BIOTEAM
Enabling Science

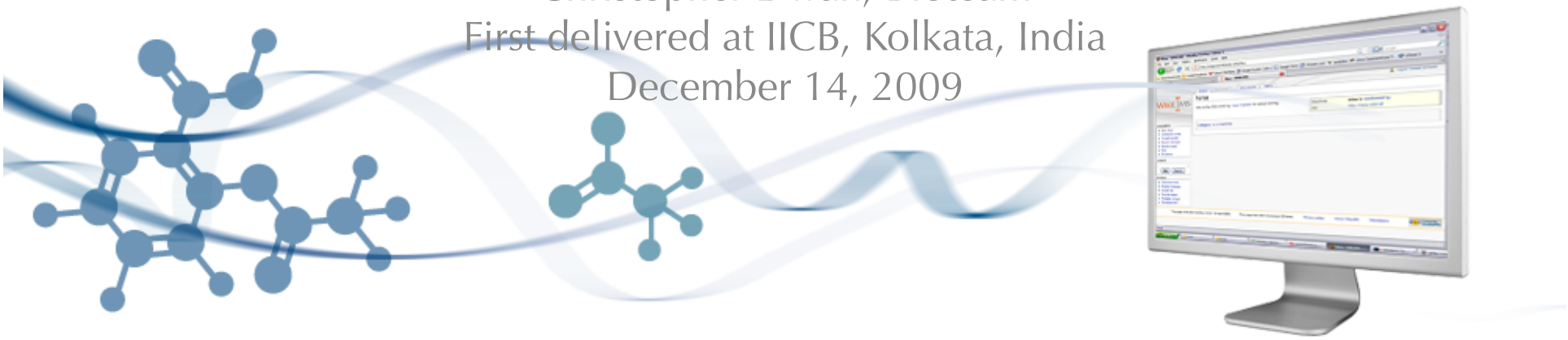
Cluster User Training

From Bash to parallel jobs under SGE in one terrifying hour

Christopher Dwan, Bioteam

First delivered at IICB, Kolkata, India

December 14, 2009



UNIX ESSENTIALS

Unix command line essentials

"**pwd**":	Print current working directory (where am I?)
"**ls**":	File listing
"**ls –l**":	Detailed file listing, including permissions
"**cd**":	Change directory
"**chmod":**	Change file permissions
"**echo**":	Print something to the terminal
"**more**":	Show contents of a text file
"**pico**"	Text editor

"**man**": Documentation on any Unix command:

Basic Unix Exercises: Making directories

What directory am I currently in?

```
Remora:~ cdwan$ pwd
/Users/cdwan/
```

Create a directory named "test_1"

```
Remora:~ cdwan$ mkdir test_1
Remora:~ cdwan$ ls
test_1
```

Change into that directory, and verify that we are there.

```
Remora:~ cdwan$ cd test_1
remora:test_1 cdwan$ pwd
/Users/cdwan/test_1
```

More basic Unix

Return to your home directory:

"**cd**" with no arguments

Exit the session:

"**exit**"

File editing.

"The best script editor" is the subject of an ongoing religious war in technical circles

You should use the tool that does not get in your way.

"**vi**": lightweight, complex, powerful, difficult to use
"**emacs**": heavyweight, complex, powerful, difficult to use
"**pico**": Possibly the simplest editor to use

To edit a file: "**pico filename**"

Hello world in 'bash'

"Bash" is a shell scripting language.

- It is the default scripting language that you have at the terminal.
- I.e: You are already using it.
- We will take this command:

```
remora:test_1 cdwan$ echo "hello world"
hello world
```

And create a wrapper script to do the same thing:

```
remora:test_1 cdwan$ pico hello.sh
remora:test_1 cdwan$ chmod +x hello.sh
remora:test_1 cdwan$ ./hello.sh
hello world
```

Running a set of bash commands

Using pico, create a file named “hello.sh” containing a single line:

```
echo "hello world"
```

Exit pico. Verify the contents of the file:

```
remora:ex_1 cdwan$ more hello.sh
echo "hello world"
```

