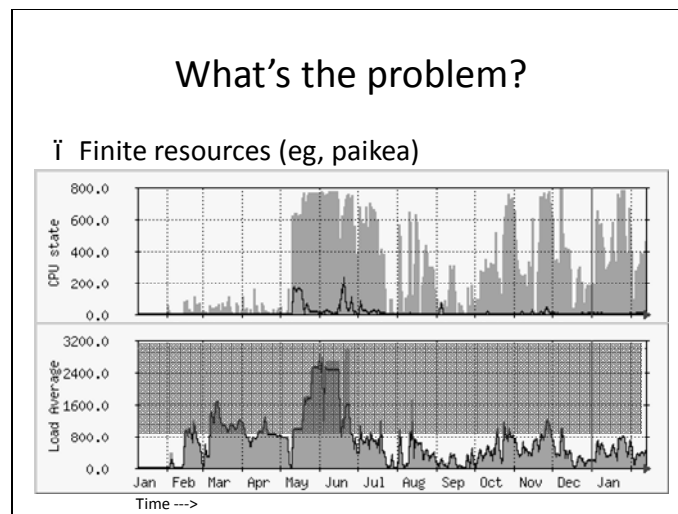


Slide 1

Grid Computing

Department of Statistics
Staff Retreat February 2008

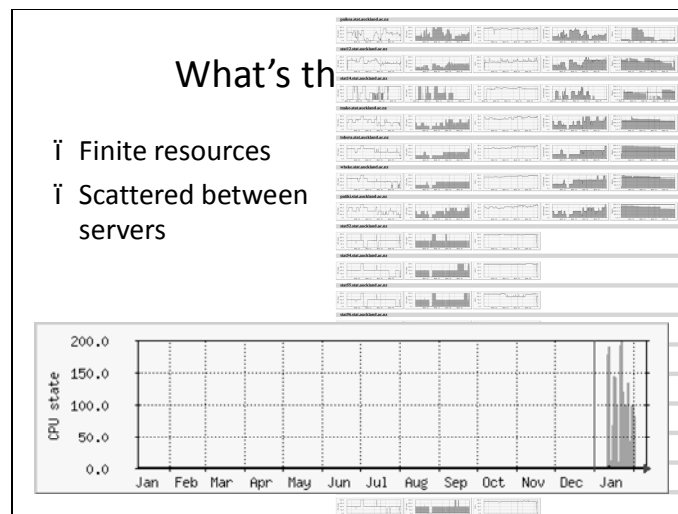
This is a talk I gave to the Department in February 2008. I followed from a talk by Werner regarding the Department's future computing resources.



Both of these graphs show 2007's usage of one of our servers, paikea.

The top graph shows how much computing power paikea has available (8 CPUs, 800%). Green is used, white is unused.

The bottom graph shows demand. Note the scale on the graphs. In May and June people were demanding 32 CPUs worth of computing power. This just brought about contention, and everyone's tasks ran slower.

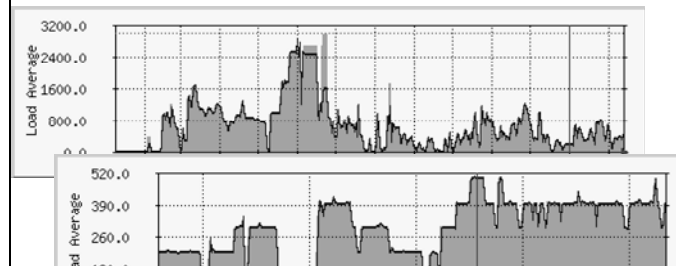


The second problem is that resources are scattered through the Department. These are graphs, just like on the previous page, but showing across all the machines available for people to run jobs on.

When a new machine becomes available, it takes a while for people to notice it. See here how one machine was unused until January.

What's the problem?

- Finite resources
- Scattered between servers
- Uneven demand



The third problem is that demand is uneven. There are the peaks of contention and over-subscription, and the troughs of idle time.

