td>
<td>1 (4.00%)</td>
<td>+++++++++++++++++</td>
</tr>
</tbody>
</table>

Sure enough, all these jobs went to a Tier 3 worker node, indicated by the "uct3-*" hostnames. You can confirm this by looking at a job output file.
$ cat log/job.out.86128-0
Start time: Tue May 21 20:26:53 CDT 2013
Job is running on node: uct3-c026.mwt2.org
Job running as user: uid=105(uct3) gid=20013(uct3) groups=20013(uct3)
Environment:
PWD=/var/lib/condor/execute/dir_12080
SHLVL=1
TEMP=/var/lib/condor/execute/dir_12080
TMP=/var/lib/condor/execute/dir_12080
TMPDIR=/var/lib/condor/execute/dir_12080
=/bin/env
_CONDOR_ANCESTOR_12080=12101:1369186013:83652637
_CONDOR_ANCESTOR_14985=14986:1361822310:1247809390
_CONDOR_ANCESTOR_14986=12080:1369186012:3387557059
_CONDOR_JOB_AD=/var/lib/condor/execute/dir_12080/.job.ad
_CONDOR_JOB_IWD=/var/lib/condor/execute/dir_12080
_CONDOR_JOB_PIDS=
_CONDOR_MACHINE_AD=/var/lib/condor/execute/dir_12080/.machine.ad
_CONDOR_SCRATCH_DIR=/var/lib/condor/execute/dir_12080
_CONDOR_SLOT=slot1
Dramatic pause...
Et voila!

Yep. Note that on UCT3, flocked jobs from Condor run under a different UNIX user ID than your UC3 (CNet) ID. This is typical for Condor flocking to independent clusters: it means that we don't need unified identity management across all sites, and that makes all this possible.

See the Job Submission page for detailed specifications on how to submit to specific UC3-flocked resources.

Workload Analysis

Our Cycleserver website is an analytics tool that UC3 users should keep at hand while submitting jobs into UC3. Cycleserver helps you visualize what’s happening within several of UC3’s constituent clusters.

[Diagram]

We’ll talk more about this in the next tutorial. For now we’ll just note the following:

- The Cycleserver URL is http://uc3-mgt.uchicago.edu:8000/home.s
- You may log in using the username guest, with password guest.
- You can link directly to cluster utilization graphs through this list:
  - IT Services
  - ATLAS Midwest Tier 2 (IU)
• ATLAS Midwest Tier 2 (UC)
• ATLAS Midwest Tier 2 (UIUC)
• OSG VOFE
• UC3 Bosco
• UC3 Seeder
• ATLAS UC Tier 3

Getting help

If anything here didn't work, please email uc3-support@uchicago.edu.
Scaling up to more resources

Table of Contents

1 Case 1: I can run anywhere!
2 Case 2: I have hardware requirements!
3 Case 3: I need to access specific software!
4 Case 4: I need my data and CPU to be collocated!
5 Case 5: I want to use the grid!

Case 1: I can run anywhere!

What kind of considerations do you need to take into account when running anywhere?

1. Can I read my data remotely? Is it small enough to take with me?
2. Can I tarball my software and send it with my job? What's installed on the other side?

In the previous tutorial we used "short.sh" to do some basic tests. What happens when we try to submit 2500 of them?

First let's make ourselves a working directory, or simply run 'tutorial scaling' on uc3-sub.

```
[cnetid@uc3-sub ~]$ mkdir -p uc3-scaling/log; cd uc3-scaling
[cnetid@uc3-sub uc3-scaling]$  
```

I'm just going to use "short.sh" from previous tutorials, but I'm going to ramp it up by letting it run for 10 minutes and I'll try to queue 2500 jobs.

```
#!/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
echo "Environment:"
/bin/env | /bin/sort

echo "Dramatic pause..."
sleep $(1-15)  # Sleep 1 second, or however much we're told to sleep
echo "Et voila!"
```

And a slightly modified submit file that you should be a bit familiar with by now:
We'll send this off to condor_submit as per the usual.

We can see from the CycleServer view that I managed to hoard somewhere in the neighborhood of 1100 CPUs.

Notice the blue lines? Those represent other organizations running opportunistically here – those could be replaced with UC3 jobs!

What does the breakdown for this look like? It's a bit difficult to analyze the output programmatically, but I did some clean up by hand and here's what I got:
You can click here for some rough distribution analysis code.

These jobs will probably take quite a while to run! If you'd like, you can just remove them with `condor_rm`

```bash
[cnetid@uc3-sub uc3-scaling]$ condor_rm cnetid
User cnetid's job(s) have been marked for removal.
```

**Case 2: I have hardware requirements!**

Most of the nodes in the UC3 fabric have 2GB of RAM per CPU. What happens when you know a job needs more than that?

Let's create a job that has a memory requirement of about 2.5GB.

```bash
file: highmem.sub

Universe = vanilla
Executable = short.sh
Arguments = 1 # sleep time in seconds
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
request_memory = 2536 # requested in megabytes
Queue 100
```

Submit as usual:

```bash
[cnetid@uc3-sub uc3-scaling]$ condor_submit highmem.sub
Submitting job(s)....................................................................................................
100 job(s) submitted to cluster 86139.
```

After the jobs have completed, take a look at the distribution. We'll need to use some shell magic to get the output into the right format.
As you can see, the ITS AppCloud can support jobs that need up to 3GB of RAM per job.

Case 3: I need to access specific software!

There are also times when you might need to make sure that your jobs land on worker nodes with certain software installed.

Suppose, for example, you need to be able to access Mathematica, which is installed on the UC3 seeder worker nodes.

Creating a submit file is pretty straightforward.

<table>
<thead>
<tr>
<th>file: math.sub</th>
</tr>
</thead>
<tbody>
<tr>
<td># Submit description file for short test program</td>
</tr>
<tr>
<td>Universe = vanilla</td>
</tr>
<tr>
<td>Executable = short.sh</td>
</tr>
<tr>
<td>Arguments = 1 # sleep time in seconds</td>
</tr>
<tr>
<td>Error = log/job.err.$(Cluster)-$(Process)</td>
</tr>
<tr>
<td>Output = log/job.out.$(Cluster)-$(Process)</td>
</tr>
<tr>
<td>Log = log/job.log.$(Cluster)</td>
</tr>
<tr>
<td>Requirements = HAS_MATHEMATICA =?= TRUE</td>
</tr>
<tr>
<td>Queue 100</td>
</tr>
</tbody>
</table>

And fire it off:

[cnetid@uc3-sub uc3-scaling]$ condor_submit math.sub
Submitting
job(s)....................................................................................................c
100
job(s) submitted to cluster 86141.

Once it's completed, we can check the distribution as before:
As expected, the jobs only run where Mathematica is installed!

Case 4: I need my data and CPU to be collocated!

Sometimes you'll need to direct your jobs to a particular cluster. This is where HTCondor's "ClassAds" feature comes into play. ClassAds describe set of capabilities for each cluster, and your jobs can require certain attributes in order to run.

In this particular example, we want to use the ClassAd "HAS_TIER3_DATA". This will ensure that jobs only run on the UC ATLAS Tier 3 cluster.

Here's a pretty trivial script that will copy data from the Tier 3 data share to the job's working directory.

```
#!/bin/bash
echo "Copying test file"
cp -av /share/t3data2/lincolnb/lincoln.test.100MB .
```

And, naturally, the submit file:
Notice that we add the "HAS_TIER3_DATA" requirement to this submit file.

We'll submit this normally:

```
[cnetid@uc3-sub scaling]$ condor_submit tier3.sub
Submitting job(s).
1 job(s) submitted to cluster 86106.
```

When it's finished, we can see that HTCondor returned the 100MB file back to our working directory.

```
[cnetid@uc3-sub uc3-scaling]$ ls
copy_tier3_data.sh highmem.sub lincoln.test.100MB log math.sub short.sh
short.sub tier3.sub
```

What happens if we try to read that software on a machine that doesn't have it mounted?

**click here for the answer**

**Case 5: I want to use the grid!**

What if all of the UC3 resources are completely full? Fear not -- the Open Science Grid is here to help!