Then invoke it using the ‘bash’ interpreter:

```
remora:ex_1 cdwan$ bash hello.sh
hello world
```

Hello world script

```
remora:test_1 cdwan$ pico hello.sh

#!/bin/bash

echo "hello world"
```

The "#!" line tells the system to automatically run it using bash

Ctrl-O: Save the file

Ctrl-X: Exit pico

File permissions

Files have properties:

- Read, Write, Execute
- Three different types of user: "owner", "group", "everyone"

To take a script you have written and turn it into an executable program, run "**chmod +x**" on it.

```
remora:test_1 cdwan$ ls -l hello.sh
-rw-r--r--  1 cdwan  staff  32 Dec 14 21:56 hello.sh

remora:test_1 cdwan$ chmod +x hello.sh

remora:test_1 cdwan$ ls -l hello.sh
-rwxr-xr-x  1 cdwan  staff  32 Dec 14 21:56 hello.sh
```

Execute the hello world script

```
remora:test_1 cdwan$ pico hello.sh

remora:test_1 cdwan$ chmod +x hello.sh

remora:test_1 cdwan$ ./hello.sh
hello world
```

SUBMITTING SGE SCRIPTS

Most useful SGE commands

- qsub / qdel
  - Submit jobs & delete jobs
- qstat & qhost
  - Status info for queues, hosts and jobs
- qacct
  - Summary info and reports on completed job
- qrsh
  - Get an interactive shell on a cluster node
  - Quickly run a command on a remote host
- qmon
  - Launch the X11 GUI interface

Interactive Sessions

To generate an interactive session, scheduled on any node:
"qlogin"

```
applecluster:~ cluster$ qlogin

Your job 145 ("QLOGIN") has been submitted
waiting for interactive job to be scheduled ...
Your interactive job 145 has been successfully scheduled.
Establishing /common/node/ssh_wrapper session to host node002.cluster.private ...
The authenticity of host '[node002.cluster.private]:50726 ([192.168.2.2]:50726)'
    can't be established.
RSA key fingerprint is a7:02:43:23:b6:ee:07:a8:0f:2b:6c:25:8a:3c:93:2b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[node002.cluster.private]:50726,[192.168.2.2]:
    50726' (RSA) to the list of known hosts.
Last login: Thu Dec  3 09:55:42 2009 from portal2net.cluster.private

node002:~ cluster$

all.q@node002.cluster.private  BIP   1/8       0.00     darwin-x86
    145 0.55500 QLOGIN     cluster      r     12/15/2009 09:15:08
```

Requesting the whole node

```
qlogin -pe threaded 8
```

```
all.q@node014.cluster.private  BIP   8/8        0.00     darwin-x86
    146 0.55500 QLOGIN     cluster       r     12/15/2009 09:34:58     8
```

We request a parallel environent called “threaded” (note, this PE does not exist by default in SGE – we create it in iNquiry)

We request 8 slots within that environment

Now, no other jobs will be scheduled to your node while that login is in place.

Most basic job example

```
qsub –b y /bin/hostname
```

You will see two new files in your home directory:

```
hostname.oYYY
hostname.eYYY
```

YYY is the job id provided by the queuing system.

These are the standard output and standard error files from running /bin/hostname on one of the nodes.

Argument **"-b y"** indicates that this is a a compiled binary. SGE will not try to parse the input, but merely run it.

Creating the sleeper script

```
#!/bin/bash

echo "hello world"
sleep 60
hostname
```

Then run like this:

```
remora:test_1 cdwan$ cp hello.sh sleeper.sh
remora:test_1 cdwan$ pico sleeper.sh
remora:test_1 cdwan$ ./sleeper.sh
hello world
remora.local
```

Submitting the sleeper script

```
genesis2:example bioteam$ qsub -cwd -S /bin/bash sleeper.sh
Your job 217 ("sleeper.sh") has been submitted

genesis2:example bioteam$ qstat
job-ID  prior   name       user         state submit/start at
   queue                          slots ja-task-ID
-----------------------------------------------------------------
   ------
    217 0.55500 sleeper.sh bioteam      r     12/14/2009 12:00:46
   all.q@node004.cluster.private      1
```