Our Department's Resources

- 52 CPUs, in 17 servers and desktops
- 160 GHz total processing power
(28x 3.4 GHz, 8x 2.8 GHz AMD64, 18x 2.4 GHz)
- 106 GB total memory
(64GB, 4.6GB, 3x 4GB, 3.4GB, 2.4GB, 10x 2GB)
- Let's take a look ...

Here's what the Department has available, scattered around the place.

Slide 6

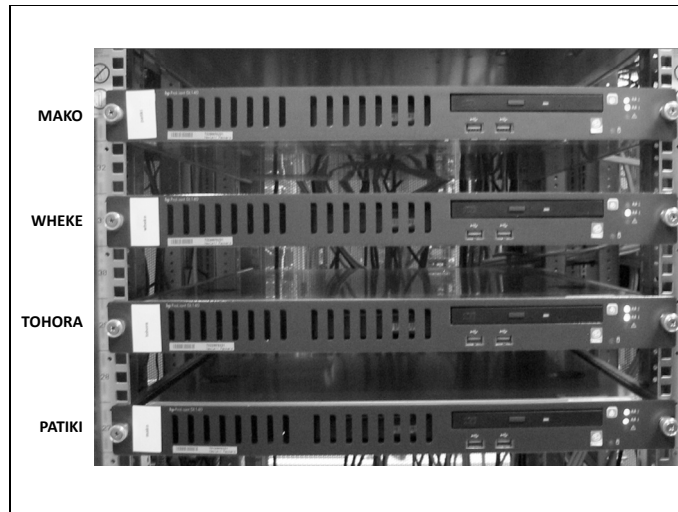
İ Basement
Server
Room

Our Racks:





Slide 8




Our Department's Resources


- 52 CPUs, in 17 servers and desktops
- 160 GHz total processing power
(28x 3.4 GHz, 8x 2.8 GHz AMD64, 18x 2.4 GHz)
- 106 GB total memory
(64GB, 4.6GB, 3x 4GB, 3.4GB, 2.4GB, 10x 2GB)
- How can we distribute this fairly?

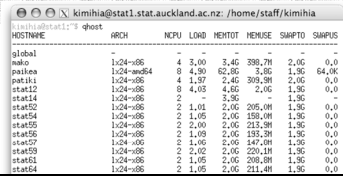
Now you know what is being fought over. How can we distribute this fairly?

Here comes the Grid!




☺ Sun Grid Engine





The solution we have installed is the Sun Grid Engine. There is a logo for it, but that’s the fanciest graphics you’ll get. In operation it looks more like the above terminal.



HOSTNAME	ARCH	NCPU	LOAD	MEMTOT	MEMUSE	SWPPTD	SWPUS
global	-	-	-	-	-	-	-
halo	i24-000	4	3.00	3.40	398.7M	2.00	0.0
patika	i24-000	0	4.90	62.00	3.00	1.00	64.0K
patiki	i24-000	4	1.37	2.40	309.9M	2.00	0.0
stat12	i24-000	0	4.03	4.40	2.00	1.00	0.0
stat14	i24-000	2	-	3.90	-	1.00	-
stat52	i24-000	2	1.01	2.00	205.0M	1.00	0.0
stat54	i24-000	2	1.05	2.00	158.0M	1.00	0.0
stat55	i24-000	2	2.00	2.00	213.9M	1.00	0.0
stat56	i24-000	2	1.09	2.00	155.3M	1.00	0.0
stat57	i24-000	2	1.00	2.00	147.0M	1.00	0.0
stat59	i24-000	2	2.00	2.00	220.1M	1.00	0.0
stat61	i24-000	2	1.05	2.00	208.0M	1.00	0.0
stat64	i24-000	2	1.05	2.00	211.4M	1.00	0.0

How can the Grid help me?

