Programming Environment

Cornell Center for Advanced Computing

December 11, 2012

Thanks to Dan Stanzione, Bill Barth, Lars Koesterke, Kent Milfeld, and Robert McLay

for their materials developed at TACC and XSEDE

that were incorporated into this talk.

Summary

1. Accessing Ranger/Lonestar
2. System Environment
3. System Overview
4. Software
5. Compiling
6. Performance
7. Editing Files
8. Batch Job Submission: SGE
9. Batch Job Submission: SLURM
10. Additional Labs

1. Accessing Ranger/Lonestar

Blue boxes show instructions for Lonestar

Yellow boxes show instructions for Ranger

Gray boxes show instructions for both

Login with SSH: Lonestar

- SSH Secure Shell for Windows
- Built-in as “ssh” for Linux or Mac
- You will be connected to login#.ls4.tacc.utexas.edu
- **Do not** save the new host key

Login to lonestar.tacc.utexas.edu:

% ssh *username*@lonestar.tacc.utexas.edu
-or-
All Programs | ClassFiles | SSH Secure Shell | SecureShell Client
use Host Name: lonestar.tacc.utexas.edu

Login with SSH: Ranger

- SSH Secure Shell for Windows
- Built-in as “ssh” for Linux or Mac
- You will be connected to login#.ranger.tacc.utexas.edu
- **Do not** save the new host key

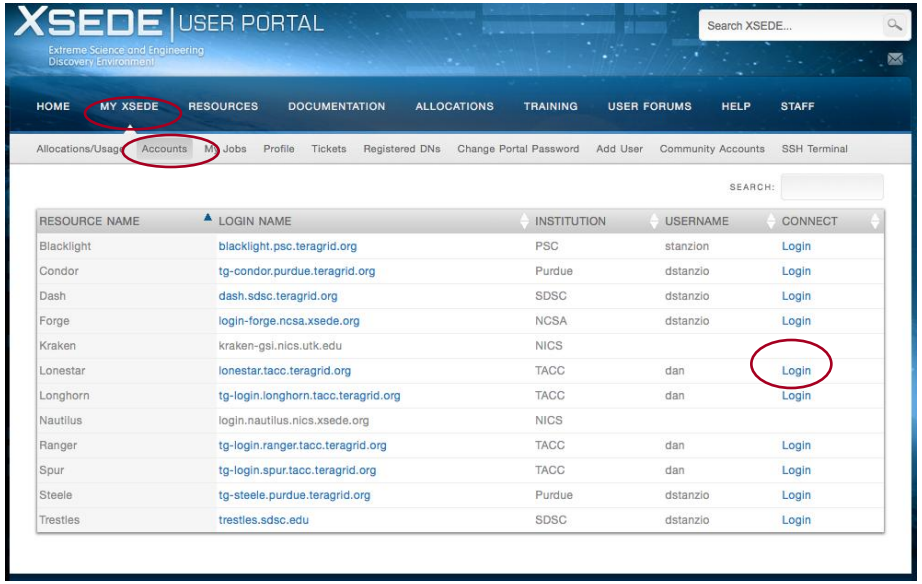
Login to ranger.tacc.utexas.edu:

% ssh *username*@ranger.tacc.utexas.edu
-or-
All Programs | ClassFiles | SSH Secure Shell | SecureShell Client
use Host Name: ranger.tacc.utexas.edu

Login with SSO

- Go to the XSEDE User Portal: portal.xsede.org
- Log in
- Go to ‘My XSEDE’ tab
- Go to the ‘Accounts’ link
- Use the appropriate ‘login’ link
- Note your username

Login using the XSEDE portal



Single Sign On (SSO)

- SSO allows you to use just one username and password (your User Portal one) to log into every digital service on which you have an account.
- The easiest way to use SSO is via the XSEDE User Portal, but you can also use SSO via a desktop client or with an X.509 certificate.
- After you authenticate using SSO with your User Portal username and password, you will be recognized by all XSEDE services on which you have account, without having to enter your login information again for each resource.

2. System Environment

Account Info

Note your account number in the splash screen.

```
---------------------- Project balances for user tg459571 ----------------------
| Name           Avail SUs     Expires |                                       |
| TG-TRA120006       49998             |                                       |
------------------------ Disk quotas for user tg459571 -------------------------
| Disk          Usage (GB)     Limit    %Used   File Usage       Limit   %Used |
| /home1             0.0       1.1     0.13            63      101000    0.06 |
| /work              0.0     250.0     0.00             1      500000    0.00 |
-------------------------------------------------------------------------------
```

Get the Lab Files

- TAR = Tape ARchive. Just concatenates files.
- tar <switches> <files>
  - z = compress or decompress
  - x = extract
  - c = create
  - v = verbose
  - t = list files
  - f = next argument is the file to read or write