Adding arguments directly into the script

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd

echo "hello world"
sleep 60
hostname
```

Any comment that starts with "#$" is interpreted as an argument to qsub.

In case of conflict, the command line wins.

More SGE commands

- "**qstat –f**": Show all queues, even the empty ones
- "**qstat –u** *": Show jobs from all users
- "**qstat –f –u** *": Both all queues and uses

- "**qdel job_id**": Delete a particular job
- "**qdel –u cdwan**": Delete all jobs run by user cdwan

Giving your job a name

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N sleeper

echo ”Hello world”
sleep 60
hostname
```

```
genesis2:demo bioteam$ qsub sleeper.sh
Your job 222 ("sleeper") has been submitted
```

Resource Requirements

```
genesis2:demo bioteam$ qsub -l arch=solaris sleeper.sh
Your job 225 ("sleeper") has been submitted


genesis2:demo bioteam$ qstat
job-ID  prior   name       user         state
-----------------------------------------------
    225 0.00000 sleeper    bioteam      qw
```

We specify a resource requirement that cannot be met (there are no solaris machines in the cluster)

Qstat –j 225 tells the story:

(-l arch=solaris) cannot run at host "node008.cluster.private" because it offers only hl:arch=darwin-ppc

Environment variables from SGE

SGE sets several variables in the script for you.

- JOB_ID numerical ID of the job

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N sleeper

echo "My job id is " $JOB_ID
sleep 60
```

```
genesis2:demo bioteam$ more sleeper.o221
My job id is  221
```

Job dependencies

```
genesis2:demo bioteam$ qsub -N "primary" sleeper.sh

genesis2:demo bioteam$ qsub -hold_jid primary -N
   "secondary" sleeper.sh
Your job 224 ("secondary") has been submitted


genesis2:demo bioteam$ qstat
job-ID  prior   name       user         state
---------------------------------------------
    223 0.55500 primary    bioteam      r
    224 0.00000 secondary  bioteam      hqw
```

Redirecting output

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -o task.out
#$ -e task.err

echo "Job ID is " $JOB_ID
```

Task arrays

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -N task_array
#$ -o task.out
#$ -e task.err

echo "Job ID is " $JOB_ID " Task is " $SGE_TASK_ID
```

Task arrays

```
genesis2:example_2 bioteam$ qsub -t 1-10  task.sh
Your job-array 228.1-10:1 ("task_array") has been
   submitted

genesis2:example_2 bioteam$ more task.out
Job ID is  228  Task ID is  10
Job ID is  228  Task ID is  1
Job ID is  228  Task ID is  3
Job ID is  228  Task ID is  4
```

- ...

Perl instead of bash

My “hello world” script, using Perl instead of Bash:

```
remora:~ cdwan$ more hello_world.pl
#!/usr/bin/perl
#! –S /usr/bin/perl

sleep(60);
print "Hello world\n";
```

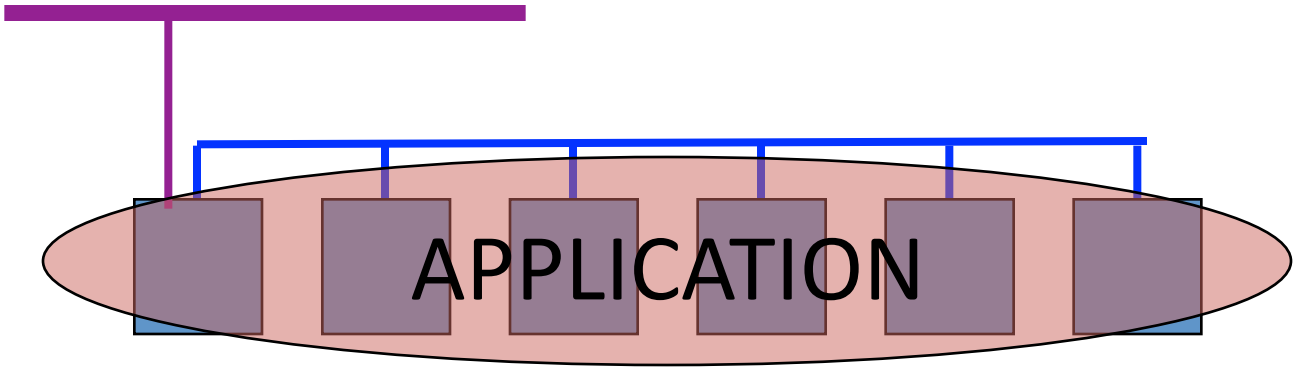
Submitted to the queuing system:

```
qsub hello_world.pl
```

PERFORMANCE TUNING / PARALLELIZATION

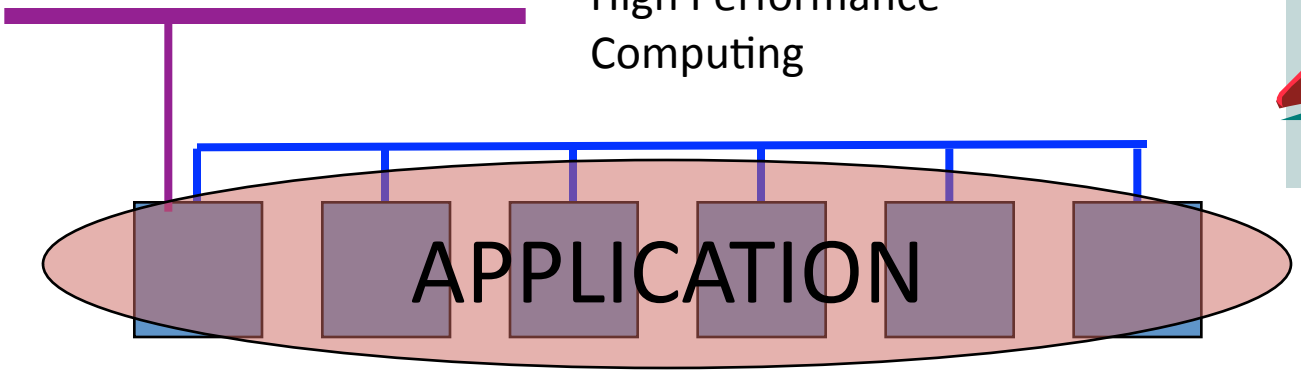
Parallel Jobs

- A parallel job runs simultaneously across multiple servers
  - Biggest SGE job I've heard of: Single application running across 63,000 CPU cores: TACC "Ranger" Cluster in Texas
  - Distinction with 'batches' of processes that include many tasks to be done in any order

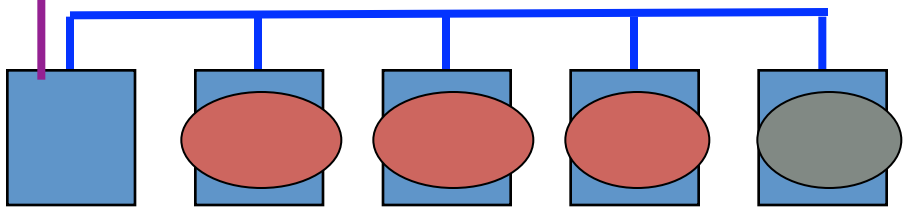


Parallel computing 101

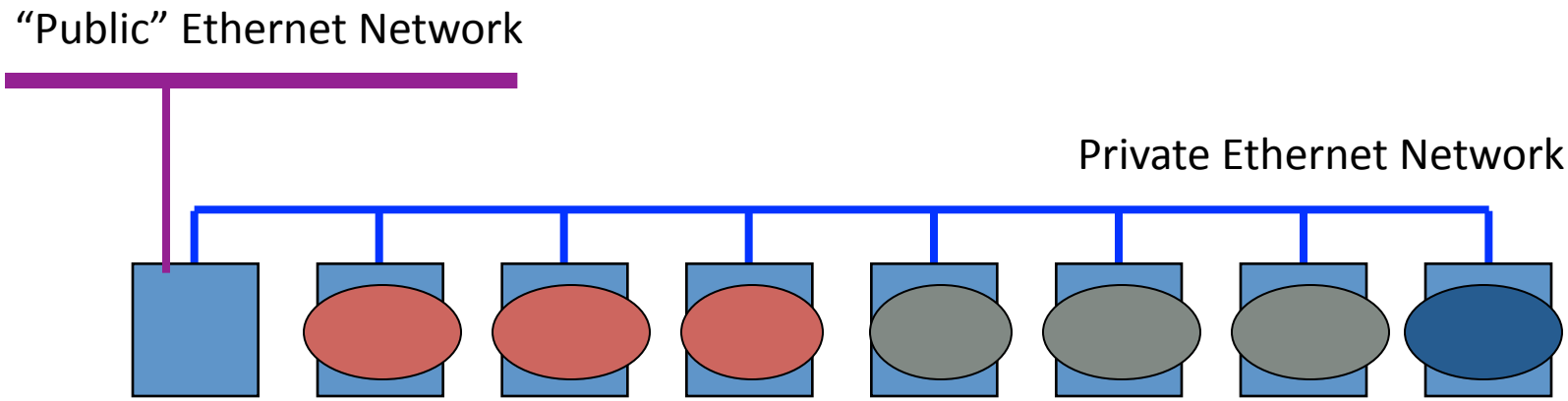
High Performance Computing



High Throughput Computing (farm)



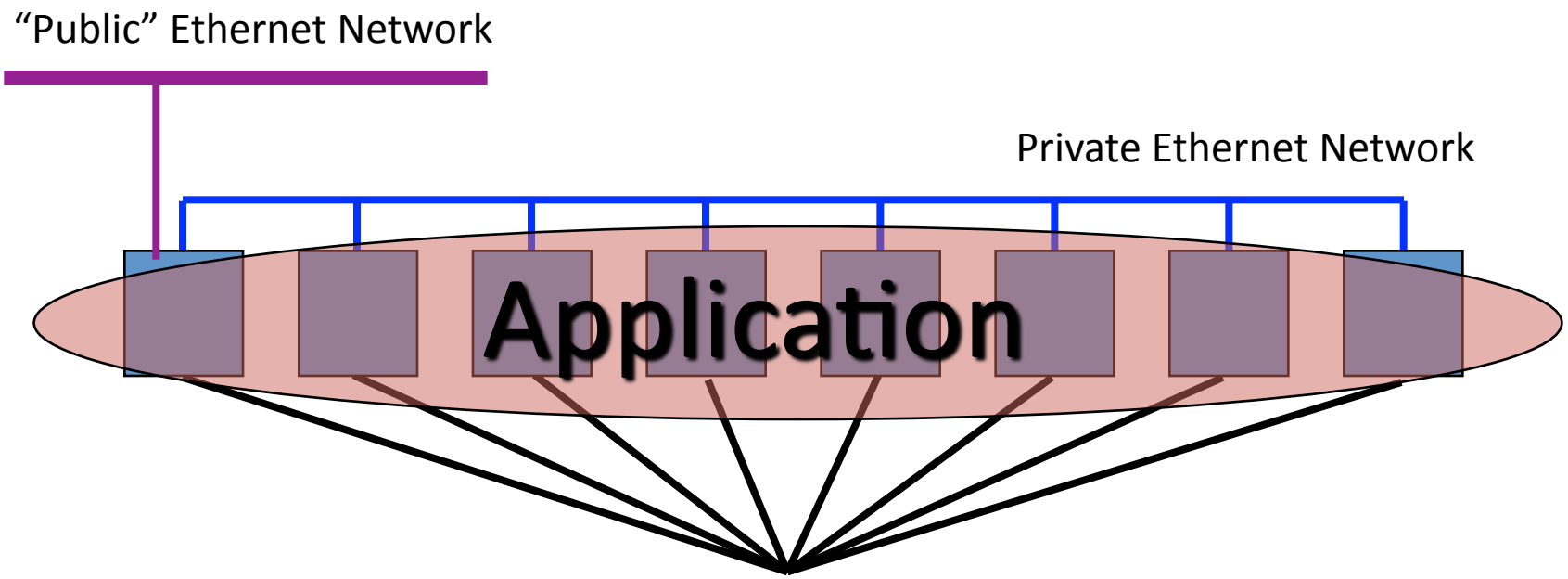
Batch Jobs



- Independent applications running at the same time
- Many jobs (batch)
- Maximum efficiency, simple to write