Grid Engine is helpful in three elementary ways. It can:

- optimally place computing tasks and **balance the load** on a set of networked computers
- allow users to generate and **queue more computing tasks** than can be run at the moment
- ensure that tasks are executed with respect to priority and to providing all users with a **fair share** of access over time

[Source:

<http://gridengine.sunsource.net/GEDomainFAQ.html>]

Lots of words cribbed from a good explanation.

For the admins ...

- Much easier to manage the servers:
 - Add / remove servers due to maintenance
 - Jobs restarted or moved
 - Sensible load is maintained
- Result:



The grid automatically solves a lot of the problems that Werner and I normally have to fix by hand.

Getting started

- Jobs are submitted to the Grid using **qsub**. Jobs are shell scripts containing commands to be run.
- If you would normally run your job by typing **./runjob**, (or worse **nohup nice ./runjob &**) you can submit it to the Grid and have it run by typing:
qsub -cwd ./runjob
- Jobs can be submitted while logged on to any submit host: stat1, stat12, or paikea.

[Source: <http://www.stat.auckland.ac.nz/~kimihia/sun-grid#submitting>]

Visit my website. It has all these words on that you can read at your leisure.

A simple example using R

• A very simple piece of R code (simple.R):

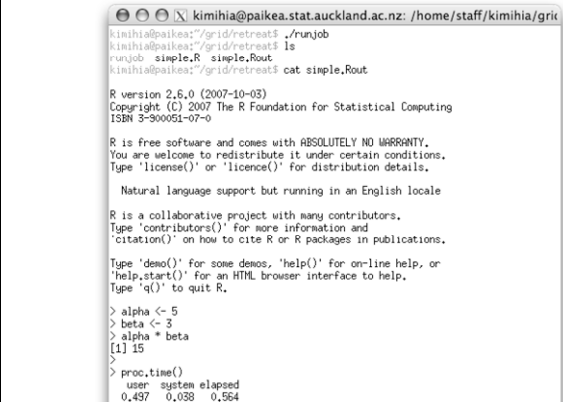
```
alpha <- 5  
beta <- 3  
alpha * beta
```

• Gets run like so (./runjob):

```
#!/bin/sh  
  
R CMD BATCH simple.R
```

Your job can be a lot more complicated than this.

In action!!!



```
kimihi@paika: /grid/retreat$ ./runjob
kimihi@paika: /grid/retreat$ ls
runjob  simple.R  simple.Rout
kimihi@paika: /grid/retreat$ cat simple.Rout

R version 2.6.0 (2007-10-03)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> alpha <- 5
> beta <- 3
> alpha * beta
[1] 15
>
> proc.time()
      user  system elapsed 
0.497    0.038    0.564
```

Here the job runs, demonstrating the usual method.

In action!!!

```

kimihia@paikaea.stat.auckland.ac.nz: /home/
10/retreat$ ./runjob
10/retreat$ ls
simple.Rout
10/retreat$ cat simple.Rout

2007-10-03)
The R Foundation for Statistical Computing

and comes with ABSOLUTELY NO WARRANTY.
redistribute it under certain conditions.
Type 'licence()' for distribution details.

support but running in an English locale

ve project with many contributors.
Type 'contributors()' for more information and
Type 'citation()' for how to cite R or R packages in publications.

some demos, 'help()' for on-line help, or
an HTML browser interface to help.
R.

elapsed
0.564

kimihia@stat1.stat.auckland.ac.nz: /home/staff/kimihia
kimihia@stat1:~/grid/retreat$ qsub -cwd ./runjob
Your job 772 ("runjob") has been submitted
kimihia@stat1:~/grid/retreat$ ls
runjob runjob.o772 runjob.o772 simple.R simple.Rout
kimihia@stat1:~/grid/retreat$ cat simple.Rout

R version 2.6.0 (2007-10-03)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
Type 'citation()' for how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> alpha <- 5
> beta <- 3
> alpha * beta
[1] 15
>
> proc.time()
user system elapsed
0.000 0.000 0.000
  
```

Here the job runs, but this time using the Sun Grid Engine. We don't know which machine it ran on, but the end result is the same. Just wait for the output or an email.