I've added a requirements line to our submit file that steers jobs out to the OSG specifically:
### file: osg.sub

```plaintext
# Submit description file for short test program
Universe = vanilla
Executable = short.sh
Arguments = 10
Error = log/job.err.$(Cluster)-$(Process)
Output = log/job.out.$(Cluster)-$(Process)
Log = log/job.log.$(Cluster)
Requirements = isUndefined(GLIDECLIENT_Name) == FALSE
Queue 100
```

These may take a while to start up. The Open Science Grid is a production environment, there's no guarantee that jobs will start immediately!

```
[cnetid@uc3-sub uc3-scaling]$ condor_submit osg.sub
Submitting
100 job(s) submitted to cluster 86148.
```

We can grep the output for "OSG_SITE_NAME" to get an idea where these jobs ran:

```
[cnetid@uc3-sub uc3-scaling]$ cat log/job.out.86148* | grep OSG_SITE_NAME |
distribution
    Val          |Ct (Pct)    Histogram
OSG_SITE_NAME-MWT2_CE_IU |30 (30.00%) ++++++++++++++++++++++++++++++++++++++++++
OSG_SITE_NAME-MWT2       |28 (28.00%) ++++++++++++++++++++++++++++++++++++++
OSG_SITE_NAME-MWT2_CE_UC |27 (27.00%) +++++++++++++++++++++++++++++++++++++
OSG_SITE_NAME=AGLT2_CE_2 | 9 (9.00%)   +++++++++++++
OSG_SITE_NAME=TTU-ANTAEUS| 6 (6.00%)   ++++++++  
```

Looks like most of these ran at Midwest Tier 2, but we also see some at Texas Tech and the ATLAS Great Lakes Tier 2!

And in the GWMS Frontend monitor we can check the Glideins that we triggered and the jobs on OSG:
Job Submission from Midway or Laptop

Table of Contents

1 Overview
2 Install and configure BOSCO
3 Before using BOSCO
4 Submit a job to UC3
   4.1 Create a workload
   4.2 Create a condor submit file:
   4.3 Check job status
5 Submit a job to Midway
6 Submit more jobs to UC3 and Midway
7 Other BOSCO commands

Overview

In this module we demonstrate job submission to the UC3 environment from RCC resources (Midway) or your laptop with BOSCO. This will allow you to manage jobs running in both environments from one host. Any Linux host can be used provided it runs RHEL5 or RHEL6 (and Scientific Linux distributions), Debian 6, or Mac OS X (10.5 or later).

To request a Midway account, see the RCC new user guide. In the examples below substitute cnetid with your actual CNetID.

Install and configure BOSCO

The following example is from Midway (midway-login1.rcc.uchicago.edu). If you are not on RCC you will not be able to access RCC directly (you still access it via UC3) and the prompt will be more something like [yourname@yourhost ~], mentioning your host instead of RCC's [cnetid@midway-login1 ~]. The rest will be the same.

1. Download the BOSCO installer package

   [cnetid@midway-login1 ~]$ wget -0 ./bosco_quickstart
   http://uc3-data.uchicago.edu/~marco/bosco_quickstart

   Expand to see the wget output
1. Run the quickstart script.

   [cnetid@midway-login1 ~]$ wget -O ./bosco_quickstart
   http://uc3-data.uchicago.edu/~marco/bosco_quickstart
   --2013-05-22 13:07:33--
   http://uc3-data.uchicago.edu/~marco/bosco_quickstart
   Resolving uc3-data.uchicago.edu... 128.135.158.217
   Connecting to uc3-data.uchicago.edu|128.135.158.217|:80... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 5859 (5.7K) [text/plain]
   Saving to: `.//bosco_quickstart'

   100%[================================================================================================================>
   5,859 --.-K/s in 0s
   2013-05-22 13:07:33 (530 MB/s) - `.//bosco_quickstart' saved [5859/5859]

   • NOTE: If you have no wget you can use curl to download the file: curl -o
   bosco_quickstart
   http://uc3-data.uchicago.edu/~marco/bosco_quickstart

   2. Run the quickstart script.

   [cnetid@midway-login1 ~]$ bash bosco_quickstart

   • 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
     uc3-sub.uchicago.edu and press ENTER.
   • When prompted "Type your name at uc3-sub.uchicago.edu (default
     YOUR_USER) and press [ENTER]:" enter your UChicago CNetID and press ENTER.
   • When prompted "Type the queue manager for uc3-sub.uchicago.edu
     (pbs, condor, lsf, sge, slurm) and press [ENTER]:" enter condor and
     press ENTER.
   • Then when prompted "cnetid@uc3-sub.uchicago.edu's password:" enter your
     UChicago CNetID password. click here to see the output

   3. Setup the environment

   [cnetid@midway-login1 ~]$ source ~/bosco/bosco_setenv

   4. BOSCO has been started for you but in the future you may need to restart it with:

   [cnetid@midway-login1 ~]$ bosco_start
   BOSCO Started
At this point, submission to uc3-sub.uchicago.edu, which gets to the full UC3 environment is now ready. The BOSCO services will remain running even if you log out unless explicitly shut down.

**Before using BOSCO**

Each time you login or start a new shell step the environment and invoke bosco_start (bosco_start is a no-op if the services are already running):

```
$ source ~/.bosco/bosco_setenv
$ bosco_start
BOSCO Started
```

**Submit a job to UC3**

Now run a simple job, like the Job 1 of the Quickstart tutorial. The workload is the same, the submit description file will be slightly different.

**Create a workload**

Inside the tutorial directory that you created or installed previously, let's create a test script to execute as your job (remember to make the script executable!):

```
$ vi short.sh
$ chmod +x short.sh
```

Here is the content of short.sh:

```bash
#!/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

echo "Environment:"
/bin/env | /bin/sort

echo "Dramatic pause..."
sleep ${1-15}    # Sleep 15 seconds, or however much we're told to sleep
echo "Et voila!"
```

**Create a condor submit file:**

The next step is to create a submission file and the log directories for the job.
Here is the bosco01.sub content, configured to use a special tutorial queue on uc3-sub.uchicago.edu. The tutorial queue enqueues jobs at high priority (so that you can see results quickly), but has a limited number of job slots (so it's not suitable for general use). This is very nearly the minimal content of a submission file.

Note that differently from the previous examples, now the Universe of the job is now grid. This tells BOSCO to run the job on the resource added during the setup.

```
$ mkdir log
$ vi bosco01.sub
```

Remember: Replace 'cnetid' with your own!

See the fully commented submission file here.

Submit the job using condor_submit.

```
$ condor_submit bosco01.sub
Submitting job(s).
1 job(s) submitted to cluster 1.
```

Note the "submitted to cluster 1": if you did a fresh installation of BOSCO the ID of the job group you've created is 1. You'll use this for monitoring the status of your jobs.

Check job status

The condor_q command tells the status of currently running jobs. Generally you will want to limit it to your own jobs:

```
2013-05-23
```
Submit a job to Midway

BOSCO by default submits to the cnetid@uc3-sub.uchicago.edu resource added in the setup above (grid_resource = batch condor cnetid@uc3-sub.uchicago.edu). If you are on Midway, to submit to the local queue on Midway, you must add the line grid_resource = batch pbs to the submit file.

To create bosco02.sub you can copy bosco01.sub and add the line grid_resource = batch pbs:

```bash
cp bosco01.sub bosco02.sub
vi bosco02.sub
```

Here is the content of bosco02.sub:

```
#Submit description file for short test program running on UC3

Universe = grid
grid_resource = batch pbs
Executable = short.sh
Error   = log/job.err.$(Cluster)-$(Process)
Output  = log/job.out.$(Cluster)-$(Process)
Log     = log/job.log.$(Cluster)
+TutorialJob=true
+AccountingGroup = "group_friends.cnetid"
Queue 1
```

Note BOSCO assumes PBS emulation for SLURM, Midway’s job manager. To submit and inspect the job repeat the steps in the previous section using bosco02.sub in place of bosco01.sub.

Submit more jobs to UC3 and Midway
This example submits 20 jobs each to both UC3 and Midway. To ease the observation of the job we'll increase the sleep time to 40 seconds. If you are not on Midway, then edit and submit only bosco01.sub to submit 20 jobs to UC3 and ignore bosco02.sub.