Get the lab files:
$ tar xvf ~tg459572/LABS/envi.tar

Change directory to the envi directory:
$ cd envi
List the lab files:
$ ls -l

Exercise: Lonestar Commands

```
$ pwd                                   ( Print the current directory)
$ ls –la                                ( List the content of the current directory)
$ cd $HOME
$ cat .login                            ( Print the file .login to the screen)
$ mkdir testdir                         ( Create the directory, testdir )
$ touch  test.txt                       ( touch command renews a file's
                                  timestamp, but here is used to create an empty file)
$ mv test.txt testdir
$ ls –F testdir
$ rm –r testdir
$ man ls                                (Show the manuel page for ls, ,'q' to quit)
$ env                                   (Show all environment/global variables)
$ export newgreeting="Hello World"      (Set an environmental variable)
$ echo $newgreeting                     ( Print the variable newgreeting)
```

Exercise: Ranger

```
% env    (Show environment variables – persists.  Use setenv to set variables)
% setenv newgreeting "Hello World" (Set a global variable)
% echo $newgreeting
% pwd
% ls –la
% cd $HOME
% cat .login
% mkdir testdir
% touch  test.txt  ( touch command renews a file‘s timestamp, but here is used to
                     create an empty file)
% mv test.txt testdir
% ls –F testdir
% rm –r testdir
% man ls (Manuel page)
```

Shell Information

- **`chsh:`** Change/examine system's shell information
- Full path of the available shells: "**`chsh -l`**"
- Set/change login shell: "**`chsh -s <full path to the shell>`**"
  - Takes some time to propagate (~1 hour)
- Lonestar/Stampede default: **`bash`**
- Ranger default: **`csh`**

```
$ echo $SHELL
$ chsh -l
```

System Configuration Files

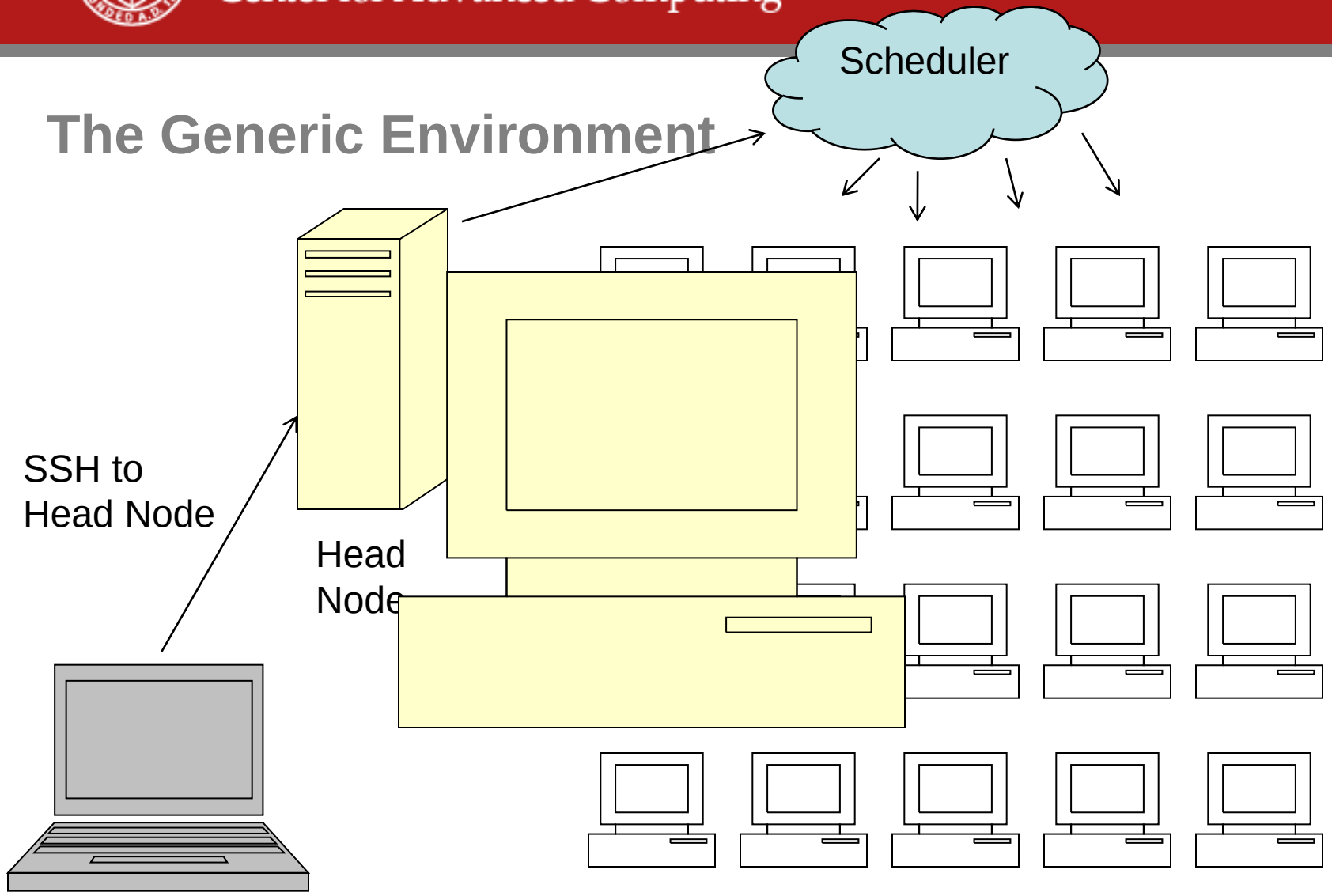
- System-wide configuration script: Executed automatically when login
- User-customized configuration script:
  - ~/.profile_user : Invoked by the login shell
  - ~/.bashrc : Invoked by the interactive shell (bash).
- TACC Instructions for editing customization script