Note here that we are told our job number is 772. We can use this for tracking it.

Where's my job?

• **qstat** and **qstat -j**

[illegible]

If you need to keep an eye on your job, these are the tools to use.

Qstat = tells you what jobs are running and which are queued.

Note here the job states:

r (lowercase) = running

R (uppercase) = being rerun (power cut?)

q = job is queued, waiting for a slot

h = job being held for some reason

E (uppercase) = error. Use the next command for more information:

Qstat -j = tells you lots and lots of information about a specific job. Use the job number you were given on the previous page.

Ls = look at the files in the current directory or your home directory.

Improvements to runjob

```
#!/bin/sh
#$ -S /bin/sh
# run in current directory
#$ -cwd
# mail at end
#$ -m e
# merge output
#$ -j y
R CMD BATCH simple.R simple.Rout$JOB_ID
```

First, let's add an output filename. We'll also add the JOB_ID to the filename, to keep separate job runs in separate files.

Because \$ is a bashism I'll force use of bash.

Run in the current directory, where my dataset is.

Mail at end.

Merge output (error and standard output, both normally empty for R CMD)

Basic options

- Run in the current directory:
 - `cwd`
- Email me when it completes:
 - `me`
- Merge the output and error message files:
 - `joy`
- Wait for job to complete:
 - `sync y`

Other useful options!

- Name your job:
 ñ N timmy
- Wait for free memory:
 ñ l mem_free=2G
- Run the same thing multiple times:
 ñ t 1-20
- Wait for a previous job to complete:
 ñ hold_jid 380

Options you'll likely never use

- Queues!
 ñ q beagle.q
- Projects!
 ñ P Im2007
- Complex resources! (eg, consume one blarg)
 ñ l blarg=1
- Parallel environments! (eg, 3 orchestra slots)
 ñ pe orchestra 3

Here are some ideas for slightly more complicated usage. A grid admin (Werner, Stephen) will need to set each of these up for you.

Queues are useful for jumping over other users or being restricted to certain hosts.

Projects are good for keeping track of how much CPU time you're using.

Complex resources, eg, Matlab licenses. (Can also be restricted via quotas.)

Parallel environments, for keeping jobs on the same host (eg, in the case of a mutex) or having only a certain number running grid-wide.

Getting the most out of the Grid

- Limitations:
 - One thread per processor/slot
- Other improvements:
 - Checkpointing

For the best performance, rewrite your code!

These are not grid-specific computer science problems.

An example in parallelism

- What is the mean age of 14 people?
- 26, 30, 33, 35, 36, 36, 38, 39, 40, 41, 51, 51, 63, 67
- Easy solution:
 $(26 + 30 + 33 + \dots + 67) / 14$
- 13 summing operations
- 1 division operation

Typical order of operations

$$\text{I } ((((((((((26 + 30) + 33) + 35) + 36) + 36) + 38) + 39) + 40) + 41) + 51) + 51) + 63) + 67) / 14$$