1. Edit both submit files bosco01.sub and bosco02.sub and add the line Arguments = 40 and change the last line to Queue 20:

```
$ vi bosco01.sub
$ vi bosco02.sub
```

Click here to see the new submit files

2. Submit both sets of 20 jobs:

```
[mmb@midway-login1 tutorial]$ condor_submit bosco01.sub
Submitting job(s)............... 20 job(s) submitted to cluster 5.
[mmb@midway-login1 tutorial]$ condor_submit bosco02.sub
Submitting job(s)............... 20 job(s) submitted to cluster 6.
```

3. Watch the jobs go through the queue by using `watch -n2 condor_q -grid`. The `-grid` option changes the format of condor_q and provides more information about where the jobs run.

```
$ condor_q -grid
   -- Submitter: midway-login1 : <128.135.112.71:11002?sock=21373_27a9_3> :
midway-login1
   ID OWNER             STATUS     GRID->MANAGER    HOST
GRID_JOB_ID
5.0   cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86277//
5.1   cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86273//
5.2   cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
5.3   cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
5.4   cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86275//
5.5   cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
5.6   cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
5.7   cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86272//
5.8   cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86278//
5.9   cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
5.10  cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86271//
5.11  cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86276//
5.12  cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
5.13  cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86269//
5.14  cnetid            IDLE       batch->mmb@uc3-sub.uchicago /86274//
5.15  cnetid            IDLE       batch->mmb@uc3-sub.uchicago
midway-login1_11
```
<table>
<thead>
<tr>
<th>Time</th>
<th>cnetid</th>
<th>State</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.16</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;<a href="mailto:mmb@uc3-sub.uchicago">mmb@uc3-sub.uchicago</a></td>
</tr>
<tr>
<td>5.17</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;<a href="mailto:mmb@uc3-sub.uchicago">mmb@uc3-sub.uchicago</a> /86270/</td>
</tr>
<tr>
<td>5.18</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;<a href="mailto:mmb@uc3-sub.uchicago">mmb@uc3-sub.uchicago</a></td>
</tr>
<tr>
<td>5.19</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;<a href="mailto:mmb@uc3-sub.uchicago">mmb@uc3-sub.uchicago</a></td>
</tr>
<tr>
<td>6.0</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.1</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.2</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.3</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.4</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.5</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.6</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.7</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.8</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.9</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.10</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.11</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.12</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.13</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.14</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.15</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.16</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.17</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
<tr>
<td>6.18</td>
<td>cnetid</td>
<td>IDLE</td>
<td>batch-&gt;[?] pbs</td>
</tr>
</tbody>
</table>

2013-05-23
6.19 cnetid            IDLE       batch->[?] pbs
midway-login1_11

Note that `condor_q` on your BOSCO installation will list only your jobs. There may be other jobs queued on UC3 but to see them you'll have to login on `uc3-sub.uchicago.edu` and issue `condor_q` there. Similarly on Midway, `qstat` will show the jobs from all the users while `condor_q` shows only the ones submitted from your BOSCO service.

### Other BOSCO commands

You can check the resources connected to BOSCO:

```bash
$ bosco_cluster --list
 cnetid@uc3-sub.uchicago.edu/condor
```

You can **stop** and **uninstall** BOSCO:

```bash
$ source ~/bosco/bosco_setenv
$ bosco_stop
Sending off command to condor_master.
Sent "Kill-Daemon" command for "master" to local master
Stopped HTCondor
BOSCO is now off.
$ bosco_uninstall
Ensuring Condor is stopped...
BOSCO is now off.
Removing BOSCO installation under /home/mmb/bosco
Done
```

All the HTCondor commands work form BOSCO. This document contains a detailed description of all the installation options and all the BOSCO commands.
Session II
Getting a Taste of UC3

- Using R on UC3
- Using Mathematica on UC3
Overview

This section covers how to use the UC3 CVMFS system to run a real application like R statistical package. For this example, we'll estimate the value of pi using a Monte Carlo method. We'll first run the program locally, then create a submit file, send it out to UC3, and collate our results.

Background

Some background is useful here. We define a square inscribed by a unit circle. We randomly sample points, and calculate the ratio of the points outside of the circle to the points inside for the first quadrant. This ratio approaches pi/4.

(See also: http://math.fullerton.edu/mathews/n2003/montecarlopimod.html)

This method converges extremely slowly, which makes it great for a CPU-intensive exercise (but bad for a real estimation!).
Accessing R on the submit host

First we'll need to create a working directory, you can either run 'tutorial R' or type the following:

```
[cnetid@uc3-sub ~]$ mkdir uc3-R; cd uc3-R
```

Since R is installed into CVMFS, it's not available in the normal system paths. We'll need to set up those paths so we can access R correctly:

```
[cnetid@uc3-sub uc3-R]$ export PATH=$PATH:/cvmfs/uc3.uchicago.edu/sw/bin
[cnetid@uc3-sub uc3-R]$ export LD_LIBRARY_PATH=/cvmfs/uc3.uchicago.edu/sw/lib64
```

Once we have the path set up, we can try to run R. Don't worry if you aren't an R expert, I'm not either.
Great! R works. You can quit out with "q()".

```
> q()
Save workspace image? [y/n/c]: n
[cnetid@uc3-sub uc3-R]$
```

Running R code

Now that we can run R, let's try using my Pi estimation code:

```
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(1000000)
```

R normally runs as an interactive shell, but it's easy to run in batch mode too.
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(1000000)
[1] 3.141688

This should take a little over a minute to run, but the estimation isn't very good. Fortunately, this problem is pleasingly parallel since we're just sampling random points. So what do we need to do to run R on the campus grid?

**Building the HTCondor job**

The first thing we're going to need to do is create a wrapper for our R environment, based on the setup we did in previous sections.
file: R-wraper.sh

#!/bin/bash

EXPECTED_ARGS=1

if [ $# -ne $EXPECTED_ARGS ]; then
  echo "Usage: R-wraper.sh file.R"
  exit 1
else
  PATH=$PATH:/cvmfs/uc3.uchicago.edu/sw/bin
  LD_LIBRARY_PATH=/cvmfs/uc3.uchicago.edu/sw/lib64
  R --slave < $1
fi

Notice here that I've changed R to --slave instead of --no-save. It makes R much less verbose and it's easier to parse the output later.

Now that we've created a wrapper, let's build a Condor submit file around it.

file: R.submit

universe = vanilla
log = log/mcpi.log.$(Cluster).$(Process)
error = log/mcpi.error.$(Cluster).$(Process)
output = log/mcpi.out.$(Cluster).$(Process)

# Setup R path, run the mcpi.R script
executable = R-wraper.sh
transfer_input_files = mcpi.R
arguments = mcpi.R
requirements = (HAS_CVMFS =?= TRUE)
queue 100

Notice the requirements line? You'll need to put HAS_CVMFS =?= TRUE any time you need software, such as R, from CVMFS. There's also one small gotcha here – make sure the "log" directory exists before you submit! Else Condor will fail because it has nowhere to write the logs.

Submit and analyze

Finally, submit the job to UC3!
Since our jobs just output their results to standard out, we can do the final analysis from the log files. Let's see what one looks like:

![Log file output]

```bash
[cnetid@uc3-sub uc3-R]$ cat log/mcpi.out.85823.1
[1] 3.14175
```

I'm just going to use a bit of awk magic to do the average for me.

```
[cnetid@uc3-sub uc3-R]$ cat log/mcpi.out.* | awk '{sum+=$2} END { print "Average = \$1", sum/NR}'
Average = 3.14151
```

Not bad!
Application Example - Mathematica

Table of Contents

1 Overview
2 Background
3 Running Mathematica locally
4 Creating an HTCondor job
5 Stitching the output together
6 Creating an HTCondor DAG
7 Pretty pictures

Overview

The second application that we'll run on UC3 is Mathematica. This application example will introduce the use of HTCondor Directed Acyclic Graphs to manage job workflows. Specifically, we'll take the Mandelbrot set calculation, split it up into \( N \) jobs, and then recombine the final data to produce a nice picture.

Background

This exercise will calculate the Mandelbrot set, one of the most easily recognized fractals. You can read more about the Mandelbrot set here

Running Mathematica locally
We'll need to set up a working directory, so you can either type 'tutorial mathematica' or type the following:

[cnetid@uc3-sub ~]$ mkdir -p uc3-mathematica/{csv,log}; cd uc3-mathematica

Like R, we can get Mathematica from CVMFS. It's installed at /cvmfs/uc3.uchicago.edu/Wolfram, so we'll need to export our PATH appropriately.

[cnetid@uc3-sub uc3-mathematica]$ export
PATH=$PATH:/cvmfs/uc3.uchicago.edu/Wolfram/Mathematica/8.0/Executables

Once we've got Mathematica in our PATH, let's try running it.