System-wide config scripts:
Bash: /etc/tacc/profile
/etc/tacc/bashrc
csh: /etc/tacc/csh.cshrc
/etc/tacc/csh.login

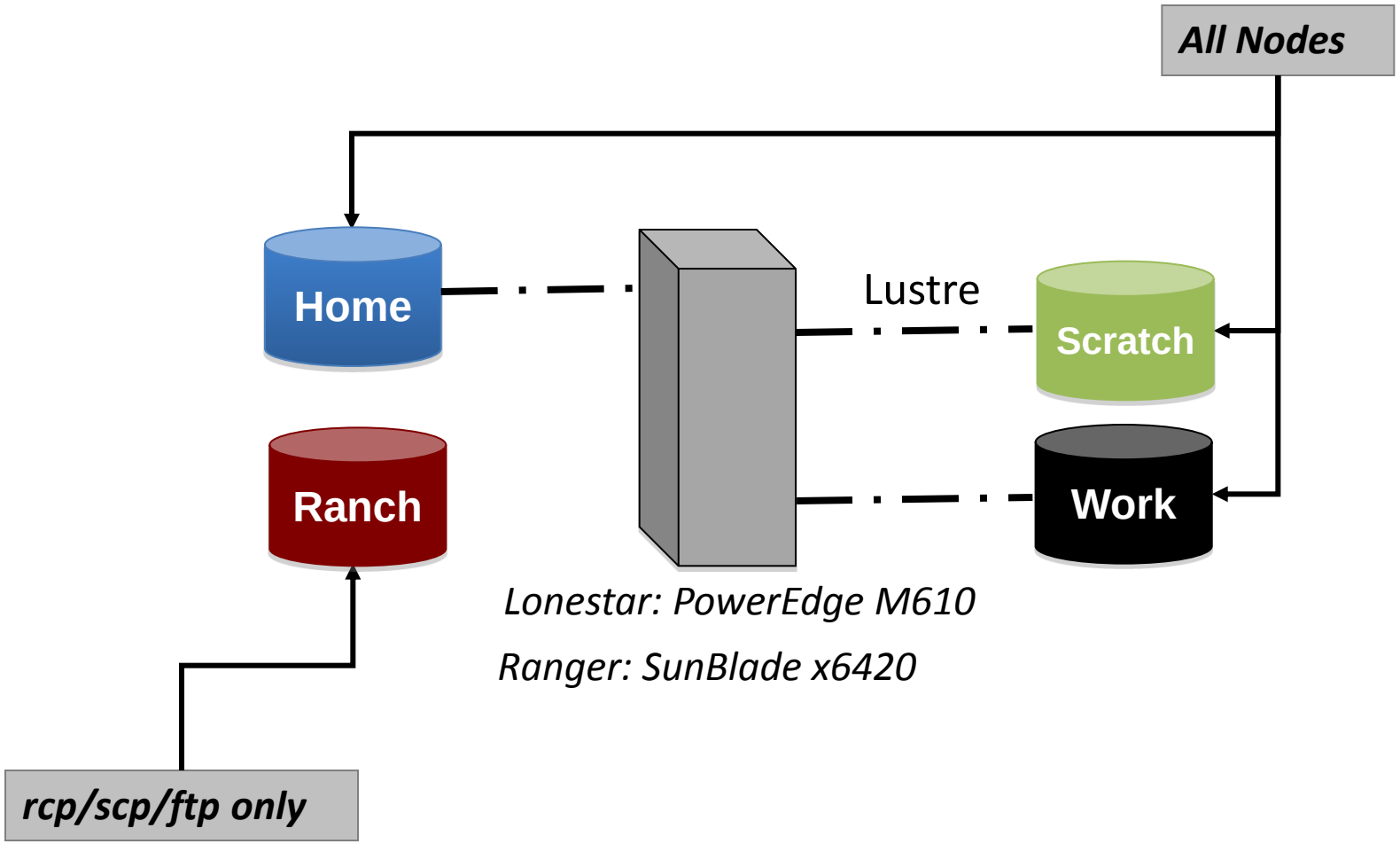
User-customizable config script:
Bash: ~/.bashrc ~/.profile_user
csh: ~/.cshrc ~/.login_user

3. System Overview

The Generic Environment



File Systems (Ranger/Lonestar)



File System

Environment Variable	Purpose	User Access Limits	Lifetime
$HOME	Source code	6 GB quota (Ranger) 1 GB quota (Lonestar)	Project (Backup)
$WORK	Large file storage	200 GB quota (Ranger) 250 GB quota (Lonestar)	Project (No backup)
$SCRATCH	Large files needed by compute jobs	~1.4 PB	10 Days (purged)
/tmp	Local disk on batch job node	300 MB (Ranger) 65 GB (Lonestar)	job
$ARCHIVE	Archival tape	~10 PB	Project

