GEL 2.0 Tutorial

Francis Tang
Bioinformatics Institute
A-STAR, Singapore

June 21, 2005

`http://wildfire.bii.a-star.edu.sg`

Revision: 1.5 – Date: 2005/06/21 07:44:37 Last Author: francis

1 Introduction

The advent of commodity hardware in HPC has resulted in a situation where researchers often find they have several clusters at their disposal. Often these clusters have subtle differences, e.g. different operating systems and different queue manager software. For example, one cluster might use SGE as its queue manager, and another might use PBS/Torque as its queue manager.

Running jobs on a cluster involves writing a submit script for the queue manager. Typically, submit scripts are not portable between queue managers. For example, an SGE submit script might not run on PBS/Torque.

Furthermore, though queue managers such as PBS/Torque, SGE and LSF provide command-line "submit" programs which can be used to submit single jobs with dependencies, or even job arrays, using dependencies in a useful manner often requires extensive script programming, or otherwise.

GEL allows you to write scripts which are independent of queue schedulers, and which specify dependencies between jobs implicitly in the script syntax. Different interpretor instances will submit your jobs to different queue managers, with the correct job dependencies.

This tutorial guides you through the development of a simple script to make an animated GIF image out of a collection of JPEG images. The reader can find further information about GEL scripting in the *GEL Reference Guide* available from `http://wildfire.bii.a-star.edu.sg`.

2 Obtaining and Installing GEL

To use GEL, you will need a UNIX-like machine with GNU utilities. If you would like to run jobs on the compute nodes, your cluster must use either PBS (or Torque), SGE or LSF, and have a shared file system.

You will need to download and install GEL. GEL is available from `http://wildfire.bii.a-star.edu.sg`.

2.1 Installing GEL

Only Unix-like environments are supported by GEL. (GEL has been tested on i386-linux, ia64-linux, alpha-OSF and sparc-SunOS.)

Download `gel-2.0.tar.gz` from `wildfire.bii.a-star.edu.sg`. GEL is available without fee under an A-STAR license which allows for non-profit research use. The precise terms of the license are available from `wildfire.bii.a-star.edu.sg` website.[1] Extract this archive into your home directory and run the install script `mk-wrapper.sh`:

```
% zcat gel-2.0.tar.gz | tar xf -
% cd gel-2.0
% ./mk-wrapper.sh ‘pwd‘
```

The last step is important.

Add `$HOME/gel-2.0` to your PATH. To check that GEL has been installed, try running `gel` on the command line:

```
% gel
GEL (Version: 2.0)
(c) 2004, 2005, Bioinformatics Institute, A-STAR, Singapore
...
```

If you see the banner above (plus help on command-line options), then you have successfully installed GEL.

2.1.1 Uninstalling GEL

To uninstall GEL, just remove the program directory, e.g. remove `$HOME/gel-2.0` in the example above.

[1] Please contact BII for information about other licenses.

3 A simple problem: building an animated GIF from a collection of JPEG images

We consider a simple problem. Suppose we have a directory of JPEG images and we would like to resize each image before joining the directory together into an animated GIF. The resizing step is embarassingly parallel because each image can be resized independently of the others. However, the animation step is dependent on the resizing step, since it can only start once all the images have been resized.

Solving this simple problem using the command-line "submit" programs (i.e. `qsub` and `bsub`) provided by PBS/Torque, SGE or LSF would require programming a script using bash, or otherwise. An example bash script for LSF is shown in Fig. 1; note that the main bulk of this script involves catching the job id from submitting the resize jobs, and then assembling the correct dependency expression for the animate job. In contrast, this script expressed as a GEL script is displayed in Fig. 2; note that the dependencies between jobs is implicit in the structure of the script.

The files relating to this tutorial can be found in the `docs/tutorial/files` directory within the GEL 2.0 install directory. The script uses the `convert` command from the ImageMagick package, and so you will also need to install that to be able to run the script.

3.1 Starting simple, resizing one image

The ImageMagick command `convert` can resize an image to fit within a 320×240 box using the following options

```
convert -geometry 320x240 input.jpg output.jpg
```

This will read `input.jpg` and write `output.jpg`.

Suppose `input.jpg` is in directory `input_one` relative to the current working directory. A GEL script to run the command above looks like this:

```
resize := {
    exec = "convert" ;
    args = "-geometry", "320x240", "input.jpg", "output.jpg" ;
    ipdir = "input_one"
}

resize
```

This script first defines a *job template* called `resize` such that running the job amounts to running the command `convert` with the arguments as listed in the `args` attribute. The `ipdir` attribute tells GEL where to find the input files.

To run this script, save the script as `one.gel` and run the following command

```
#!/bin/bash

jobs=""
outfiles=""
for f in *.jpg; do
    out=${f%.jpg}out.jpg
    outfiles="$outfiles $out"
    jid=‘echo convert -geometry 320x240 $f $out \
        | bsub -H \
        | sed ’s/[^0-9]//g’‘
    jobs="$jobs $jid"
    echo $jid
done

deps=""
for jid in $jobs; do
    if [ -z "$deps" ]; then
        deps="ended($jid)"
    else
        deps="$deps && ended($jid)"
    fi
done

echo convert -delay 20 $outfiles anim.gif | bsub -w "$deps"
bresume $jobs
```

Figure 1: LSF script to resize a directory of JPEG images and join them into an animated GIF. Note that this script must be modified for submitting to SGE or PBS since the submit program command-line options are different; especially the dependency condition.

```
init := {
    exec = "date" ;
    ipdir = "input"
}

final := {
    exec = "date" ;
    cmdir = "results"
}

resize(f) := {
    exec = "convert";
    args = "-geometry", "320x240",
           $f,
           $f %".jpg" ."out.jpg"
}

animate := {
    exec = "animate.sh" ;
    dir = "bin"
}

init ;                      # copy files
pforeach file of "*.jpg" do
    resize($file)           # resize images in parallel
endpforeach ;
animate ;                   # make animation
final                       # copy files
```

Figure 2: GEL script to resize and then animate a directory of JPEG images.

```
gel -f one.gel
```

GEL will create a temporary working directory of the form `Jtmp-`*nnnn* where *nnnn* is a sequence of digits. Next, it will copy the contents of `input_one` into the working directory, and execute the `convert` command. When GEL has terminated, `ouput.jpg` can be found in this temporary working directory.

