PHS Shared Compute Environments

For high performance computing

Introduction

Keith Jose

Information Systems Manager, Public Health Sciences

kljose@ucdavis.edu

530-752-8055

Comparison Overview

Mercury

- Stand-Alone Single Server
- No Workload Manager/Scheduler
- Large Attached Storage

Shiva

- High Performance Compute Cluster
- 4 compute nodes
- Dedicated GPU node
- SLURM workload manager/scheduler use is **required**

How to request access and connect

- User accounts on both Mercury and Shiva are local to those servers.
- You must request an account on one or both of those systems by email.
- Please send an email to kljose@ucdavis.edu and CC dmrocke@ucdavis.edu with your request.

- Both Mercury and Shiva support **SSH** as the main method of connecting

  Windows: **Putty** is a good choice

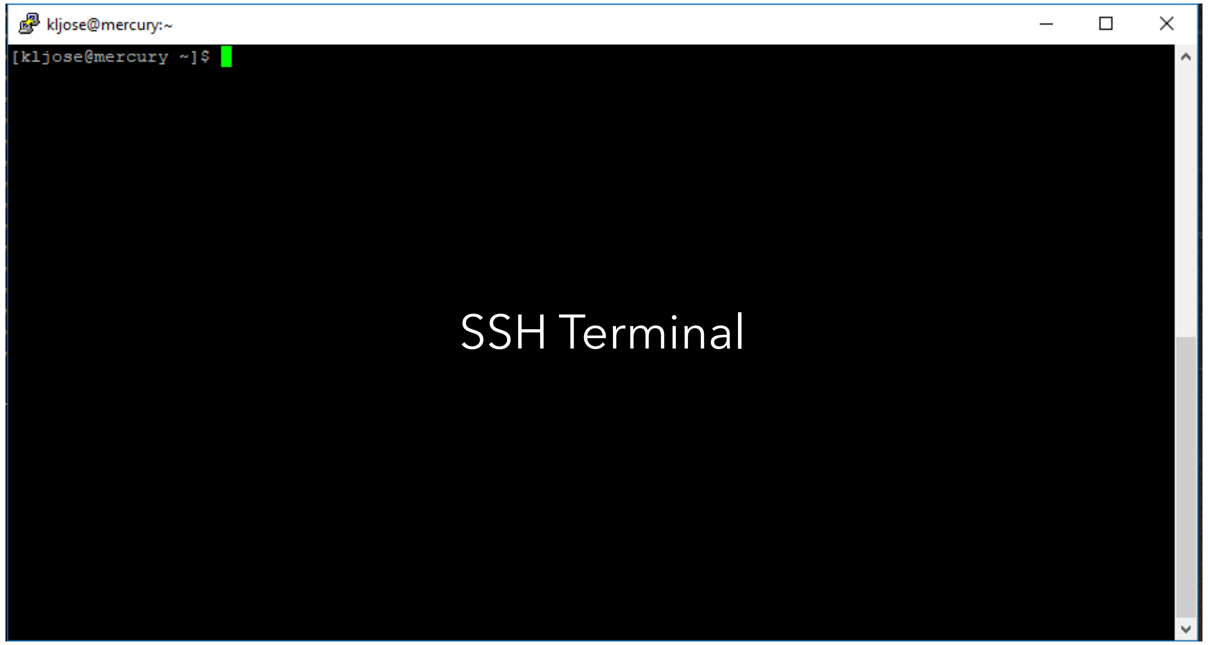
  Mac OS: Native terminal

- When using SSH, it is a command line interface in CentOS Linux. SSH connections can be made from any worldwide Internet connected location.