http://bioteam.net
Chris Dwan cdwan@bioteam.net

Tightly Coupled / Parallel



One parallel application running over the entire cluster

- One job, where response time is important.
- Overall efficiency is lower
- Scalability is hard

http://bioteam.net
Chris Dwan cdwan@bioteam.net

Amdahl's Law

Maximum expected speedup for parallelizing any task

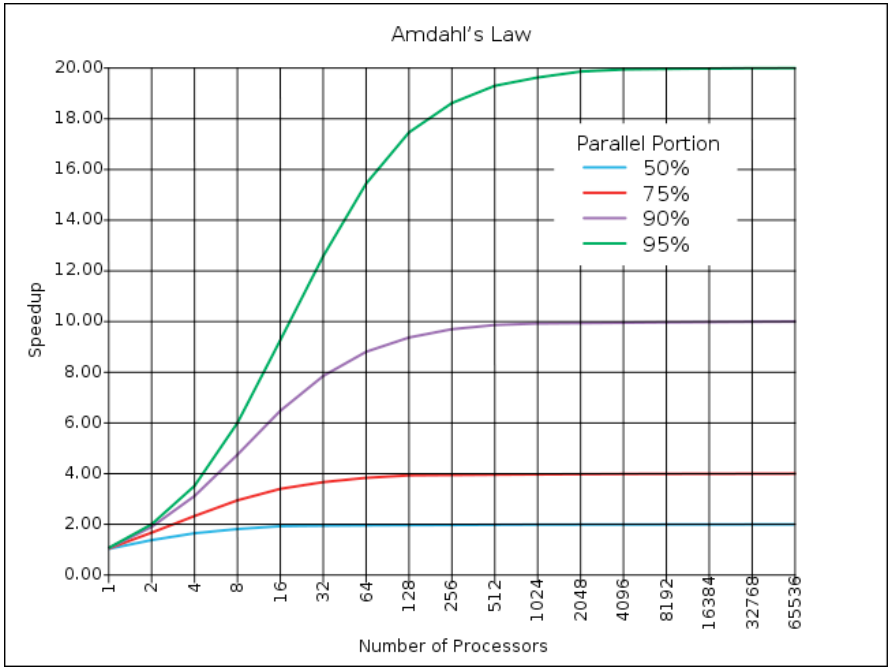
- Serial portion (non parallelizable)
- Parallel portion (can be parallelized)

Additionally:

- Cost associated with using more machines (startup, teardown)
- At least, scheduling. Possibly some other factors like communication
- Communication scales with number of processes

Important to note:

Re-stating the problem can radically alter the serial / parallel ratio



Network Latency

- **Latency**:
  - Time to initiate communication
- **Throughput**:
  - Data rate once communication is established

- **Gigabit Ethernet:**
  - Latencies: ~100ms
  - Throughputs up to 80% of wire speed (800Mb/sec)
  - $10 / network port

- **Myranet / Infiniband:**
  - Latencies: ~3ms
  - Throughput: 80% of wire speed
  - $800 / network port

RUNNING PARALLEL TASKS

Parallel Jobs

Many different software implementations are used to support parallel tasks:

- MPICH
- LAM-MPI
- OpenMPI
- PVM
- LINDA

No magic involved

- Requires work
- Your application must support parallel methods

Submitting a standalone MPI job

Build the code with a particular version of MPI:

```
genesis2:examples bioteam$ pwd
/common/mpich/ch_p4/examples

genesis2:examples root# which mpicc
/common/mpich/ch_p4/bin/mpicc

genesis2:examples root# mpicc cpi.c

genesis2:examples root# mpicc -o cpi cpi.o
```

Run without any MPI framework:

```
genesis2:examples root# ./cpi
Process 0 on genesis2.mit.edu
pi is approximately 3.1416009869231254, Error is
   0.0000083333333323
wall clock time = 0.000214
```

Submitting a standalone MPI job (no SGE)

MPI needs to know which hosts to use: We create a hosts file which simply lists the machine

```
genesis2:~ bioteam$ more hosts_file
node001
node002
Node003
node004
```

Then start the job using 'mpirun'

```
genesis2:~ bioteam$ mpirun -machinefile hosts_file \
          -np 4 /common/mpich/ch_p4/examples/cpi
Process 0 on genesis2.mit.edu
Process 2 on node002.cluster.private
Process 3 on node003.cluster.private
Process 1 on node001.cluster.private
pi is approximately 3.1416009869231249, Error is 0.0000083333333318
wall clock time = 0.002106
```

Critical Notes

In order to have MPICH jobs work reliably, you need to compile and run them with the same version of MPICH.

/common/mpich

/common/mpich2

/common/mpich2-64

All user account issues must be in order for this to work.

- Password free ssh in particular

If the application does not work from the command line, SGE will not help.

Loose Integration with SGE

Loose Integration

- Grid Engine used for:
  - Picking when the job runs
  - Picking where the job runs
  - Generating the custom machine file
- Grid Engine does not:
  - Launch or control the parallel job itself
  - Track resource consumption or child processes

Advantages of loose integration

- Easy to set up
- Can trivially support almost any parallel application technology

Disadvantages of loose integration

- Grid Engine can't track resource consumption
- Grid Engine must "trust" the parallel app to honor the custom hostfile
- Grid Engine can not kill runaway jobs

Tight integration with SGE

Tight Integration

- Grid Engine handles all aspects of parallel job operation from start to finish
- Includes spawning and controlling all parallel processes

Tight integration advantages:

- Grid Engine remains in control
- Resource usage accurately tracked
- Standard commands like "`qdel`" will work
  - Child tasks will not be forgotten about or left untouched

Tight Integration disadvantages:

- Can be really hard to implement
- Makes job debugging and troubleshooting harder
- May be application specific

Running an mpich job with loose SGE integration

Step one: Job must work without SGE.

- Until you can demonstrate a running job using a host file and manual start up, there is no point to involving SGE

Step two: Create a wrapper script to allow SGE to define the list of hosts and the number of tasks

Step three: Submit that wrapper script into a 'parallel environment'

- Parallel environment manages all the host list details for you.

MPICH Wrapper for CPI

A trivial MPICH wrapper for Grid Engine:

```
#!/bin/bash

## ---- EMBEDDED SGE ARGUMENTS ----
#$ -N MPI_Job
#$ -pe mpich 4
#$ -cwd
#$ -S /bin/bash
## -------------------------------------
MPIRUN=/common/mpich/ch_p4/bin/mpirun
PROGRAM=/common/mpich/ch_p4/examples/cpi
export RSHCOMMAND=/usr/bin/ssh

echo "I got $NSLOTS slots to run on!"
$MPIRUN -np $NSLOTS -machinefile $TMPDIR/machines $PROGRAM
```

Job Execution

Submit just like any other SGE job:

```
[genesis2:~] bioteam% qsub submit_cpi
Your job 234 ("MPI_Job") has been submitted
```

Output files generated:

```
[genesis2:~] bioteam% ls -l *234
-rw-r--r--  1 bioteam  admin  185 Dec 15 20:55 MPI_Job.e234
-rw-r--r--  1 bioteam  admin  120 Dec 15 20:55 MPI_Job.o234
-rw-r--r--  1 bioteam  admin   52 Dec 15 20:55 MPI_Job.pe234
-rw-r--r--  1 bioteam  admin  104 Dec 15 20:55 MPI_Job.po234
```

Output