System	Ranger	Lonestar	Stampede
Purpose	HPC	HPC	HPC
Nodes	3,936	1,888	6000+
CPUS/node x cores/CPUS	4 x 4	2 x 6	2 x 8
Total cores	62,976	22,656	96,000+
CPUS	AMD Barcelona 2.3GHz	Intel Westmere 3.3GHz	Intel Sandy Bridge Intel Xeon Phi coprocessor
Memory	2GB/core	2GB/core	2GB/core (128 nodes) 1 TB/node (16 nodes)
Interconnect	~8Gb/s SDR IB	40Gb/s QDR IB	56Gb/s FDR IB
Disk	1.7PB Lustre (IB)	1PB Lustre (IB)	14PB Lustre (IB)

Expected file systems: Stampede

- Integration with archive, and (coming soon) TACC global work file system and other data systems
- A global, high-speed file system, running across 72 I/O servers uses the Lustre file system.
- Users will interact with the system via multiple dedicated login servers, and a suite of high-speed data servers.
- File systems on Stampede will mirror Ranger/Lonestar layout (/home, /work, /scratch)

Disk Usage

%quota <*username*>
%lfs quota –u <*username*> $WORK
%lfs quota –u <*username*> $SCRATCH
%cd change directory to $HOME
%pwd
%cdw change directory to $WORK
%pwd
%cds change directory to $SCRATCH
%pwd

%lfs quota –u <*username*> $WORK
%lfs quota –u <*username*> $HOME
%cd change directory to $HOME
%pwd
%cdw change directory to $WORK
%pwd
%cds change directory to $SCRATCH
%pwd

4. Software

Software

Software available on Ranger and Lonestar

Software search available on XSEDE

Use the module utility on Ranger, Lonestar, and Stampede to provide a consistent, uniform method to access software.

Module

This utility is used to set up your PATH and other environment variables:

$ module help	{lists options}	% module help
$ module avail	{lists available modules}	% module avail
$ module list	{lists loaded modules}	% module list
$ module load intel	{add a module}	% module load pgi
$ module load pgi	{try to load intel}	% module load intel
$ module swap intel pgi	{swap two modules}	% module swap pgi intel
$ module list	{compare to the old list}	% module list
$ module load boost	{add a module}	% module load boost
$ module unload boost	{remove a module}	% module unload boost
$ module help boost	{module-specific help}	% module help boost
$ module spider	{lists all modules}	% module spider
$ module spider petsc	{list all version of petsc}	% module spider petsc

MODULE Command

- Can be used in the batch file and Makefile
- The *module* command may alter:
  - User path:
    - $PATH, $MANPATH, $LIBPATH
  - Environmental Variables:
    - mkl: TACC_MKL_LIB, TACC_MKL_INC
    - Gotoblas: TACC_GOTOBLAS_LIB
- First choose compiler, then application software.

More Module Notes:

- Customize the default software list

  $ module purge

  $ module load TACC git petsc

  $ module setdefault (The system will load TACC, git, and petsc on default
- Restore System Default

  $ module restore system
- TACC supports two Families
  - Compilers
  - MPI implementations.
- Only one compiler, one MPI stack.
  - Env. Var: TACC_FAMILY_COMPILER: intel, pgi, gcc
  - Env. Var: TACC_FAMILY_MPI: mvapich, mvapich2, openmpi

5. Compiling

Compiling Serial Code

- Lonestar default: Intel compilers (as will Stampede).
- Ranger default: Portland Group, pgi compiler.
- Use the **module** utility to examine compiler information and change between compilers

Vendor	Compiler	Language	File Extension	Example
intel	icc	C	.c	$ icc -o test.exe -O3 prog.c
intel	icpc	C++	.C/c/cc/cpp/cxx/c++/i/ii	
intel	ifort	F77/F90/F95	.f, .for, .ftn, .f90, .fpp	$ ifort -o test.exe -O3 prog.f90
pgi	pgcc	C	.c	
pgi	pgcpp	C++	.C, .cc	
pgi	pgf95	F77/90/95	.f, .F, .FOR, .f90, .f95, .hpf	$ pgf95 -o test.exe prog.f90
gnu	gcc	C	.c	$ gcc -o test.exe prog.c

See the User Guide for the complete list.

Compiler Options

- Use compiler options to achieve optimal performance.
- Obtain best results by
  - Select the appropriate optimization level
  - Target the architecture of the computer (CPU, cache, memory system)
  - Allow for interprocedural analysis (inlining, etc.)
- No single answer for all cases; test different combinations.

Intel Option	Description
-O3	performs some compile time and memory intensive optimizations in addition to those executed with -O2, but may not improve performance for all programs.
-xW	Includes specialized code for SSE and SSE2 instructions (recommended).
-xO	Includes specialized code for SSE, SSE2 and SSE3 instructions. Use, if code benefits from SSE3 instructions.
-fast	Includes: -ipo, -O2, -static DO NOT USE -- static load not allowed.