$$\text{I } (26 + 30)$$

$$\text{I } + 33$$

$$\text{I } + 35$$

$$\text{I } + 36$$

$$\text{I } + 36$$

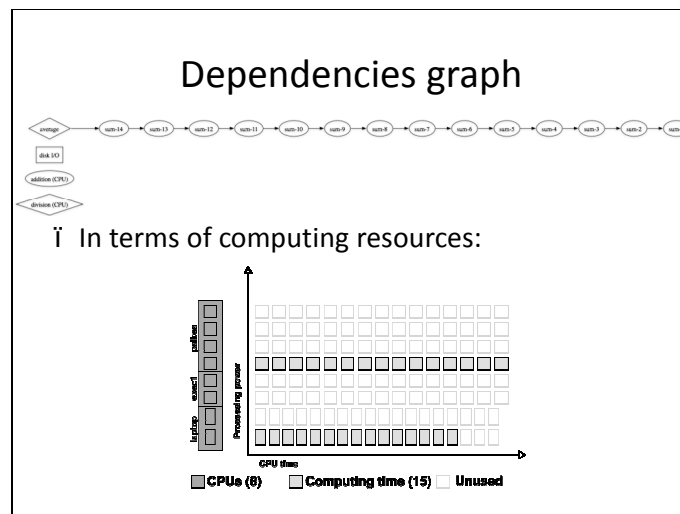
$$\text{I } \dots$$

$$\text{I } + 67$$

$$\text{I } / 14$$

Usually we work left to right, starting at the inside, sum that up, then move on.

Here's a bit of a tree layout of how the operations are done, adding, adding, adding, until finally you do the division.



Each job is dependent on the previous calculation. That's why the string of dependencies is so long.

This graph shows the job being run twice, once on a slow machine, once on a fast machine. Each box is a mathematical operation. Note how the faster machine completes earlier.

MATHEMATICAL JARGON

- Addition is commutative!
- Also associative!
- How can we make the most of this?

Reordered operations

• $((26 + 30) + (33 + 35)) + ((36 + 36) + (38 + 39)) +$

• $((40 + 41) + (51 + 51)) + ((63 + 67) + 0)$

• $/ 14$

• 14 summing operations

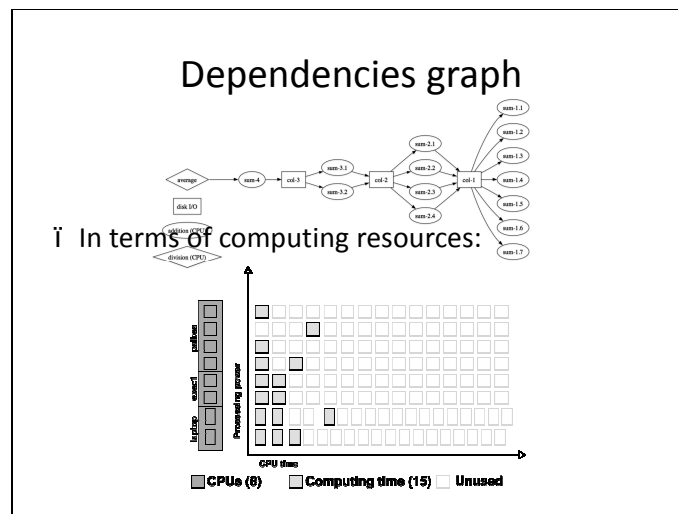
• note extra 0, added for ease of coding

• 1 division operation

Extra 0 is because I hacked this up in bash.

The actual algorithm for planning execution of the problem is to sum each pair of numbers, and then in the next step, sum each pair again, hence why we always need an even number of numbers.

However, this does not matter as it runs in parallel with other operations.



This dependencies graph looks a bit better. All the dependencies are reordered, and some can be run in parallel. Each operation (oval) in the top graph matches a square in the lower graph. Rightmost circles have to be completed first; they match the leftmost squares.

In terms of computing resources, we still have the same number of computations, but they are spread around the grid semi-randomly, and due to parallelism complete a lot sooner.

Want to see it in action?