```
[genesis2:~] bioteam% more MPI_Job.o234
I got 5 slots to run on!
pi is approximately 3.1416009869231245, Error is
   0.0000083333333314
wall clock time = 0.001697

[genesis2:~] bioteam% more MPI_Job.e234
Process 0 on node006.cluster.private
Process 1 on node006.cluster.private
Process 4 on node004.cluster.private
Process 2 on node013.cluster.private
Process 3 on node013.cluster.private
```

Parallel Environment Usage

- "qsub -pe mpich 4 ./my-mpich-job.sh"
- "qsub -pe mpich 4-10 ./my-mpich-job.sh"
- "qsub -pe lam-loose 3 ./my-lam-job.sh"

Behind the Scenes: Parallel Environment Config

```
genesis2:~ bioteam$ qconf -sp mpich
pe_name            mpich
slots              512
user_lists         NONE
xuser_lists        NONE
start_proc_args    /common/sge/mpi/startmpi.sh $pe_hostfile
stop_proc_args     /common/sge/mpi/stopmpi.sh
allocation_rule    $fill_up
control_slaves     FALSE
job_is_first_task  TRUE
urgency_slots      min
```

Behind the scenes: MPICH

The "`startmpi.sh`" script is run before job launches and creates custom machine file

The user job script gets date required by '`mpirun`' from environment variables:

`$NODES, $TEMPDIR/machines`, etc.

The "`stopmpi.sh`" script is just a placeholder

Does not really do anything (no need yet)

Behind the scenes: LAM-MPI

- Just like MPICH
- But 2 additions:
  - The “lamstart.sh” script launches LAMBOOT
  - The “lamstop.sh” script executes LAMHALT at job termination
- In an example configuration, lamboot is started this way:
  - `lamboot -v -ssi boot rsh -ssi rsh_agent "ssh -x -q" $TMPDIR/machines`

Behind the scenes: LAM-MPI

- A trivial LAM-MPI wrapper for Grid Engine:

```
#!/bin/sh

MPIRUN="/common/lam/bin/mpirun"

## ---- EMBEDDED SGE ARGUMENTS ----
#$ -N MPI_Job
#$ -pe lammpi 3-5
#$ -cwd
## --------------------------------------
echo "I have $NSLOTS slots to run on!"
$MPIRUN C ./mpi-program
```

OpenMPI

In absence of specific requirements, a great choice

Works well over Gigabit Ethernet

Trivial to achieve tight SGE integration

Recent personal experience:

- Out of the box: 'cpi.c' on 1024 CPUs
- Out of the box: heavyweight genome analysis pipeline on 650 Nehalem cores

Behind the scenes: OpenMPI

OpenMPI 1.2.x natively supports automatic tight SGE integration

- Build from source with "--enable-sge"
- `mpirun -np $NSLOTS /path-to-my-parallel-app`

OpenMPI PE config:

```
pe_name            openmpi
slots              4
user_lists         NONE
xuser_lists        NONE
start_proc_args    /bin/true
stop_proc_args     /bin/true
allocation_rule    $round_robin
control_slaves     TRUE
job_is_first_task  FALSE
urgency_slots      min
```

OpenMPI Job Script

```
#!/bin/sh

## ---- EMBEDDED SGE ARGUMENTS ----
#$ -N MPI_Job
#$ -pe openmpi 3-5
#$ -cwd
## --------------------------------------

echo "I got $NSLOTS slots to run on!"
mpirun -np $NSLOTS ./my-mpi-program
```

Application profiling

- **Most basic: System Monitoring**
  - 'top'
  - Ganglia
- **Apple 'shark' tools**
- **Deep understanding of code.**

Tuning parallel jobs

- **Round Robin:**
  - Jobs are distributed to as many nodes as possible
  - Good for tasks where memory may be the bottleneck

- **Fill up:**
  - Jobs are packed onto as few nodes as possible
  - Good for jobs where interprocess communications may be the bottleneck

- **Single chassis**
  - “threaded” environment from earlier sessions
  - For multi-threaded programs (BLAST)

Thank you