PGI Option	Description
-O3	Performs some compile time and memory intensive optimizations in addition to those executed with -O2, but may not improve performance for all programs.
-tp barcelona-64	Includes specialized code for the barcelona chip.
-fast	Includes: -O2 -Munroll=c:1 -Mnoframe -Mlre -Mautoinline -Mvect=sse -Mscalarsse -Mcache_align -Mflushz

6. Program Performance

Timers

- Using **Wall-clock** to time your code to gauge effectiveness of code and software changes.
- **/usr/bin/time -p <executable>** is preferred over the shell's time command
  (-p specifies traditional precision output in seconds)

```
$ cd $HOME/envi/intro
$ make
g++ hello.c -o hello
$ /usr/bin/time -p ./hello
Hello world!
real 0.01
user 0.00
sys 0.01
$
```

You can also time specific sections of your code by inserting timer calls before and after important sections.

Profilers: gprof (GNU profiler)

- gprof reports a basic profile of time spent in each subroutine
- Find the most time-consuming routines, the hotspots
- Read the data file into an ASCII report or a graphic display.
- **Compile the code using** *-pg* **option** (Intel) **to enable profiling.**
- More detail can be found in the Profiling and Debugging Virtual Workshop module.

(Remember to swap back to intel compiler – *module swap pgi intel*)

```
$ cd $HOME/envi/precision
$ ifort –pg precision.f90      instrument code with –gp (if pgi is your default use pgf90)
$ a.out                        produce gmon.out trace file
$ gprof  | more                reads gmon.out (default args: a.out gmon.out) report sent to
                               STDOUT
```

Profilers: gprof (GNU profiler)

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        1     0.00     0.00  MAIN__