[cnetid@uc3-sub uc3-mathematica]$ math
Mathematica 8.0 for Linux x86 (64-bit)
Copyright 1988-2011 Wolfram Research, Inc.
In[1]:=

Looks like it works. You can quit with Quit[]

In[1]:= Quit[]
[cnetid@uc3-sub uc3-mathematica]$ 

I've written the Mandelbrot set code for you already. I'll spare you the gory details, but the important things to note here is that it takes 2 variables as input: "PID" and "Jobs".
**mandelbrot.m**

```plaintext
Scaling = 5
TotalCols = LCM[7/4, 2]*Jobs*Scaling
TotalRows = Ceiling[(TotalCols - 1)*4/7]
maxIterations = 120;

MandelbrotPixel = 
    Compile[{{ColNum, _Integer}, {RowNum, _Integer}},
        Module[{x = 0., y = 0., xtemp, iterations = 0},
            While[x^2 + y^2 <= 4 && iterations < maxIterations,
                xtemp = x*x - y*y + (ColNum - 1)*3.5/TotalCols - 2.5;
                y = 2*x*y + (RowNum - 1)*2/TotalRows - 1;
                x = xtemp;
                iterations = iterations + 1;]
            ];
            iterations], CompilationTarget -> "C";

MandelbrotData =
    Table[MandelbrotPixel[i, j], {j, 1, TotalRows}, {i, (PID - 1)*TotalCols/Jobs + 1, (TotalCols/Jobs)*PID}];

Export[StringJoin["mandelbrot." <> IntegerString[PID, 10, IntegerLength[Jobs]] <> ".csv"],
    MandelbrotData, "CSV"]
```

There's also a scaling factor at the top of the code that you are free to modify. If you run 10 jobs with the default scaling, you can expect to produce a 700x400 pixel Mandelbrot.

Let's try running this from the command line. We'll pass in PID and Jobs as arguments. 'Jobs' determines how many times we want to cut up the calculation, and 'PID' specifies which chunk we want to evaluate.

```
[cnetid@uc3-sub uc3-mathematica]$ math -run "PID=1;Jobs=10" < mandelbrot.m
Mathematica 8.0 for Linux x86 (64-bit)  
Copyright 1988-2011 Wolfram Research, Inc.

In[1]:= Out[1]= 700

In[2]:= Out[2]= 400

In[3]:= In[4]:= In[5]:= In[6]:= In[6]=
Out[6]= mandelbrot.01.csv

In[7]:= [cnetid@uc3-sub uc3-mathematica]$
```
There’s a lot of stuff on standard out, but we see that "mandelbrot.01.csv" was created.

You can look at this is data if you'd like, but it's probably not very interesting yet.

**Creating an HTCondor job**

How do we wrap this up into a HTCondor job? First we need to create a small script that sets up our environment variables and runs Mathematica in batch mode. Here’s my script:

```bash
#!/bin/bash
# This script assumes that 1 argument has no additional arguments
# and that 3 arguments follows the form PID and Jobs

export PATH=$PATH:/cvmfs/uc3.uchicago.edu/Wolfram/Mathematica/8.0/Executables

if [ $# -eq 1 ]; then
    math -run < $1
elif [ $# -eq 2 ]; then
    math -run "PID=`expr $2 + 1`" < $1
elif [ $# -eq 3 ]; then
    math -run "PID=`expr $2 + 1`;Jobs=$3" < $1
else
    echo "Wrong number of arguments"
    echo "Usage: math.sh batch.m [PID] [Number of Jobs]"
fi
```

We're going to be creating another Mathematica batch file later, so this script has been made a bit generic. Let's create a Condor submit file to go along with it.

**Remember: Replace 'cnetid' with your own!**
This Condor submit script is a bit more complex than others for a few reasons.

- I don't like to pollute my submit directory with data and logs, so I split them off into directories called 'csv' and 'logs' respectively.
- I also want to make sure that my Mathematica script is aware of how many jobs are being made, so I define a macro called "Jobs" and pass it to arguments and queue.
- Mathematica requires a license! We use a boolean expression to make sure the CVMFS and the license server are available to the worker.
- Finally, $(Process)$ translates to our Mathematica script's PID variable.

**Stitching the output together**

The problem is that this code will create 10 separate CSVs for us to stitch together, when we actually just want one to make a nice picture. I've created a Mathematica script for handling this as well.

```
stitch.m
Export["mandelbrot.csv",
Flatten[Table[
    Transpose[Import[FileNames["*.csv"]][[i]]], {i, 1, 
    Length[FileNames["*.csv"]]], 1]];
```

As before, we create a submit file. This time we'll transfer the entire contents of "csv" with the job.

**Remember: Replace 'cnetid' with your own!**
Creating an HTCondor DAG

We could submit these by hand, but why not let HTCondor's workflow manager take care of it? Enter DAGs, or Directed Acyclic Graphs. I won't bother to explain in detail, but they let you create job workflows and also have some nice features like automatic retry in the event of failure. You can read more about them here.

Writing one for our jobs is easy since our workflow is pretty linear. This DAG will launch mandelbrot.submit, which creates our CSV files, and then, once finished, collates the output with stitch.submit.

Submitting DAGs is just as easy as submitting HTCondor jobs:

```
[cnetid@uc3-sub uc3-mathematica]$ condor_submit_dag mandelbrot.dag
```

```bash
File for submitting this DAG to Condor : mandelbrot.dag.condor.sub
Log of DAGMan debugging messages      : mandelbrot.dag.dagman.out
Log of Condor library output          : mandelbrot.dag.lib.out
Log of Condor library error messages  : mandelbrot.dag.lib.err
Log of the life of condor_dagman itself: mandelbrot.dag.dagman.log
```

```
Submitting job(s).
1 job(s) submitted to cluster 85959.
```
The DAG will fire up all of our math.sh scripts for us:

```
<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>85959.0</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:14</td>
<td>R</td>
<td>0</td>
<td>0.3</td>
<td>condor_dagman</td>
</tr>
<tr>
<td>85960.0</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.1</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.2</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.3</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.4</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.5</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.6</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.7</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.8</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
<tr>
<td>85960.9</td>
<td>cnetid</td>
<td>5/18 13:43</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>0.0</td>
<td>math.sh mandelbrot</td>
</tr>
</tbody>
</table>
```

And once it's finished, you should see the completed "mandelbrot.csv" in your homedir!

### Pretty pictures

Sadly I don't think Mathematica can export graphics without having the GUI running, so you'll need to 'scp' the code to your laptop or another machine that has Mathematica installed.

Nevertheless, here's the (trivial) code for getting Mathematica to plot it:

```
Mandelbrot = Import["/Users/cnetid/mandelbrot/mandelbrot.csv"]
ArrayPlot[Transpose[Mandelbrot], ColorFunction -> "Rainbow"]
```

Here it is!
Session III
Software and Data Access Fundamentals

• Software access in the UC3 environment
  • UC3 storage system
• Accessing data and software from anywhere
  • Using Globus Online with UC3 Storage
Software access in the UC3 environment

Table of Contents

1 Introduction
2 Starting Up
3 HTCondor File Transfer Method
4 Distributing Applications Using the UC3 Web Server
5 Use the CernVM-FS (CVMFS) Network Filesystem
6 Software available in the UC3 CernVM-FS repository

Introduction

This module will illustrate how software is accessed in UC3. Three different methods are described. For your convenience, all of the example files mentioned here can be found in /opt/uc3/access on uc3-sub.uchicago.edu or uc3-data.uchicago.edu.

Starting Up

Before going through the examples, login to uc3-sub.uchicago.edu and setup a workarea:

```
% ssh uc3-sub.uchicago.edu
% mkdir -p access_tutorial/logs
% cd access_tutorial
%
```

Then set up a public_html directory as well:

```
% mkdir ~/public_html
% chmod 755 ~/public_html
%
```

HTCondor File Transfer Method

If the software that your job needs isn't installed on the compute nodes, you can have HTCondor transfer the files to compute nodes. To do this, use the should_transfer_files and transfer_input_files options in the submit file. Let's look at ex1_transfer.submit which can be found in /opt/uc3/access/:
Now create the files and directories that HTCondor will transfer:

```
test.sh
```

```bash
#!/bin/bash  
cat test_dir/hello

% mkdir test_dir  
% echo "hello world" > test_dir/hello
```

Finally, change submit file to by replacing cnetid with your CNET id and submit to HTCondor.

```
% vi ex1_transfer.submit
% condor_submit ex1_transfer.submit
Submitting job(s).
10 job(s) submitted to cluster 86083.
```

This technique is well suited for distributing small applications or data files or for ad-hoc job submissions.

