Running Inspect on the FWGrid

Samuel Payne

Winter 2007

When your Inspect job is scheduled for completion some time next week, then it is time to start thinking about using a grid. This is sometimes a painful experience; grids are difficult to deal with. But we have some scripts and helpful advice that should get you running. In this tutorial we will go over

- Grid basics
- Getting files on the grid
- Directory setup
- Generating jobs
- Submitting and monitoring jobs
- NBCR specifics

Part 1: Grid Basics

A supercomputer, or a grid, is really just a bunch of computers hooked together in such a way to make use of multiple computers at once easy. Individual computers in a grid are often no better than your desktop computer. The only advantage is that there are 500 of them. If you can get 50 of them to process your results, you finish 50 times faster.

Most grids have a single processor dedicated to scheduling node ("front node") that distributes work out to the rest of the nodes ("compute nodes"). The *modus operandi* is to write a small shell script containing your desired work (eg. inspect.exe –i Inspect.in –o Inspect.out) and then submit that script to the queue. The scheduler farms it out to a free compute node, and your work gets done. Our contribution to your sanity is that we've automated shell script creation. If this is abstract or confusing, read the appendix for a vocabulary boost. As a stern warning, everyone should read the entry for "Front node." If you don't understand it, please ask for help. You could single-handedly SNAFU everyone's runs on the grid. That would be very, very bad.

Part 2: Getting files on the grid

When you open your grid account, it's empty. Your data files, stored on some computer or server, need to be copied to the grid. This includes your spectra files, Inspect source code, database files, etc. If they were on an FTP server, you would follow these commands

>ftp bioinfo2.ucsd.edu

After typing in your username and password, you will be logged into the ftp server. Navigate your way to the proper directory, using "cd" and "dir". You can get a file from the server by typing

> get filename.txt

After the file transfer is complete, type "quit" to exit the ftp program. Alternate methods for file transfer include scp, secure copy. You can research this if you need.

Part 3: Directory setup

We have written a few python programs that will create these shell scripts (job scripts) for you. They assume a directory structure, which you will setup. When you log in to the grid, you will be in your home directory, eg. /home/username. The storage space allowed in the home directory is pretty small, so we actually work in a scratch directory. To create a scratch directory, type "mkdir /scratch/username" at a prompt. In your scratch directory create the following directories, all named exactly as follows (yes capitalization matters!): CopyFlags, Done, Inspect, jobs, mzxml, output, ResultsX.

CopyFlags - keeps track of files that are actively being copied. You don't ever need to bother with this directory.

Done - keeps track of files that have been searched. If a file has been previously searched, then ClusterSub.py will ignore it in job creation (this can be overridden with the –a option).

jobs - where you create and launch all jobs.
Inspect - where the Inspect code goes.
mzxml - where you put all mzxml files to search (or mgf for that matter).
output - where the stdout from Inspect goes for each run.
ResultsX - where final results of the Inspect program are written.

In your scratch directory you also need a few python programs: ClusterSub.py, ClusterRunInspect.py, ClusterUtil.py. GenerateScanCount.py creates a utility file, ScanCount.txt, to keep track of how many spectra are in each file.

Repeated for emphasis, you need to actually put the Inspect on the grid. So in your Inspect directory, unzip the Inspect distribution. Then type "make clean" followed by "make all". Then "touch –m *".

Part 4: Generating jobs

A job is simply a task that you want the computer to do, like run Inspect. To get the computer to do this, you create a job script and submit it to the queue. You never run Inspect on the front end. To create job scripts, use ClusterSub.py. It expects that you have MS/MS files in the mzxml directory. Secondly, it expects a zipped database (.trie and .index zipped together) in your scratch directory. Finally it expects that you have a ScanCount.txt file, which we will create. If you are in your scratch directory, the following commands will create the shell scripts for you.

```
>python GenerateScanCount.py
>cd jobs
>python ../ClusterSub.py –d /scratch/username/myDB.RS.zip  -c
/scratch/username/Common.RS.zip
```

Yes, we run this script from the jobs directory. (We just do.) You should see some output to the screen like "qsub filename.sh", for example:

```
qsub jP19-diff-Total-2mg-TiO2-Elution-a-2D19-110906-LTQ1-01.0.sh
```

Part 5: Submitting and Monitoring Jobs

Scripts are submitted to the queue by typing "qsub scriptname.sh". When its turn comes up, the script will be run on a compute node. If the script successfully finishes, then results are written to the ResultsX folder. When you ran ClusterSub, it printed out "qsub" next to the name of every job script it created. So you can copy and paste these into a prompt *en masse*. The FWGrid allows you to run 64 jobs at once. If you submit more than that, the computer ignores you.