…
…
…
```

7. Text Editor

Available Text Editor

- vi (vim)
- nano
- emacs
- nedit (A Graphic User Interface Text Editor)
- Edit files on your desktop, and transfer the files to the system

nano

- All operations commands are preceded by the Control key:
  - ^G Get Help
  - ^O WriteOut
  - ^X Exit
  - ….
- If you have modified the file and try to exit (^X) without writing those changes (^O) you will be warned.
- Makes text editing simple, but it has less powerful options than vi (search with regular expressions, etc..)

vi (short for “visual”)

- “vi filename” will open it or create it if it doesn’t exist.
- Command mode: keystrokes are commands
- Input mode: keystrokes are text you are adding to the file
- Last line mode: start with : end with <return>
- Examples:
  - i Insert characters before current position (use ESC to exit)
  - dd Delete current line
  - R Overwrite existing text (until ESC)
  - u Undo last operation
  - :wq Writes a file to disk and exit editor
  - :q! Quit without saving

Use Your Computer's Editor

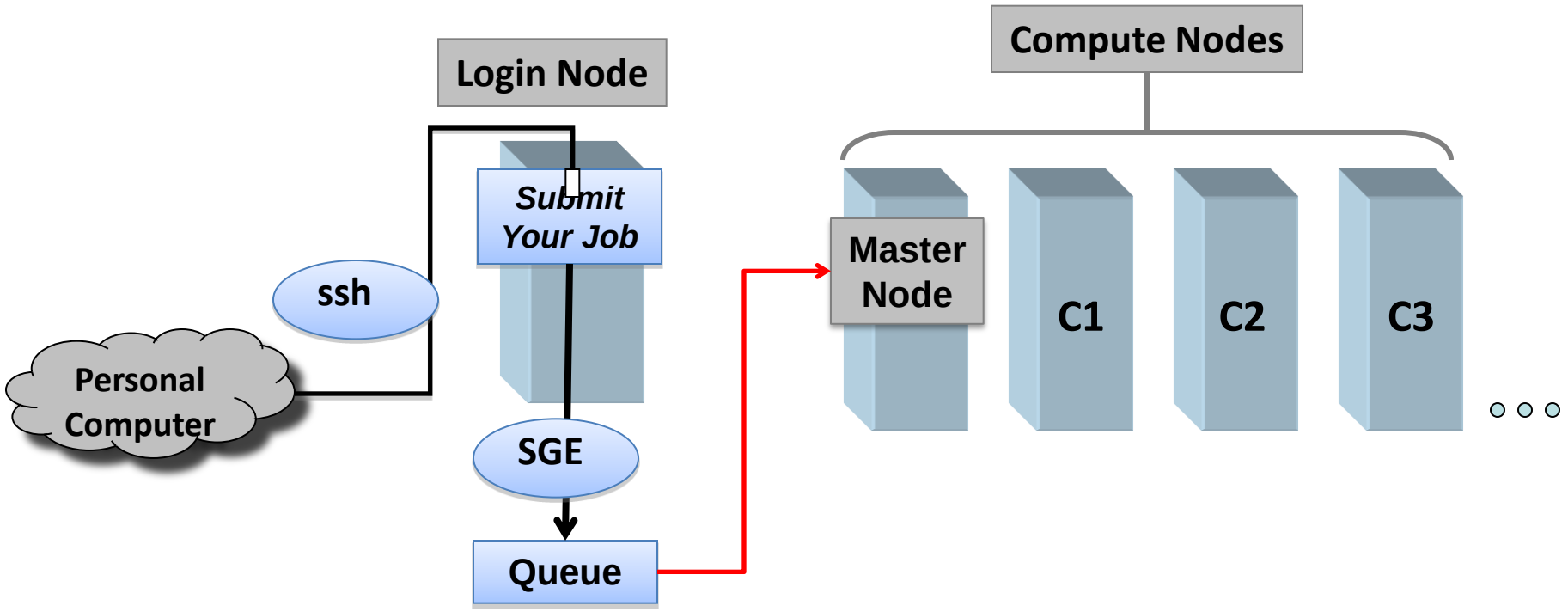
Copying the file to your computer might be quicker than learning a new editor. Use a simple file transfer client:

Start menu
All Programs
Class Files
SSH Secure Shell
Secure File Transfer Client ← Right click, "Pin to Start Menu"

Start Secure File Transfer Client
Use Quick Connect, specify hostname lonestar.tacc.utexas.edu
In the left pane, navigate to the desktop.
Drag files between panes to copy.

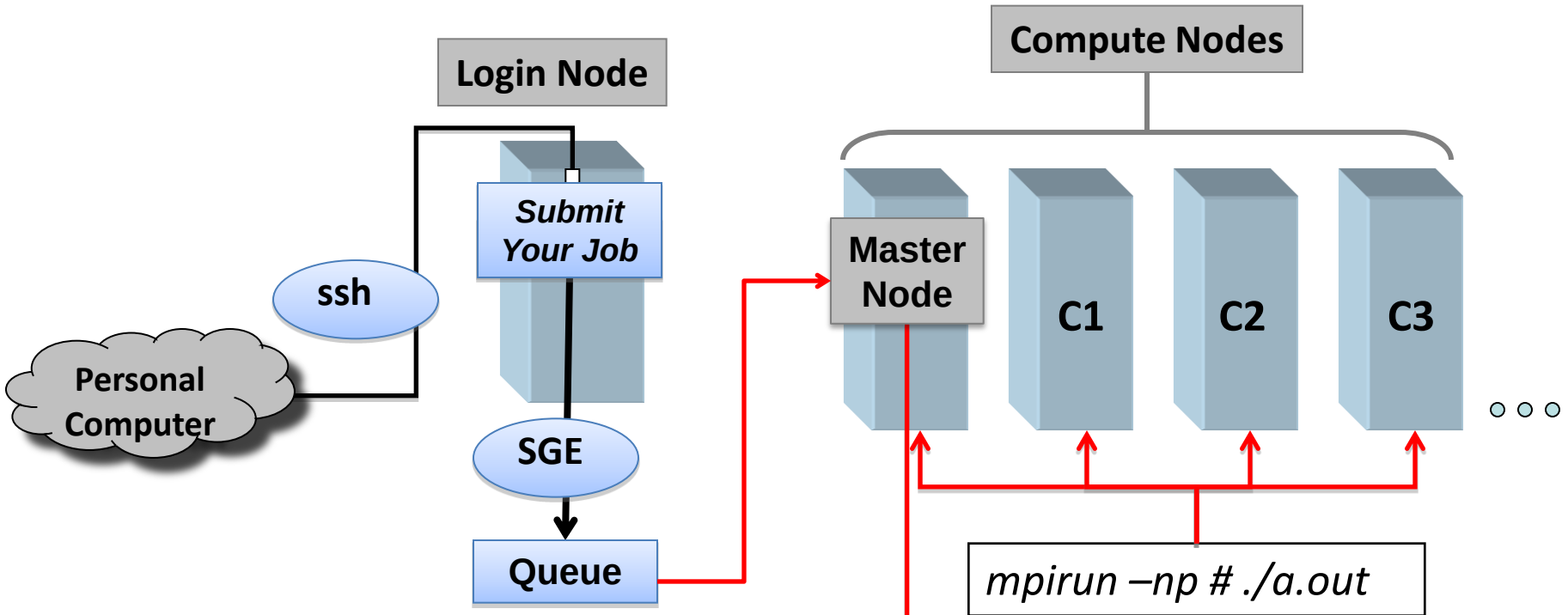
8. Batch Job Submission on Sun Grid Engine

Batch Submission Process



Queue: Job script waits for resources.
Master: Compute node that executes the job script, launches all MPI processes.

Batch Submission Process



Queue: Job script waits for resources.
Master: Compute node that executes the job script, launches all MPI processes.

Batch Job Submission

1. Write a script

```
#!/bin/sh
echo Starting job
date
/usr/bin/time ./hello
date
echo Ending job
```

2. Include scheduler information

```
#!/bin/sh
#$ -N hello
#$ -cwd
#$ -o $JOB_NAME.o$JOB_ID
#$ -j y
#$ -q development
#$ -pe 1way 12
#$ -V
#$ -l h_rt=00:2:00
echo Starting job
date
/usr/bin/time ./hello
date
echo Ending job
```

3. Submit

Batch Job Script

Scheduler Information: Followed by "#$"

- Queue Information (-q)
- Resource Allocation (-pe)
- Inherit current environment (-V)
- Wall Time Limit (-l)
- Account Number (-A)
- More Scheduler Information

Running Script : How program is run.

An example batch script

```
#!/bin/sh
#$ -N hello
#$ -cwd
#$ -o $JOB_NAME.o$JOB_ID
#$ -j y
#$ -q development
#$ -pe 1way 12
#$ -V
#$ -l h_rt=00:2:00
#$ -A TG-TRA120006
echo Starting job
date
/usr/bin/time ./hello
date
echo Ending job
```

Submit a Job