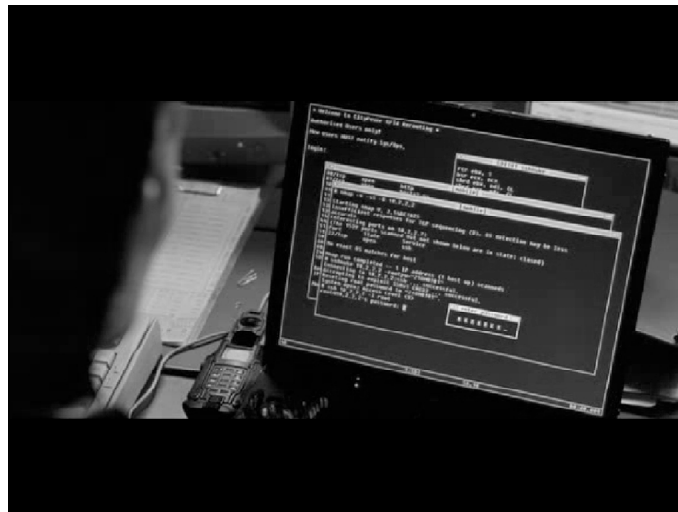
```
kimihia@stat1.stat.auckland.ac.nz: /home/staff/kimihia/grid/
kimihia@stat1:~/grid/average$ ls
ages  average  average-j  collect  fast_divide  fast_sum  sum-step
kimihia@stat1:~/grid/average$ cat ages
26
30
33
35
36
36
38
39
39
40
41
51
51
63
67
kimihia@stat1:~/grid/average$ ./average-j ages mean-age
Your job-array 774,1-7:1 ("sum-1") has been submitted
Your job 775 ("col-1") has been submitted
Your job-array 776,1-4:1 ("sum-2") has been submitted
Your job 777 ("col-2") has been submitted
Your job-array 778,1-2:1 ("sum-3") has been submitted
Your job 779 ("col-3") has been submitted
Your job 780 ("average") has been submitted
kimihia@stat1:~/grid/average$
```

You can see our data file, one age per line, and the scripts required.

We submit, and jobs are added to the queue.

Not exciting enough?

Slide 30



Here, have a picture of Trinity hacking something in The Matrix Reloaded.

GRID COMPUTING STATUS						
job-ID	prior	name	user	state	submit/start at	queue
781	0.00000	sum-1	kimihia	qw	02/14/2008 12:39:04	
782	0.00000	col-1	kimihia	hqw	02/14/2008 12:39:04	
783	0.00000	sum-2	kimihia	hqw	02/14/2008 12:39:04	
784	0.00000	col-2	kimihia	hqw	02/14/2008 12:39:04	
785	0.00000	sum-3	kimihia	hqw	02/14/2008 12:39:04	
786	0.00000	col-3	kimihia	hqw	02/14/2008 12:39:04	
787	0.00000	average	kimihia	hqw	02/14/2008 12:39:04	

Jobs sitting in the queue waiting.

I'm running 'qstat' repeatedly to show all my jobs.

GRID COMPUTING STATUS							
job-ID	prior	name	user	state	submit/start at	queue	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@patiki.stat.a	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@patiki.stat.a	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@stat52.stat.a	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@stat59.stat.a	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@stat64.stat.a	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@stat77.stat.a	
781	0.55500	sum-1	kimihia	r	02/14/2008 12:39:13	all.q@stat52.stat.a	
782	0.00000	col-1	kimihia	hqw	02/14/2008 12:39:04		
783	0.00000	sum-2	kimihia	hqw	02/14/2008 12:39:04		
784	0.00000	col-2	kimihia	hqw	02/14/2008 12:39:04		
785	0.00000	sum-3	kimihia	hqw	02/14/2008 12:39:04		
786	0.00000	col-3	kimihia	hqw	02/14/2008 12:39:04		
787	0.00000	average	kimihia	hqw	02/14/2008 12:39:04		

First jobs running, remaining jobs queued.

GRID COMPUTING STATUS							
job-ID	prior	name	user	state	submit/start at	queue	
783	0.55500	sum-2	kimihia	r	02/14/2008 12:40:43	all.q@patiki.stat.a	
783	0.55500	sum-2	kimihia	r	02/14/2008 12:40:43	all.q@stat52.stat.a	
783	0.55500	sum-2	kimihia	r	02/14/2008 12:40:43	all.q@stat59.stat.a	
783	0.55500	sum-2	kimihia	r	02/14/2008 12:40:43	all.q@stat77.stat.a	
784	0.00000	col-2	kimihia	hqw	02/14/2008 12:39:04		
785	0.00000	sum-3	kimihia	hqw	02/14/2008 12:39:04		
786	0.00000	col-3	kimihia	hqw	02/14/2008 12:39:04		
787	0.00000	average	kimihia	hqw	02/14/2008 12:39:04		

Some have disappeared because they are completed, less jobs waiting.