### Distributing Applications Using the UC3 Web Server

If your application is fairly large or if you notice that the file transfer mechanism is causing inefficiencies in your jobs, you can the UC3 web server to distribute your application.

First, create the HTCondor submit script `ex2_staging.submit`:
# Submit description file for short test program using staging

## ex2_staging.submit

```
Universe       = vanilla
Executable     = ex2_start.sh
Error          = logs/stage.err.$(Cluster)-$(Process)
Output         = logs/stage.out.$(Cluster)-$(Process)
Log            = logs/stage.log.$(Cluster)
should_transfer_files = YES
+AccountingGroup = "group_friends.cnetid"
```

Queue 10

Next, create or copy a tar file (from /opt/uc3/access/ex2.tar.gz) containing our application. We'll use the same files that we used in the file transfer example:

```
% mkdir app/
% cp test.sh app
% cp -r test_dir app/
% chmod 755 app/test.sh
% tar cvzf ex2.tar.gz app
```

Then, copy the file to your public_html directory:

```
% cp ex2.tar.gz ~/public_html
% chmod 644 ~/public_html/ex2.tar.gz
```

Create a shell script ex2_start.sh that when run will download the tar file, extract it, and run the script found inside the file:

**Remember to replace "cnetid" with your username!**

```
#!/bin/bash

cur_dir=`cwd`
temp_dir=`mktemp -d`

cd $temp_dir
wget http://uc3-data.uchicago.edu/~cnetid/ex2.tar.gz
tar xzf ex2.tar.gz
cd app
./test.sh
cd $cur_dir
rm -fr $temp_dir
```

Finally, edit your submit file by replacing cnetid with your CNET id and submit to HTCondor:
Use the CernVM-FS (CVMFS) Network Filesystem

When one requires software that is frequently used (e.g. community software such as R, MatLab, Mathematica, etc.), CVMFS may be the way to go. **Expand for more.**

To use CVMFS, the target compute cluster must either have CVMFS installed or one must use SkeletonKey/Parrot (discussed elsewhere). Clusters in the UC3 environment that have CVMFS installed will have the HAS_CVMFS ClassAd set to TRUE. The following submit file (ex3_mathematica.submit) uses the HAS_CVMFS requirement to select nodes that provide Mathematica.

First create or copy the `ex3_mathematica.submit` file:

```bash
#!/bin/bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/cvmfs/uc3.uchicago.edu/sw/lib
/cvmfs/uc3.uchicago.edu/sw/bin/Rscript ex3_test.R
```

Some software requires a modification to the `LD_LIBRARY_PATH` so that their libraries can be found. R is one of these applications, therefore, we will use a wrapper script to setup the `LD_LIBRARY_PATH` before calling R e.g.:
Next, create a submit file called \texttt{ex3\_R.submit} for R:

```
###
# Submit description file for short test program using CVMFS
###
Universe       = vanilla
Executable     = ex3\_R.sh
Error   = logs/R.err.$(Cluster)-$(Process)
Output  = logs/R.out.$(Cluster)-$(Process)
Log     = logs/R.log.$(Cluster)
Requirements = (HAS_CVMFS =?= TRUE)
+AccountingGroup = "group\_friends.cnetid"
Transfer\_executable = True
Should\_transfer\_files = True
transfer\_input\_files = ex3\_test.R

Queue 10
```

Now, create the \texttt{ex3\_test.R} script:

```
hilbert<-function(n) 1/(outer(seq(n) ,seq(n) ,"+")-1)
print("hilbert n=500")
print(system.time(eigen(hilbert(500))))
print("hilbert n=1000")
print(system.time(eigen(hilbert(1000))))
print("sort n=6")
print(system.time(sort(rnorm(10^6))))
print("sort n=7")
print(system.time(sort(rnorm(10^7))))
# loess
loess.me<-function(n) {
print(paste("loess n="",as.character(n) ,sep=""))
for (i in 1:5) {
  x<-rnorm(10^n); y<-rnorm(10^n); z<-rnorm(10^n)
  print(system.time(loess(z\~x+y)))
}
}
loess.me(3)
loess.me(4)
```

Finally, edit the submit file by replacing cnetid with your CNET id and submit the file to HTCondor:

```
% vi ex3\_R.submit
% condor\_submit ex3\_R.submit
```

Software available in the UC3 CernVM-FS repository
The following software is available on the UC3 CVMFS server:

<table>
<thead>
<tr>
<th>Application</th>
<th>Location</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mathematica</td>
<td>/cvmfs/uc3.uchicago.edu/Wolfram/Mathematica/8.0/Executables</td>
<td>8.0</td>
</tr>
<tr>
<td>R</td>
<td>/cvmfs/uc3.uchicago.edu/sw/bin</td>
<td>2.15.2</td>
</tr>
<tr>
<td>Octave</td>
<td>/cvmfs/uc3.uchicago.edu/groups/swift/bin</td>
<td>3.6.4</td>
</tr>
<tr>
<td>Python/Scipy/Numpy</td>
<td>/cvmfs/uc3.uchicago.edu/groups/spt/bin/</td>
<td>Python 2.7 / SciPy 0.9.0 / NumPy 1.6.1</td>
</tr>
<tr>
<td>ROOT</td>
<td>/cvmfs/uc3.uchicago.edu/groups/dchooz/ROOT-5.24/bin</td>
<td>5.24</td>
</tr>
<tr>
<td>ROOT</td>
<td>/cvmfs/uc3.uchicago.edu/groups/dchooz/ROOT-5.34/bin</td>
<td>5.34</td>
</tr>
</tbody>
</table>
UC3 storage system

Table of Contents

1 Description
2 Storage policy
3 References

Description

UC3 maintains a storage system based on the Hadoop Distributed File System (HDFS). To examine this, login to uc3-data.uchicago.edu and browse /mnt/hdfs/cnetid. The directory convention under /mnt/hdfs is:

/mnt/hdfs/users/cnetid/ - for individual users (semi-persistent data)
/mnt/hdfs/scratch/cnetid/ - simple scratch

Data stored here is accessible from the UC3 Cycle Seeder cluster (uc3-*), and from anywhere else using either Globus Online or SkeletonKey/Parrot. Presently the system has 52 TB of usable capacity. As the HDFS replication factor is 1 (two copies of each file in the system), the effective capacity is 26 TB.

While we currently use HDFS, we may elect at some future date to deploy a fully POSIX compliant file system such as Luture.

Storage policy

• The storage area is not backed up. It should be considered scratch space.
• No user quotas are enabled.
• If overall system capacity exceeds 80%, we will automatically delete files based on an LRU algorithm.

References

• See Using HDFS filesystem for more information.
Accessing data and software from anywhere

Table of Contents

1 Introduction
2 Starting Up
3 Remote software access
   3.1 The CVMFS network file system
   3.2 Simplify software access with SkeletonKey
   3.3 Creating the application tarball
   3.4 Creating a job wrapper
   3.5 Using the job wrapper
   3.6 Submitting jobs to UC3
4 Remote data access
   4.1 Chirp data server
   4.2 Data access example
   4.3 Creating the application tarball
   4.4 Creating a job wrapper
   4.5 Verification
   4.6 Submitting jobs to UC3
5 Combining both software and data access
   5.1 Creating the application tarball
   5.2 Submitting to UC3

Introduction

This module will introduce using SkeletonKey to users and explain how to use it to access software and data in the UC3 environment. SkeletonKey is a tool that allows users to easily use the capabilities of Parrot and Chirp data server to allow user applications to transparently access software and data remotely.

Starting Up

Before going through the examples, login to uc3-data.uchicago.edu and setup a workarea:

```bash
% ssh uc3-data.uchicago.edu
% mkdir -p sk_tutorial/logs
% cd sk_tutorial
```

Let's also set up the public_html directory as well:
Finally, we'll need to install SkeletonKey, setup SkeletonKey, and start Chirp:

```
% mkdir ~/public_html
% chmod 755 ~/public_html
```

Remote software access

The CVMFS network file system

CVMFS is a remote access protocol that allows a read-only filesystem to be exported using a web server. Using FUSE or Parrot, this filesystem can be mounted on a system and will appear to be a local filesystem and can be used to run applications installed on the exported filesystem. The following example will show how to mount a CVMFS repository and use it to run your applications.