```
cd $HOME/envi/intro
ls -la
cat Makefile                # Review the makefile
make                        # Compile hello.c
ls –la                      # Take a look at what compiled
cat job.sge                 # View the script (q)

        (Add #$ -A TG-TRA120006  to the scheduler
        information if you have more than 1 account)

qsub job.sge                # submit the job
showq -u                    # View queue information
cat hello.o[jobID] | more   # View the job output
```

States

- Unscheduled – Likely not good
- DepWait – You can ask that one job run after another finishes.
- w(aiting) – Queued, waiting for resources to run.
- r(unning) – As far as SGE is concerned, it's going.
- h(old)
- s(uspended)
- E(rror)
- d(eletion)

Queue Examples

```
$qconf -sql
development
gpu
grace
grace-serial
largemem
normal
request
serial
stci
sysdebug
systest
vis
```

- Slots = number of cores
- 12 cores/node on Lonestar (16 on Ranger)
- pe = wayness, how many cores per node
- Job is killed if over time limit

```
$qconf -sq development | more
qname         development
qtype         BATCH INTERACTIVE
pe_list       12way 11way 10way 8way 6way 4way 2way 1way
slots         12
tmpdir        /tmp
…
```

Wayness: *#$ -pe option*

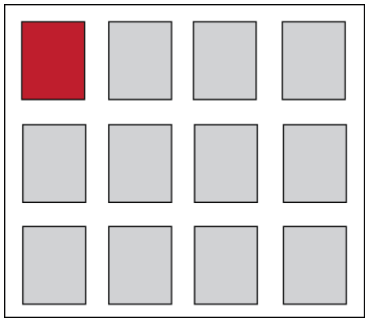
```
#!/bin/sh
#$ -N hello
#$ -cwd
#$ -o $JOB_NAME.o$JOB_ID
#$ -j y
#$ -q development
#$ -pe 1way 12
#$ -V
#$ -l h_rt=00:2:00
#$ -A TG-TRA120006
echo Starting job
date
/usr/bin/time ./hello
date
echo Ending job
```

Wayness

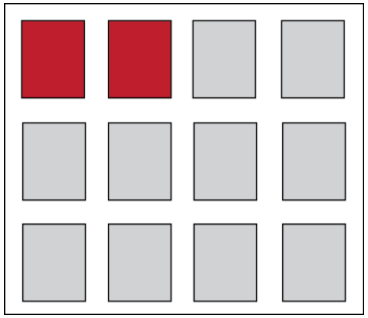
-pe <n>way <m>

- ***n*** **– Number of tasks per node**
- ***m*** **– Number of cores requested**
  - One node on Lonestar, m=12
  - Two nodes on Lonestar, m=24

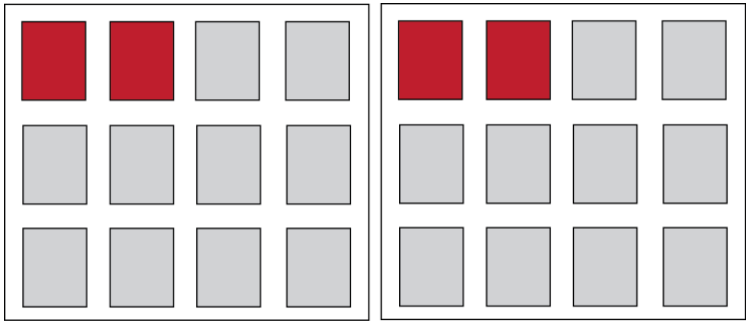
-pe 1way 12
(1 task on one node)



-pe 2way 12
(Run 2 task/node in on 1 node)



-pe 2way 24
(Run 2 task/node in parallel on 2 nodes)



Serial Job

```
#!/bin/sh
#$ -N hello
#$ -cwd
#$ -o $JOB_NAME.o$JOB_ID
#$ -j y
#$ -q development
#$ -pe 1way 12
#$ -V
#$ -l h_rt=00:2:00
echo Starting job
date
/usr/bin/time ./hello
date
echo Ending job
```

MPI Job

```
#!/bin/bash
#$ -N myMPI
#$ -cwd
#$ -o $JOB_NAME.o$JOB_ID
#$ -j y
#$ -q development
#$ -pe 12way 24
#$ -V
#$ -l h_rt=01:30:00
echo Starting job
date
ibrun ./mpihello
date
echo Ending job
```

SGE: Memory Limits

- Default parallel job submission allocates all compute cores per node (12 on Lonestar, 16 on Ranger)
- If you need more memory per MPI task, request fewer cores per node by using one of the 'Nway' environments
- Even if you only launch 1 task/node, you will be charged for the entire node.

Lonestar Parallel environment	Description
12way	12 tasks/node, 2 GB/task
8way	8 tasks/node, 3 GB/task
4way	4 tasks/node, 6 GB/task
2way	2 tasks/node, 12 GB/task
1way	1 task/node, 24 GB/task

Submitting a Parallel Job