We can modify this script so that the output is stored in `results_one`, by adding the `cmdir` attribute to the `resize` job:

```
resize := {
    exec = "convert" ;
    args = "-geometry", "320x240", "input.jpg", "output.jpg" ;
    ipdir = "input_one" ;
    cmdir = "results_one"
}

resize
```

Save this file as `one1.gel`. Make sure `results_one` directory exist and is empty, and run GEL as before:

```
gel -f one1.gel
```

When GEL terminates, you can inspect `results_one` and find the output file in there.

3.1.1 Running GEL on a cluster

We can tell GEL to run the `convert` command on the compute node of a cluster by specifying a queue manager option. If your cluster uses PBS/Torque, you can run

```
gel --pbs -f one1.gel
```

In this case, GEL will submit the individual jobs to the compute nodes using the `qsub` command with the correct dependency expressions. GEL will catch the job ids and calculate the dependencies for you. Similarly, if your cluster uses SGE, you can run

```
gel --sge -f one1.gel
```

and if it uses LSF, you can run

```
gel --lsf -f one1.gel
```

The behaviour of the GEL script is exactly the same as before.

Congratulations! You have written a submit script that works on PBS/Torque, SGE and LSF without modification.

3.2 Exploiting parallelism

Suppose you have a collection of JPEG images in directory `input`, and you want to resize all of them in parallel. The following GEL script can do this for you:

```
init := {
    exec = "date" ;
    ipdir = "input"
}

resize(f) := {
    exec = "convert";
    args = "-geometry", "320x240",
           $f,
           $f %".jpg" ."out.jpg"
}

init ;
pforeach file of "*.jpg" do
    resize($file)
endpforeach
```

We have now introduced two job templates: `init` and `resize`. The `init` job simply forces the files from `input` to be copied into the working directory, it runs the command `date` which does not modify any files in the working directory.

The `resize` job is now parameterised by parameter `f`. For the command line arguments, the input file is now `$f`, and the output file is `$f %".jpg" ."out.jpg"` which is the file name `$f` with the `.jpg` extension substituted with `out.jpg`. For example, `input.jpg` would become `inputout.jpg`.

The `resize` job is invoked inside a `pforeach` construct. This construct first finds all files matching `*.jpg`, and for each such file, it will execute `resize($file)` where `$file` is replaced by the file name.

Save this script as `many.gel` and execute it as before, making sure you have several JPEG files in the `input` directory. As before, the output images will be in the temporary working directory.

3.3 Bringing it all together

You can take several images and join them together into an animated GIF using the `convert` command. The syntax is

```
convert -delay 20 img1.jpg img2.jpg img3.jpg anim.gif
```

where `-delay 20` means use a delay of 20/100 seconds between images, and `img{1,2,3}.jpg` are the three frames.

Let us write a wrapper script called `animate.sh` and save it in a directory called `bin`. The script is as follows:

```
#!/bin/sh

convert -delay 20 *out.jpg anim.gif
```

This simple shell script joins all files matching `*out.jpg` into one animated GIF file called `anim.gif`.

Next we modify the previous GEL script so that we run the `animate.sh` script after all the images have been resized. Figure 2 shows the modified GEL script. First we add a new job template called `animate` whose `dir` attribute specifies that the `animate.sh` script can be found in the directory `bin`. We then use the sequential composition operator ";" to compose the `animate` job with the `pforeach` loop. We also add another job called `final` which is analogous to `init` and copies the results into directory `results`.

3.4 Running the example script

The `docs/tutorial/files` directory within the GEL 2.0 install directory contains the script displayed in Fig. 2, and scripts `one.gel`, `one1.gel` and `many.gel`. To experiment with the examples, you should copy the whole `files` directory to a writable, shared directory on the cluster (i.e. a directory which all compute nodes can write to).

First run the `setup.sh` script to: (1) copy the input JPEG images into the `input` and `input_one` directories, and (2) make empty `results` and `results_one` directories.

To run the script in Fig. 2, invoke the GEL interpretor as follows:

1. `gel -f make_anim.gel`
   This will run the GEL script using the local interpretor, i.e. run the jobs on the same machine that runs the intepretor, using a default value of 4 concurrent processes.

2. `gel --nproc=8 -f make_anim.gel`
   This is the same as above except GEL will run up to 8 concurrent processes. Since the `convert` processes terminate very quickly, this is useful even if you have fewer than 8 cpus in your machine since it reduces the amount of idle time while the interpretor pauses before polling for job completion.

3. `gel --sge -f make_anim.gel`
   This will run the script by submitting the jobs through SGE using SGE's `qsub` command.

4. `gel --pbs -f make_anim.gel`
   As above except using PBS/Torque.

5. `gel --lsf -f make_anim.gel`
   As above except using LSF and `bsub`.

The resulting file will be called `anim.gif` in the `results` directory.

The scripts `one.gel`, `one1.gel` and `many.gel` can be run similarly.