(This is the output of repeatedly running 'qstat'.)

GRID COMPUTING STATUS						
job-ID	prior	name	user	state	submit/start at	queue
787	0.00000	average	kimihia	qw	02/14/2008 12:39:04	

Final job running.

Slide 35

GRID COMPUTING STATUS						
job-ID	prior	name	user	state	submit/start at	queue
PROCESSING COMPLETED!!!						

All done!



The script rounds down from 41.8, but for purposes of making a Hitchhiker's Guide to the Galaxy joke I'm going to round it up.

This result is the same result as the most intensive computational problem of all time.

```
JOB OUTPUT AND LOGS

==> sum-1.0781.1 <==
Thu Feb 14 12:40:13 NZDT 2008
stat77.stat.auckland.ac.nz

==> sum-1.0781.2 <==
Thu Feb 14 12:40:13 NZDT 2008
stat59.stat.auckland.ac.nz

==> sum-1.0781.3 <==
Thu Feb 14 12:40:13 NZDT 2008
stat52.stat.auckland.ac.nz

==> sum-1.0781.4 <==
Thu Feb 14 12:40:13 NZDT 2008
patiki.stat.auckland.ac.nz

==> sum-1.0781.5 <==
Thu Feb 14 12:40:13 NZDT 2008
stat64.stat.auckland.ac.nz

==> sum-1.0781.6 <==
Thu Feb 14 12:40:13 NZDT 2008
patiki.stat.auckland.ac.nz

==> sum-1.0781.7 <==
Thu Feb 14 12:40:13 NZDT 2008
stat52.stat.auckland.ac.nz

==> col-1.0782 <==
col-1.1
col-1.2
col-1.3
col-1.4
col-1.5
col-1.6
col-1.7
collected 8 numbers.
Thu Feb 14 12:40:28 NZDT 2008
stat59.stat.auckland.ac.nz
```

This is a little hard to read, but left as an appendix for the reader. Note the time when jobs were run, and on which hosts they were completed.

As an aside, each mathematical operation took one minute, due to a “sleep 60” in each script.

JOB OUTPUT AND LOGS

```
==> sum-2.o783.1 <==  
Thu Feb 14 12:41:43 NZDT 2008  
stat59.stat.auckland.ac.nz
```

```
==> sum-2.o783.2 <==  
Thu Feb 14 12:41:43 NZDT 2008  
patiki.stat.auckland.ac.nz
```

```
==> sum-2.o783.3 <==  
Thu Feb 14 12:41:43 NZDT 2008  
stat52.stat.auckland.ac.nz
```

```
==> sum-2.o783.4 <==  
Thu Feb 14 12:41:43 NZDT 2008  
stat77.stat.auckland.ac.nz
```

```
==> col-2.o784 <==  
Found files to collect:  
col-2.1  
col-2.2  
col-2.3  
col-2.4  
Collected 4 numbers.  
Thu Feb 14 12:41:58 NZDT 2008  
stat52.stat.auckland.ac.nz
```

JOB OUTPUT AND LOGS

```
==> sum-3.o785.1 <==  
Thu Feb 14 12:43:13 NZDT 2008  
patiki.stat.auckland.ac.nz  
  
==> sum-3.o785.2 <==  
Thu Feb 14 12:43:13 NZDT 2008  
stat52.stat.auckland.ac.nz  
  
==> col-3.o786 <==  
Found files to collect:  
col-3.1  
col-3.2  
Collected 2 numbers.  
Thu Feb 14 12:43:28 NZDT 2008  
stat59.stat.auckland.ac.nz  
  
==> average.o787 <==  
Final sum required  
Thu Feb 14 12:45:43 NZDT 2008  
patiki.stat.auckland.ac.nz
```

Jobs submitted at 12:40, final operation completed at 12:45. 15x 1 minute operations were completed in five minutes, due to parallelism.