```
% cd $HOME/envi/batch
% ls -la
% mpif90 –O3  mpihello.f90 –o mpihello
          OR
% mpicc  –O3  mpihello.c –o mpihello
% cat job                    (need to edit account?)
% qsub job
% watch showq -u -l          (Ctrl-C to quit watching)
% vi job                     (add “sleep 60”)
% qsub job                   (note the returned jobid)
% qdel jobid
```

9. Batch Job Submission: SLURM

Batch on Stampede: SLURM

- Simple Linux Utility for Resource Management (SLURM)
- Open source, fault-tolerant, and highly scalable cluster management and job scheduling system for Linux clusters
- Three key functions:
  - it allocates exclusive and/or non-exclusive access to resources
  - it provides a framework for starting, executing, and monitoring work
  - it arbitrates requests by managing the pending queue

Batch on Stampede: SLURM Commands

Select commands:

- **showq** - view summary of jobs in the batch system (not SLURM native)
- **sacct** - report job or job step accounting information.
- **salloc** - allocate resources for a job in real time.
- **sbatch** - submit a job script for later execution.
- **sbcast** - transfer a file from local disk to local disk on the job nodes.
- **scancel** - cancel a pending or running job or job step.
- **sinfo** - reports the state of partitions and nodes managed by SLURM.
- **squeue** - reports the state of jobs or job steps.
- **srun** - submit an interactive job

Man pages exist for all SLURM daemons, commands, and API functions. The command option--help also provides a brief summary of options. Note that the command options are all case insensitive.

Batch on Stampede: SLURM Commands

1. Use **sinfo** to list queues, nodes, and system state
2. Issue **showq** to show all queued jobs
3. Issue **srun** to run simple commands (e.g. an interactive shell)
   $ srun -p devel --pty /bin/bash –l
4. Issue **sbatch** to submit a batch script

```
$ cat my.script
#!/bin/bash
#SBATCH -J myMPI
# Job name
#SBATCH -o myjob.%j.out   # stdout file (%j expands to jobId)
#SBATCH -p devel          # Queue name
#SBATCH -N 2   # Total number of nodes requested (16 cores/node)
#SBATCH -n 32  # Total number of mpi tasks requested
#SBATCH -t 01:30:00 # Run time (hh:mm:ss) - 1.5 hours
ibrun ./a.out
$ sbatch my.script
sbatch: Submitted batch job 469
```

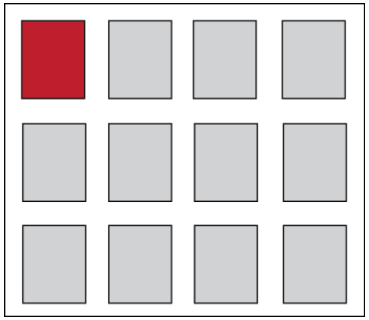
5. Issue **squeue** to see the job status
6. Run **scancel** to cancel the job

These general examples are from the SLURM documentation, and may not exactly reflect implementation on Stampede.

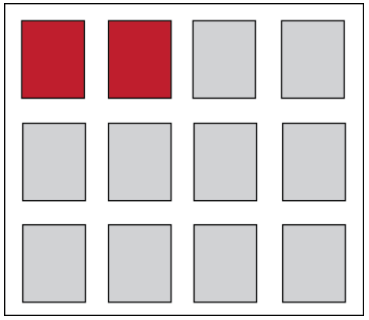
Resource Allocation on SLURM

- ***-N*** **– Number of node requested**
- ***-n*** **– Number of tasks to run**

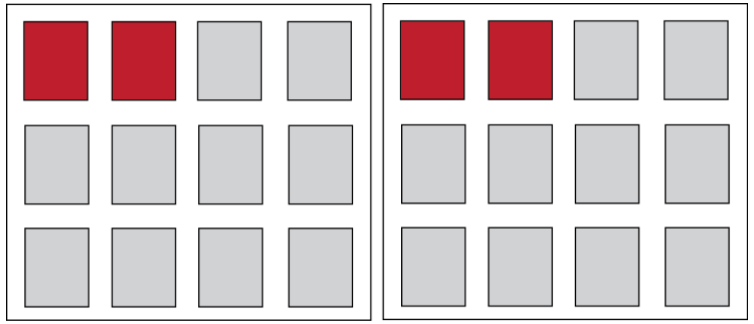
Serial Job
#SBATCH -N 1
#SBATCH -n 1



2 Tasks
#SBATCH -N 1
#SBATCH -n 2



4 Tasks Parallel
#SBATCH -N 2
#SBATCH -n 4



10. Additional Labs

Makefiles

% cd $HOME/envi/using_makefiles	
% cat Makefile	Read over the Makefile
% make	Compile the program, generate a.out
% make	Reports “up to date”, i.e. not recompiled
% touch suba.f	Renew the file timestamp
% make	suba.f (and only suba.f) is recompiled

Precision

The precision program computes and prints *sin(π)*.
The *π* constant uses “E” (double precision) format in one case and “D” (single) in the other.

```
% cd $HOME/envi/precision
% cat precision.f
% module load intel
% ifort –FR precision.f
 (or)
% ifort precision.f90
% ./a.out
```

(The ifc compiler regards “.f” files as F77 fixed format programs. The –FR option specifies that the file is free format.)

Questions?

- CAC
  help@cac.cornell.edu
- XSEDE
  - portal.xsede.org -> Help (in the navigation bar)
  - portal.xsede.org -> My XSEDE -> Tickets
  - portal.xsede.org -> Documentation -> Knowledge Base