Simplify software access with SkeletonKey

Creating the application tarball

Since we'll be running an application from a CVMFS repository, we'll create an application tarball to do some initial setup and then run the actual application.

Create a directory for the script

```
% mkdir -p cvmfs/cvmfs_access
% cd cvmfs
```

Create a shell script, `/sk_tutorial/cvmfs/cvmfs_access/ex1.sh` with the following lines or copy it from `/opt/uc3/sk`:

```
#!/bin/bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/cvmfs/uc3.uchicago.edu/sw/lib
/cvmfs/uc3.uchicago.edu/sw/bin/Rscript ./cvmfs_access/test.R
echo "Finishing script at: "
echo `date`
```
Create a R script ~/sk_tutorial/cvmfs/cvmfs_access/test.R with the following lines or copy it from /opt/uc3/sk:

```r
hilbert<-function(n) 1/(outer(seq(n) ,seq(n) ,"+")-1)
print("hilbert n=500")
print(system.time(eigen(hilbert(500)))))
print("hilbert n=1000")
print(system.time(eigen(hilbert(1000)))))
print("sort n=6")
print(system.time(sort(rnorm(10^6))))
print("sort n=7")
print(system.time(sort(rnorm(10^7))))
loess
loess.me<-function(n) {
  print(paste("loess n=",as.character(n) ,sep=""))
  for (i in 1:5) {
    x<-rnorm(10^n); y<-rnorm(10^n); z<-rnorm(10^n)
    print(system.time(loess(z~x+y)))
  }
  loess.me(3)
  loess.me(4)
```

Next, make sure the ex1.sh script is executable and create a tarball:

```bash
% chmod 755 ~/sk_tutorial/cvmfs/cvmfs_access/ex1.sh
% cd ~/sk_tutorial/cvmfs
% tar cvzf cvmfs_access.tar.gz cvmfs_access
```

Then copy the tarball to your public_html directory

```bash
% cd ~/sk_tutorial/cvmfs
% cp cvmfs_access.tar.gz ~/public_html
% chmod 644 ~/public_html/cvmfs_access.tar.gz
```

One thing to note using CVMFS through Parrot works the same way as using the CVMFS repositories on UC3 nodes that have them mounted. In particular, your application may require modifications to the LD_LIBRARY_PATH to function correctly.

**Creating a job wrapper**

You'll need to do the following on the machine where you installed SkeletonKey
Open a file called ~/sk_tutorial/cvmfs/cvmfs_access.ini and add the following lines or copy it from /opt/uc3/sk:
Run SkeletonKey on cvmfs_access.ini:

```
% cd ~/sk_tutorial/cvmfs
% ~/bin/skeleton_key -c cvmfs_access.ini
```

Run the job wrapper to verify that it's working correctly

```
% python run_job.py
```

**Using the job wrapper**

**Submitting jobs to UC3**

The following part of the tutorial is optional and will cover using a generated job wrapper in a HTCondor submit file.

On uc3-sub.uchicago.edu, create a file called ~/sk_tutorial/cvmfs/cvmfs.submit with the following contents

```
cvmfs.submit

universe = vanilla
notification=never
executable = ./run_job.py
output = logs/test_$(Cluster).$(Process).out
error = logs/test_$(Cluster).$(Process).err
log = logs/test.log
ShouldTransferFiles = YES
when_to_transfer_output = ON_EXIT
queue 1
```

Next, create a directory the log and output files
Finally submit the job to HTCondor and verify that the jobs ran successfully:

```
[cnetid@uc3-sub cvmfs] cd ~/sk_tutorial/cvmfs
[cnetid@uc3-sub cvmfs] condor_submit cvmfs.submit
```

**Remote data access**

**Chirp data server**

SkeletonKey installs chirp_control to so that the user start and stop the Chirp data server. In order to start Chirp, run `chirp_control start`. Stop Chirp by running `chirp_control stop`.

**Data access example**

The next example will guide the user through creating a job that will read and write from a filesystem exported by Chirp.

**Creating the application tarball**

First, create an application tarball with an script that will access data remotely:

First login to `uc3-data.uchicago.edu`

```
% ssh uc3-data.uchicago.edu
```

Create a directory for the script:

```
% mkdir -p ~/sk_tutorial/data/data_access
% cd ~/sk_tutorial/data
```

Create a shell script, `~/.sk_tutorial/data/data_access/ex2.sh` with the following lines:

```
#!/bin/bash

echo "testing output on: `date`" > $CHIRP_MOUNT/data_access_test

# cat $CHIRP_MOUNT/data_access_test
```

Next, make sure the `ex2.sh` script is executable and create a tarball:
% cd ~/sk_tutorial/data/
% chmod 755 data_access/ex2.sh
% tar cvzf data_access.tar.gz data_access

Then copy the tarball to your public_html directory

% cp data_access.tar.gz ~/public_html
% chmod 644 ~/public_html/data_access.tar.gz

Notice the use of the $CHIRP_MOUNT variable when reading or writing to the the directory exported through Chirp. SkeletonKey defines and sets $CHIRP_MOUNT so that it will correspond to the directory being exported from the chirp server.

Creating a job wrapper

You’ll need to do the following on uc3-data.uchicago.edu
Open a file called ~/sk_tutorial/data/data_access.ini and add the following lines:

```
data_access.ini

[Directories]
extport_base = /mnt/hdfs/users/cnetid
read = /
write = /

[Parrot]
location = http://uc3-data.uchicago.edu/parrot

[Application]
location = http://uc3-data.uchicago.edu/~cnetid/data_access.tar.gz
script = ./data_access/ex2.sh
```

Run SkeletonKey on data_access.ini:

% ~/bin/skeleton_key -c data_access.ini

Run the job wrapper to verify that it’s working correctly

% python run_job.py

Verification

On uc3-data.uchicago.edu, run the following following to verify that the file was written correctly:
The output should match the output given in the example above.
Once the output is verified, delete the output file

% rm /mnt/hdfs/users/cnetid/data_access_test

Submitting jobs to UC3

On uc3-sub.uchicago.edu, create a file called ~/sk_tutorial/data/data.submit with the following contents

```
universe = vanilla
notification=never
executable = ./run_job.py
output = logs/data_${Cluster}.$(Process).out
error = logs/data_${Cluster}.$(Process).err
log = logs/data.log
ShouldTransferFiles = YES
when_to_transfer_output = ON_EXIT
+AccountingGroup = "group_friends.cnetid"
queue 1
```

Next, create the directory for log and output files:

% mkdir ~/sk_tutorial/data/logs

Finally edit the submit file to use the correct cnet id, submit the job to HTCondor, and verify that the jobs ran successfully

% ssh uc3-sub.uchicago.edu
[cnetid@uc3-sub ~] cd sk_tutorial/data/
[cnetid@uc3-sub data] condor_submit data.submit

Combining both software and data access

The next example show how to combine data and software access in a single job.

Creating the application tarball

Since we'll be running an application from a CVMFS repository, we'll create an application tarball to do some initial setup and then run the actual application.
Login to `uc3-data.uchicago.edu`

```
% ssh uc3-data.uchicago.edu
```

Create a directory for the script

```
% mkdir -p ~/sk_tutorial/combined/combined_access
% cd ~/sk_tutorial/combined/
```

Create a shell script, `~/sk_tutorial/combined/combined_access/ex3.sh` with the following lines:

```
ex3.sh

#!/bin/bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/cvmfs/uc3.uchicago.edu/sw/lib
/cvmfs/uc3.uchicago.edu/sw/bin/Rscript ./combined_access/test2.R
./combined_access/data.grb $CHIRP_MOUNT/output/$1
echo "Finishing script at: "
echo `date`
```

Create a R script, `~/sk_tutorial/combined/combined_access/test2.R` with the following lines:

```
#!/usr/bin/Rscript --vanilla
library( raster)
args <- commandArgs(TRUE)
grbFile <- args[1]
scanHowMany <- args[2]
output <- args[3]
grb <- brick( grbFile)
a <- vector()
for( n in 1:scanHowMany) {
  r <- subset( grb, n)
  a <- append(a, paste( names( r), cellStats( r, "sum"), sep= " "))
}
cat( a, sep="\n", file=output)
```