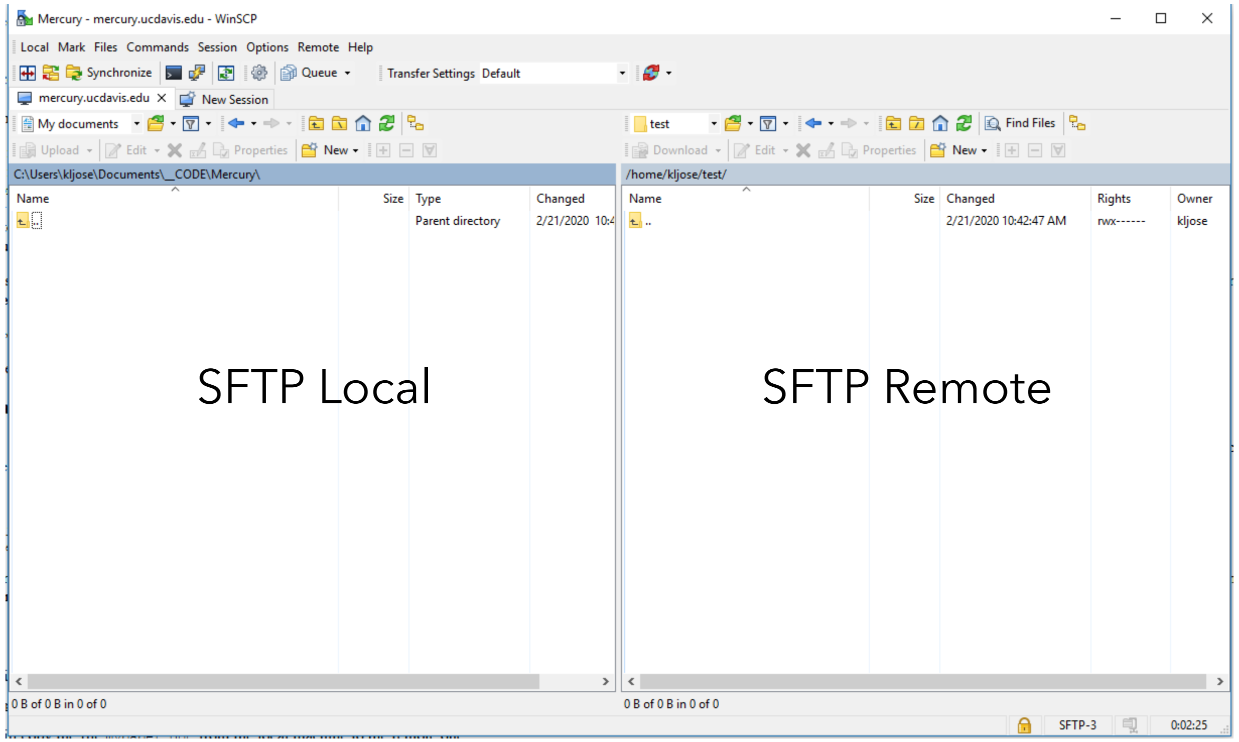
- Host IP addresses for the systems to connect with SSH are including in your account creation email.
- SAS Studio and R Studio have Web-based interfaces. When you receive your account creation email, information about those Web addresses will be included.

Connecting, Transferring, and Managing Files

- When you log into either of the systems using SSH, you will be placed in your home directory, and on both Mercury and Shiva are by default protected so only you can read content in your home directory.
- Your home directory is also located in the area where the main storage for the system exists.
- To transfer files to your home directory, you will need to use SFTP, which utilizes SSH.
- Windows: WinSCP or FileZilla are good choices for graphical SFTP. Bitvise is also good.
- Mac OS: Filezilla or command line using Terminal



```
[kljose@mercury ~]$
```



C:\Users\kljose\Documents\__CODE\Mercury\

Name	Size	Type	Changed
..		Parent directory	2/21/2020 10:4

0 B of 0 B in 0 of 0

/home/kljose/test/

Name	Size	Changed	Rights	Owner
..		2/21/2020 10:42:47 AM	rwx------	kljose

0 B of 0 B in 0 of 0

0:02:25

Basic Linux Commands

pwd - When you first open the terminal, you are in the home directory of your user. To know which directory you are in, you can use the **“pwd”** command. It gives us the absolute path, which means the path that starts from the root. The root is the base of the Linux file system.

ls - Use the **"ls"** command to know what files are in the directory you are in. You can see all the hidden files by using the command **“ls -a”**.

cd - Use the **"cd"** command to go to a directory.

mkdir & rmdir - Use the **mkdir** command when you need to create a folder or a directory. Use **rmdir** to delete a directory. But **rmdir** can only be used to delete an empty directory. To delete a directory containing files, use **rm**.

passwd - To change your password.

tar - Use **tar** to work with tarballs (or files compressed in a tarball archive) in the Linux command line.

zip , unzip - Use **zip** to compress files into a zip archive, and **unzip** to extract files from a zip archive.

mv - Use the **mv** command to move files through the command line.

find - The **find** command is used to locate a file in a Linux system

chmod - Use **chmod** to make a file executable and to change the permissions granted to it in Linux.

man - To know more about a command and how to use it, use the **man** command.

