Campus Grid Application Example - Mathematica

Table of Contents

1 Overview
2 CERN Virtual Machine Filesystem
3 Running Mathematica locally
4 Creating an HTCondor job
5 Stitching the output together
6 (Optional) Creating an HTCondor DAG
7 Pretty pictures
8 Bonus Section: Parrot

Overview

This application example will show how easy it is to distribute a piece of software like Mathematica within a campus grid context using the CERN Virtual Machine Filesystem.

CERN Virtual Machine Filesystem
The CERN Virtual Machine Filesystem (CVMFS) is a read-only, HTTP-based, FUSE filesystem that leverages squid caching as well as local disk caching. At the University of Chicago, we maintain a CVMFS repository for user applications, and distribute this filesystem to member clusters in the University of Chicago Computing Cooperative framework. One such application, and the subject of this tutorial, is Wolfram Research’s Mathematica scientific computing system.

Mathematica can be found in the UC3 CVMFS system at /cvmfs/uc3.uchicago.edu/Wolfram/Mathematica. We use a campus licensing server provided by the University of Chicago’s IT Services team, though we initially ran our own license server.

**Running Mathematica locally**

We'll need to set up a working directory. Let's create a Mathematica directory as well as some additional directories we'll need later.

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

In order to grab Mathematica from CVMFS, we'll need to export our PATH appropriately.

```
[user@uc3-sub 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.

```
[user@uc3-sub 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[]
```

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".
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.

There's a lot of stuff on standard out, but we see that "mandelbrot.01.csv" was created.

You can look at this 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:
math.sh

#!/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.

mandelbrot.submit

echo executable = math.sh
echo universe = vanilla
Log = ../log/log.$(Cluster).$(Process)
Output = ../log/out.$(Cluster).$(Process)
Error = ../log/err.$(Cluster).$(Process)

WhenToTransferOutput = ON_EXIT
should_transfer_files = YES
transfer_input_files = ../mandelbrot.m
initial_dir = csv

Arguments = mandelbrot.m $(Process) $(Jobs)

requirements = HAS_CVMFS =?= True
+AccountingGroup = "group_friends.username"

Jobs = 10
queue $(Jobs)

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

```bash
stitch.submit

executable = math.sh
universe = vanilla
Log = log/log.(Cluster).$(Process)
Output = log/out.(Cluster).$(Process)
Error = log/err.(Cluster).$(Process)

WhenToTransferOutput = ON_EXIT
should_transfer_files = YES
transfer_input_files = stitch.m, csv/
Arguments = stitch.m
requirements = HAS_CVMFS =?= True
+AccountingGroup = "group_friends.username"
queue 1
```

(Optional) Creating an HTCondor DAG

>> Click here

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.

```bash
mandelbrot.dag

JOB A mandelbrot.submit
JOB B stitch.submit
PARENT A CHILD B
Retry A 3
```

Submitting DAGs is just as easy as submitting HTCondor jobs:
The DAG will fire up all of our math.sh scripts for us:

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!
Bonus Section: Parrot

It's not always possible for us to mount CVMFS on a cluster in a campus grid. One possible solution for this is to use Parrot, a tool for attaching to remote I/O systems. Parrot can speak CVMFS, provided we point it at the correct public key and HTTP proxy. I've written a script to set up and run Mathematica through Parrot, but unfortunately the performance is abysmal for anything but the most trivial of code.

```plaintext
file: version.m

$Version
100!
```

I had originally intended to do the Mandelbrot calculation through Parrot, but I was never able to get it to perform well enough. In lieu of that, I have a small script that prints the Mathematica version and a small factorial instead.
#!/bin/bash
# This script will download and run Parrot to access the UC3 CVMFS system
# I have put all of my files in the same directory
FILES_URL=https://uc3.uchicago.edu/files/

# Set up CVMFS
CVMFS_REPO="http://uc3-cvmfs.uchicago.edu/opt/uc3/"
CVMFS_KEY="uc3.uchicago.edu.pub"

# Web directory to grab CCTools from
CCTOOLS_LINK=$FILES_URL

check_release () {
    # Lame hacky way to check for version of EL
    cat /etc/redhat-release | cut -d'.' -f1 | grep 5 2>&1 > /dev/null
    if [ $? -eq 0 ]; then
        echo "redhat5"
    fi
    cat /etc/redhat-release | cut -d'.' -f1 | grep 6 2>&1 > /dev/null
    if [ $? -eq 0 ]; then
        echo "redhat6"
    fi
}

EL_RELEASE=$(check_release)
CCTOOLS_RELEASE=cctools-3.7.2-x86_64-$EL_RELEASE

echo "Downloading CCTools from $FILES_URL/$CCTOOLS_RELEASE.tar.gz"
wget --quiet $CCTOOLS_LINK/$CCTOOLS_RELEASE.tar.gz

tar -xf $CCTOOLS_RELEASE.tar.gz
rm $CCTOOLS_RELEASE.tar.gz

echo "Grabbing CVMFS key from $FILES_URL/$CVMFS_KEY"
wget --quiet $FILES_URL/$CVMFS_KEY

# Some additional stuff needed for parrot
export HTTP_PROXY="uc3-data.uchicago.edu:3128;http://uct2-grid1.uchicago.edu:3128;DIRECT"
export PARROT_HELPER="$CCTOOLS_RELEASE/lib/libparrot_helper.so"

echo "Running parrot"
./$CCTOOLS_RELEASE/bin/parrot_run -r uc3.uchicago.edu:url=http://uc3-cvmfs.uchicago.edu/opt/uc3/,pubkey=$CVMFS_KEY,
quota_limit=1000 /bin/bash -c '/cvmfs/uc3.uchicago.edu/Wolfram/Mathematica/8.0/Executables/math < version.m'

hostname