Next, make sure the `ex3.sh` script is executable and create a tarball:

```
% chmod 755 ~/sk_tutorial/combined/combined_access/ex3.sh
% cd ~/sk_tutorial/combined
% cp /opt/uc3/sk/data.grb combined_access
% tar cvzf combined_access.tar.gz combined_access
```
Then copy the tarball to your public_html directory

```bash
% cp combined_access.tar.gz ~/public_html
% chmod 644 ~/public_html/combined_access.tar.gz
```

Finally, create the directories that the R script will access

```bash
% mkdir /mnt/hdfs/users/cnetid/output
% cp ~/.chirp/acl /mnt/hdfs/users/cnetid/output/.__acl
```

**Creating a job wrapper**

You'll need to do the following on the `uc3-data.uchicago.edu`

Open a file called `~/sk_tutorial/combined/combined.ini` and add the following lines:

```ini
combined.ini

[CVMFS]
repo1 = uc3.uchicago.edu
repo1_options =
url=http://uc3-cvmfs.uchicago.edu/opt/uc3/,pubkey=http://uc3-data.uchicago.edu/
= http://uc3-data.uchicago.edu/keys/uc3.key

[Directories]
export_base = /mnt/hdfs/user/cnetid
read = /, data
write = /, output

[Parrot]
location = http://uc3-data.uchicago.edu/parrot

[Application]
location = http://uc3-data.uchicago.edu/~cnetid/combined-access.tar.gz
script = ./combined_access/ex3.sh
```

Run **SkeletonKey** on `combined.ini`:

```bash
% ~/bin/skeleton_key -c combined.ini
```

Run the job wrapper to verify that it's working correctly

```bash
% python run_job.py test.output
```

**Submitting to UC3**
The following part of the tutorial is optional and will cover using a generated job wrapper in a HTCondor submit file.

On uc3-sub.uchicago.edu, create a file called ~/sk_tutorial/combined/combined.submit with the following contents

```
combined.submit

universe = vanilla
notification=never
executable = ./run_job.py
arguments = test.output.$(Process)
output = logs/test_${Cluster}.${Process}.out
error = logs/test_${Cluster}.${Process}.err
log = logs/test.log
ShouldTransferFiles = YES
when_to_transfer_output = ON_EXIT
queue 5
```

Next, create a directory for log and output files

```
[cnetid@uc3-sub ~] mkdir ~/sk_tutorial/combined/logs
```

Finally submit the job to HTCondor and verify that the jobs ran successfully

```
[cnetid@uc3-sub ~] cd ~/sk_tutorial/combined
[cnetid@uc3-sub combined] condor_submit combined.submit
```

Something to note in the HTCondor submit file, is that we're passing the name of the output file that should be written using the arguments setting and then using the $(Process) variable to ensure that each queued job writes to a different file. HTCondor will then pass the variable to the run_job.py which then makes sure that it gets appended to the arguments passed to the ex3.sh script.
Using Globus Online with UC3 Storage

Table of Contents

1 Introduction
2 Getting Started
3 Define Globus Online Endpoints for UC3
4 Activating the uc3#uc3-data endpoint
5 Transferring files

Introduction

The basics for data transfers into/from this storage system using Globus Online are discussed. This module assumes no familiarity with Globus Online. For additional information see the quick start guide and Globus Online tutorials.

For technical HDFS reasons, the file system to use for Globus Online transfers is /mnt/hdfs-nfs.

Getting Started

We will demonstrate transferring files from your laptop or Linux host to UC3 HDFS using Globus Connect.

1. If you don't have a Globus Online account, go to the Globus Online signup page and sign up.
2. Go to the Globus Online downloads page and download the appropriate installer.
3. Install and start the software, adding your login details when asked to do so.

Define Globus Online Endpoints for UC3

An endpoint has already been defined for the UC3 HDFS storage system; it is uc3#uc3-data. Globus Connect creates and activates an adhoc endpoint, so the only endpoint that needs to be activated is the uc3#uc3-data endpoint.

Activating the uc3#uc3-data endpoint

The Globus online endpoint uses CILogon which simplifies integration with the uc3-sub.uchicago.edu login host (as well as Midway). The steps required are as follows:

1. Go to the manage endpoints page and click on the view all tab.
2. Enter uc3 in the filter by box, and hit the search button:
3. Click on the activate link next to uc3#uc3-data:

4. Select University of Chicago in the following dropdown:
5. Enter your cnetid and password in the CILogon page:
6. You should then be able to enter the `uc3#uc3-data` endpoint as a source or destination in the start transfers page.

**Transferring files**

1. Now that the UC3 endpoint has been activated, go the the transfer page and select your laptop as the first endpoint. Your system should be called something like `username#mac` or `username#pc` where `username` has been replaced with the login name for your Globus Online account.
2. Next select the UC3 endpoint as the second endpoint. Globus Online should default to your home directory.
3. Select a file on your laptop that you'd like to transfer and click on the -> arrow button to initiate the transfer.
4. You should receive an email when the transfer has been completed.
Session IV
Specific Application Examples

• Using Octave on UC3 •
• Using ROOT on UC3 •
Example Application - Octave

Table of Contents

1 Overview
2 Accessing Octave locally
3 Running Octave code
4 Building the HTCondor job
5 Submit and analyze

Overview

This section covers how to use the UC3 CVMFS system to run a real application like Octave or MatLab statistical package. For this example, we'll use Octave although Matlab can be substituted without any changes in the code. This example will go through using Matlab to generate a random matrix, doing some simple matrix operations and then calculate the eigenvectors of the matrix.

Accessing Octave locally

Since Octave is installed into CVMFS, it's not available in the normal system paths. We'll need to set up those paths so we can access Octave correctly:

```
% export PATH=$PATH:/cvmfs/uc3.uchicago.edu/groups/swift/bin
% export LD_LIBRARY_PATH=/cvmfs/uc3.uchicago.edu/groups/swift/lib64
```

Once the path is set up, we can run Octave:

```
[sthapa@uc3-cvmfs]/% octave
GNU Octave, version 3.6.4
Copyright (C) 2013 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.
Octave was configured for "x86_64-unknown-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.

For information about changes from previous versions, type 'news'.

warning: no graphical display found
warning: readline is not linked, so history control is not available
```
Running Octave code

Now let's run the Octave code. First, let's create a work directory:

```
% mkdir octave-tutorial
% cd octave-tutorial
```

You can cut and paste the following file or copy it from /opt/uc3/octave/ex1_matrix.octave:

```
A = rand(40, 40)
B = A' * A
[v, d] = eig(A)
diag(d)
```

Run the script:

```
[sthapa@uc3-cvmfs] /tmp% octave test.octave
```

Click to see full output

This should run fairly quickly.

Building the HTCondor job

Although the previous script ran quickly, suppose we needed to use a 100x100 matrix instead of a 10x10 matrix or do this a few thousand times. Since each invocation is independent of others, we can use condor to easily parallelize this and run it on UC3.

First, we'll need to create a wrapper script to setup the environment for Octave and before running it. You can cut and paste the following lines or copy the script from /opt/uc3/octave/octave-wrapper.sh:
#!/bin/bash

EXPECTED_ARGS=1

if [ $# -ne $EXPECTED_ARGS ]; then
echo "Usage: octave-wrapper.sh file.R"
exit 1
else
    PATH=$PATH:/cvmfs/uc3.uchicago.edu/groups/swift/bin
    LD_LIBRARY_PATH=/cvmfs/uc3.uchicago.edu/groups/swift/lib64
    PATH=$PATH:/cvmfs/uc3.uchicago.edu/sw/bin
    octave $1
fi

Now that we've created a wrapper, let's build a Condor submit file around it. As always, you can copy this file from /opt/uc3/octave/octave.submit:

```
universe = vanilla
log = octave.log.$(Cluster).$(Process)
error = octave.err.$(Cluster).$(Process)
output = octave.out.$(Cluster).$(Process)

# Setup Octave path, run the ex1_matrix.octave script
executable = octave-wrapper.sh
transfer_input_files = ex1_matrix.octave
arguments = ex1_matrix.octave
requirements = (HAS_CVMFS =?= TRUE)
queue 20
```

Since we're using Octave from CVMFS, we will always need to have a requirement (HAS_CVMFS =?= TRUE) that selects nodes with CVMFS installed.