Summary & Questions

```

kimihia@paika.stat.auckland.ac.nz: /home/staff/kimihia/gri
kimihia@paika:~$ cd /tmp
kimihia@paika:~/tmp$ ls
runjob  simple.R  simple.Rout
kimihia@paika:~/tmp$ cat simple.Rout
R version 2.6.0 (2007-10-03)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> alpha <- 5
> beta <- 3
> alpha * beta
[1] 15
kimihia@paika:~/tmp$

```

```

kimihia@stat1.stat.auckland.ac.nz: /home/staff/kimihia/grid/
kimihia@stat1:~$ cd /tmp
kimihia@stat1:~/tmp$ ls
runjob  runjob.o772  runjob.o772  simple.R  simple.Rout
kimihia@stat1:~/tmp$ cat simple.Rout
R version 2.6.0 (2007-10-03)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> alpha <- 5
> beta <- 3
> alpha * beta
[1] 15
kimihia@stat1:~/tmp$

```

Hardly any difference, correct?

On the right we don't know where it ran, but we know it did run, and without stepping on anybody's toes.

Questions:

-Q (Ross Ihaka): Which jobs are better suited to parallelism? (Jobs with large data sets do not lend themselves to this sort of parallelism due to I/O overheads.)

A: Most of the jobs being used here are CPU intensive. The grid copies your script to /tmp on the local machine on which it runs. You could copy your data file across as well at the start of the job, thus all your later I/O is local.

-Q (Chris Wild): Can I make sure I always run on the fastest machine?

A: The grid finds the machine with the least load to run jobs on. If you pile all jobs onto one host, then that host will slow down and become the slowest overall. Submit it through the grid and some days you'll get the fast host, and some days you'll get the slow host, and it is better in the long run. Also it is fair for other users. You can force it with `-l`, however, it is selfish.

-Q (Nicholas Horton): Preemptable queues? A person who paid for a certain machine might like it to be available only to them when they require it all for themselves.

A: Yes, the Grid has support for queues like that. It can all be configured. This particular

example will have to be looked in to further. Beagle.q, as an example, only runs on paikea and overlaps with all.q . Also when the load on paikea, again using that as an example, gets too high, jobs in a certain queue (dnetc.q) are stopped.

-Q (Ross Ihaka): Did I see my desktop listed earlier?

A: No. So far the grid is only running on the servers in the basement and the desktops in the grad room. Desktops in staff offices and used by PhD candidates will have to opt in. This increases the total speed of the grid, but your desktop may run slower at times. It is a two way street.

-Q: Is there job migration?

A: It's crude, and depends on your job. If something goes wrong (eg, the server crashes, power goes out) your job can be restarted on another host. When queue instances become unavailable (eg, we're upgrading paikea) they can send a signal to your job, telling it to save its work and quit, then can be restarted on another host.

-Q (Chris Wild): What happens if a faster host becomes available while my job is running?

A: Nothing. Your job will continue running on the host it is on until it ends. If a host is overloaded, and not due to the grid's fault, some jobs can be suspended until load decreases. The grid isn't migrating jobs. The best method is to break your job down into smaller jobs, so that when the next part of the job is started it gets put onto what is currently the best available host.

-Q (Stephane Guindon): What about when I'm not at my desktop. Can I have my machine be on the grid then, and when I get to the desktop the jobs are migrated?

A: Yes, we can set up calendars so that at certain times no new jobs will be started on your machine. Jobs that are already running will continue until they end. (Disabling the queue.) Since some jobs run for days this can appear to have no influence on how many jobs are running. Alternatively jobs can be paused, which frees up the CPU, but leaves the job sitting almost in limbo. (Suspending the queue.) Remember the grid isn't doing migration. It can stop your job and run it elsewhere (if you're using the --notify option on submission and handling the USR1 signal).