In a perfect world, submitting your scripts would be the end of your trouble. Ours is not a perfect world. Jobs can fail for many reasons, which may or may not be your fault. You should check up on your processes. The command "qstat" prints out all the jobs currently submitted to the queue, "qw", and all those running,"r". To see only your jobs, type "qstat | grep username" To count your jobs, type "qstat | grep –c username"

Inspect is fast, but no Inspect run should finish in under 1 minute (or nothing that you should bother putting on the grid). So I recommend checking your processes within a few minutes of starting them. If they have all stopped and there is nothing in your ResultsX folder, then you have a failed run. If however, things are still running after 5 minutes, then chances are good the batch will successfully finish.

Trouble shooting. Where do we start? If something has failed, the first place to look is the output of the shell script. If you were good and submitted the shell scripts from the "jobs" directory, then it will contain each shell script along with some log-style output files, which lists every operation and the outcome.

jP19-undiff-Total-2mg-TiO2-Elution-a-2D19-110906-LTQ2-11.0.sh
jP19-undiff-Total-2mg-TiO2-Elution-a-2D19-110906-LTQ2-11.0.sh.o276118
jP19-undiff-Total-2mg-TiO2-Elution-a-2D19-110906-LTQ2-11.0.sh.po276118

I can't list all the potential errors, or how to solve them. But reading these log files is a good place to start. If you see something like "Segmentation fault (core dumped)" then you know that the C code for Inspect crashed. Possible causes for this is that the expected files did not exist (because you did not "touch" them), or that the code base is unstable (which we try not to release).

If your program is running well, but your anxiety requires an estimation of finishing time, the best way to check progress is to read the stdout of Inspect as it is running. To do this, you have to log into the node running your job. The output of qstat

278865 0.60500 jwt1pct.0. fwg.spayne r 01/15/2007 10:34:24 all.q@fwgrid-compute-11-17.local

tells you what node your on, e.g. 11-17. ssh into that node by "ssh fwg-c11-17". After some security hoops, you're in. Your job is running in a temp directory on /state/partition1. Look for something with your name, and go into that folder. The Inspect stdout has the convenient extension ".out". Typing "tail filename.out" will show the progress of the program, eg. "11.5% complete".

Part 6: NBCR Notes

The NBCR is a different supercomputer grid, and frankly it is more difficult to deal with. The first problem comes at the login. We have grid accounts on kryptonite.nbcr.net. But you cannot directly log in to kryptonite. You must first log into puzzle.nbcr.net, and then into kryptonite. This extra level of indirection is a security feature.

Each of the two logins requires a password protected public/private key pair. Let's take it one step at a time. To gain a login for puzzle, generate a key pair using

a. 'ssh-keygen -t dsa' on Linux or Cygwin on Windows
or
b. PuTTYgen on Windows

The password used to protect the keys should be at least 8 characters. Then send the public key file to Chris Misleh (cmisleh@sdsc.edu). He will authorize your account, after which you can log into puzzle. Once you have logged into to puzzle, the first thing that you have to do is make the key pair for logging into kryptonite. So, type

> ssh-keygen -p

It will ask you for a file holding the key. There is a default, so you can hit enter. Then it will prompt you for the password. This should be a different password from the first (write it down so you don't forget!). Then after that you can ssh from puzzle to kryptonite. Then once on kryptonite you can transfer files and run stuff like you would on the FWGrid.

Another difference between the NBCR and the FWGrid is that the scratch directory has a different name. Here it is "/nas3/username".

Appendix: terminology

Job – A single program running on the grid. Thus if you are using 50 computers, you have 50 jobs running.

Submit a job – tell the grid that you have a job for it to do. Note this is different than just entering the command and running the job yourself.

Queue – a list of jobs that have been submitted and are waiting for a free node. The front end keeps this list.

Node - each one of the computers in the massive super computer network.

Front end - On the FWGrid, there is a single node set up as the main interface with the world. It job is really like a scheduler or manager directing job submission. The biggest taboo in any grid environment is to run your job on the front end (as opposed to submitting the job and letting a compute node run it). You may get permanently kicked off for such an offense. You will definitely ruin people's day by this offense.

Compute nodes - All other nodes besides the front end. These just run processes day and night as the queue directs them.

Zipped file – a compressed version of one or more files. To create a zipped database file for the ClusterSub.py program do this
> zip myDB.zip myDB.*

where your current working directory contains the two files myDB.trie and myDB.index. This command will wrap and compress both of those files into a single new file, myDB.zip

stdout – standard output of a program. Typically progress statements, or debugging verbiage spewed out by the program.