## Submit and analyze

Finally, submit the job to UC3!

```
% condor_submit octave.submit
```
Application Example - ROOT

Table of Contents

1 Overview
2 Background
3 Testing ROOT on the submit host
4 Running some ROOT code
5 Accessing software anywhere using Parrot
6 Building an HTCondor Job

Overview

This application example will cover the use of the ROOT data analysis framework on UC3. In this example, we'll use Parrot in order to access CVMFS on any worker node, regardless of whether or not it is natively mounted.

Background

ROOT is a piece of software commonly used in high energy physics. We'll use a sample piece of code (shamelessly stolen from Ilija Vukotic) that prints all of the TTrees and their branches for a given ROOT file.

Testing ROOT on the submit host

For this example, we're going to use ROOT in a manner similar to a typical ATLAS job. The first thing to do is set up our working directory for the tutorial, or simply run 'tutorial root'.

```
[cnetid@uc3-sub ~]$ mkdir -p uc3-root/log; cd uc3-root
```

We'll need to run a few scripts to get the ROOT environment set up properly. This will add ROOT to our PATH and point LD_LIBRARY_PATH at the correct libraries.
Let's try running ROOT. We'll use the `-l` flag because we don't want ROOT's splash screen:

```
[cnetid@uc3-sub uc3-root]$ source environment.sh
Setting up gcc
Setting up ROOT
Setting up xRootD
[cnetid@uc3-sub uc3-root]$ root -l
*** DISPLAY not set, setting it to 10.150.25.138:0.0
root [0]
```

There are some complaints about DISPLAY, but that's alright because we don't plan to do anything requiring X11 graphics. You can quit out of root with `.q`

```
root [0] .q
```

**Running some ROOT code**

We're going to need some ROOT code, as well as a Makefile to compile it. Here is the ROOT code:

```
file: inspector.C

// This piece of code has to get tree names, tree sizes, branch names, branch sizes.
// Should be compiled.

#include <stdlib.h>
#include "Riostream.h"
#include "TROOT.h"
#include "TFile.h"
#include "TNetFile.h"
#include "TTree.h"
#include "TTreeCache.h"
```
```cpp
#include "TBranch.h"
#include "TClonesArray.h"
#include "TStopwatch.h"
#include "TKey.h"
#include "TEnv.h"
#include <iostream>
#include <fstream>
#include <sstream>
using namespace std;

class mTree{
public:
    mTree(TTree *t){
        name=t->GetName();
        entries=(long)t->GetEntries();
        totSize=t->GetZipBytes();
        leaves=t->GetListOfBranches()->GetEntriesFast();
        for (int i=0; i<leaves; i++) {
            TBranch* branch = (TBranch*)t->GetListOfBranches()->UncheckedAt(i);
            branch->GetBasketSize() << " BS: " << branch->GetBasketSize() << " size: " << branch->GetTotalSize() << " totbytes: " << branch->GetTotBytes() << endl;
            branchSizes.insert(std::pair<string,long>(branch->GetName(),branch->GetZipBytes()));
        }
    }
    string name;
    long entries;
    long totSize;
    int leaves;
    map<string,long> branchSizes;// this is value of ZIPPED SIZES collected from all the files
    void print(){
        cout<<name<<" entries":" totSize":" branchSizes.size()"<<endl;
        for(map<string,long>::iterator it = branchSizes.begin(); it != branchSizes.end(); it ++){
            cout<<it->first<<" size: "<<it->second<<endl;
        }
    }
};

int main(int argc, char **argv){
    if (argc<2) {
        cout<<"usage: inspector <filename> "<<endl;
        return 0;
    }
    vector<mTree> m_trees;
    string fn = argv[1];
    TFile *f = TFile::Open(fn.c_str());
    TIter nextkey( f->GetListOfKeys() );
    TKey *key;
    while ( (key = (TKey*)nextkey()) ) {
        TObject *obj = key->ReadObj();
    }
}
```

if ( obj->IsA()->InheritsFrom( "TTree" ) ) {
    TTree *tree = (TTree*)f->Get(obj->GetName());
    int exist=0;
    for(vector<mTree>::iterator i=m_trees.begin();i!=m_trees.end();i++)
        if (obj->GetName() == (*i).name) exist=1;
    if (!exist) m_trees.push_back(mTree(tree));
}
}
cout<<m_trees.size()<<endl;
for (vector<mTree>::iterator it = m_trees.begin(); it != m_trees.end(); it++)
    it->print();
f->Close();
Here's the Makefile:

```makefile
file: Makefile

RC := root-config
ifeq ($(shell which $(RC) 2>&1 | sed -ne "s@.*/$(RC)@$(RC)@p"),$(RC))
  MKARCH := $(wildcard $(shell $(RC) --etcdir)/Makefile.arch)
endif
ifeq ($(MKARCH),)
  include $(MKARCH)
else
  include $(ROOTSYS)/test/Makefile.arch
endif

ALIBS = $(LIBS) -lTreePlayer

#------------------------------------------------------------------------------
INSPO = inspector.$(ObjSuf)
INSPS = inspector.$(SrcSuf)
INSP = inspector$(ExeSuf)
OBJS = $(INSPO)
PROGRAMS = $(INSP)

#------------------------------------------------------------------------------

.SUFFIXES: .$(SrcSuf) .$(ObjSuf) .$(DllSuf)

all: $(PROGRAMS)

$(INSP): $(INSPO)
  $(LD) $(LDFLAGS) $^ $(ALIBS) $(OutPutOpt)$@
  $(MT_EXE)
  @echo "$@ done"

You can run "make" to create the executable inspector code.

[cnetid@uc3-sub uc3-root]$ make
```

Let's try it out. We're going to be remotely reading data from the UChicago Tier 3 XRootD filesystem.
Accessing software anywhere using Parrot

Suchandra has spoken a bit about Parrot for data access. I've written a bit of shell code to do that for you:

```
#!/bin/bash
# This script downloads parrot and sets it up to work with the ATLAS CVMFS
wget http://uc3-data.uchicago.edu/~sthapa/parrot.tar.gz
tar -xvzf parrot.tar.gz
export HTTP_PROXY="uc3-data.uchicago.edu:3128;http://uct2-grid1.uchicago.edu:3128;DIRECT"
export PARROT_HELPER="parrot/lib/libparrot_helper.so"
wget http://uc3-data.uchicago.edu/rwg-web/cern.ch.pub
./parrot/bin/parrot_run -r atlas.cern.ch:url=http://cvmfs.racf.bnl.gov:8000/opt/atlas,pubkey=cern.ch.pub,quota_limit=1000
/bin/bash -c 'source environment.sh; make; ./inspector root:///uct3-xrd.mwt2.org///atlas/uct3/data/users/lincolnb/ZZincllNp0.ntuple.root'
hostname
```

Building an HTCondor Job

Creating a job submit file for this code is pretty straightforward. The wrapper script does the bulk of the heavy lifting, we just have to make sure we are transferring the appropriate files. The requirements line is optional here, but I've included it because I'd like to see my job run on OSG.
file: root.submit

```plaintext
executable = wrapper.sh
universe = vanilla

Error = log/err.$(Cluster).$(Process)
Output = log/out.$(Cluster).$(Process)
Log = log/log.$(Cluster).$(Process)

transfer_executable = True
transfer_input_files=inspector.C,Makefile,environment.sh
when_to_transfer_output = ON_EXIT

requirements = !isUndefined(GLIDECLIENT_Name) == FALSE

queue 1
```

Let's submit the code

```plaintext
[cnetid@uc3-sub uc3-root]$ condor_submit root.submit
Submitting job(s).
1 job(s) submitted to cluster 85995.
```

We can see that it's running:

```plaintext
[cnetid@uc3-sub uc3-root]$ condor_q

-- Submitter: uc3-sub.uchicago.edu : <10.1.3.94:9618?sock=25212_0c25_14> :
uc3-sub.uchicago.edu
ID OWNER SUBMITTED RUN_TIME ST PRI SIZE CMD
85995.0 cnetid 5/20 13:16 0+00:00:05 R 0 0.0 wrapper.sh
```

This code puts all of its output on stdout, so let's check the output:

```plaintext
[cnetid@uc3-sub uc3-root]$ tail -n10 log/out.85995.0
vx_m 465260
vx_n 16759
vx_nTracks 223060
vx_px 480170
vx_py 480150
vx_pz 481488
vx_sumPt 465725
vx_x 481620
vx_y 444255
vx_z 499665
c-110-34.aglt2.org
```

Success!
Session V
Higher Level Tools: Swift and UC3

- Composing workflows
- Running in parallel