Security & Restrictions

- **No root / sudo access**
- **SAS Studio and R Studio restricted to UCD/UCDH**
- **PHI/PII must be kept encrypted (at rest) –** If you are working with any kind of identifiable information, please utilize **Veracrypt** to store your data, and only have the data accessible from Veracrypt while you are actively working with it.
- **Key-based authentication is encouraged –** This is a public/private keypair that you can create, and you can configure Putty or OS X terminal to use your keypair to log in. This will disable password authentication on your account and greatly increase authentication security. You **must** keep your private key private. Information on how to set this up will be included in your account creation email, or you can email me kljose@ucdavis.edu and I can send you some steps to set this up.

Veracrypt Compliance:
•SO/IEC 10118-3:2004 [21]
•FIPS 197 [3]
•FIPS 198 [22]
•FIPS 180-2 [14]
•FIPS 140-2 (XTS-AES, SHA-256, SHA-512, HMAC) [25]
•NIST SP 800-38E [24]
•PKCS #5 v2.0 [7]
•PKCS #11 v2.20 [23]

Mercury

Mercury

Stand-Alone Server

Hardware

- Xeon E7 @ 2.1 GHz: Total of **96 processing cores available for computation**
- **88TB of available disk storage**
- **1TB of available memory**

Operating System

- CentOS 7.7 (based on RHEL)

Mercury: Environment and Tools

- Install software locally, if possible
- System maintenance happens regularly, will notify.
- **No backups. Please plan accordingly.**

Maintained Packages & Tools:

- SAS
- SAS Studio
- R
- R Studio
- Python
- C/C++
- Vi / Emacs
- Veracrypt

Mercury: running jobs: SAS

Ensure the data you need for your job is in on the server

SAS Batch Mode: Example shell script to run a single SAS program:

mySASProgram.sh

```
#!/usr/bin/sh
dtstamp=$(date +%Y.%m.%d_%H.%M.%S)
pgmname=“./sas/code/project1/program1.sas"
logname=“./sas/code/project1/program1_$dtstamp.log"
/usr/local/SASHome/SASFoundation/9.4/sas $pgmname -log $logname
```

Ensure the script is executable:
```
$ chmod 750 ./mySASProgram.sh
```

Run the script:
```
$ ./mySASProgram.sh
```

More information on SAS Batch Mode: https://bit.ly/2T2RNMZ

SAS Studio: SAS Studio is a Web-based version of SAS that shows a familiar user interface as PC SAS. All data you load or create from within SAS Studio resides on the server. You must use SFTP to transfer data to and from. SAS Studio is only accessible from UCD/UCDH network locations.

Mercury: Installing R Packages

R is installed with some base packages, but likely not with everything you may need.

R packages should be installed locally in your home directory. The following is an example script that would install packages.

install.packages.R

```
pkgvec <- scan("R.pkgs.txt",what=character())
biovec <- scan("Bio.pkgs.txt",what=character())
install.packages(pkgvec,repos="http://cran.cnr.Berkeley.edu")
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite(pkgs=biovec)
```

where R.pkgs.txt contains the names of R packages and Bio.pkgs.txt contains the names of bioconductor packages.

Packages will reside in your home directory in ~/R/x86_64-redhat-linux-gnu-library/[R Version]

Mercury: running jobs: R

R Program Console: Just type R at your command prompt to start R.

```
$ R
```

R Batch Mode: Specify your R script in the command.

myScript.R

```
---------------------
set.seed(1)
M<-matrix(runif(20),5,4)
write.csv(M, file="M.csv")
```

```
$ R CMD BATCH myScript.R
```

R Studio: R Studio is a Web-based version of R with GUI. Note that files you see in the Web interface reflect files on the server, not your local computer. You must always ensure data files or programs are on the server by copying them using SFTP.

Shiva

Shiva

Compute Cluster

Hardware

- Head/Storage Node + 4 Compute Nodes
- Xeon Gold 5118 @ 2.3 GHz on ALL compute nodes. Each compute node having 48 processing cores, for a total of **192 processing cores available for computation** across the cluster.
- Each compute node has 754GB of memory, providing a total of **over 3TB of available memory for computation** across the cluster.
- **43TB of available disk storage**
- GPU Node: **Discrete Nvidia Tesla V100 GPU**
- Normal compute nodes do not have discrete GPU
- **Infiniband** connected nodes for high speed CPU communications between nodes (56.25Gbps communication)

Operating System

- CentOS 7.7 (based on RHEL)
- OpenHPC software stack

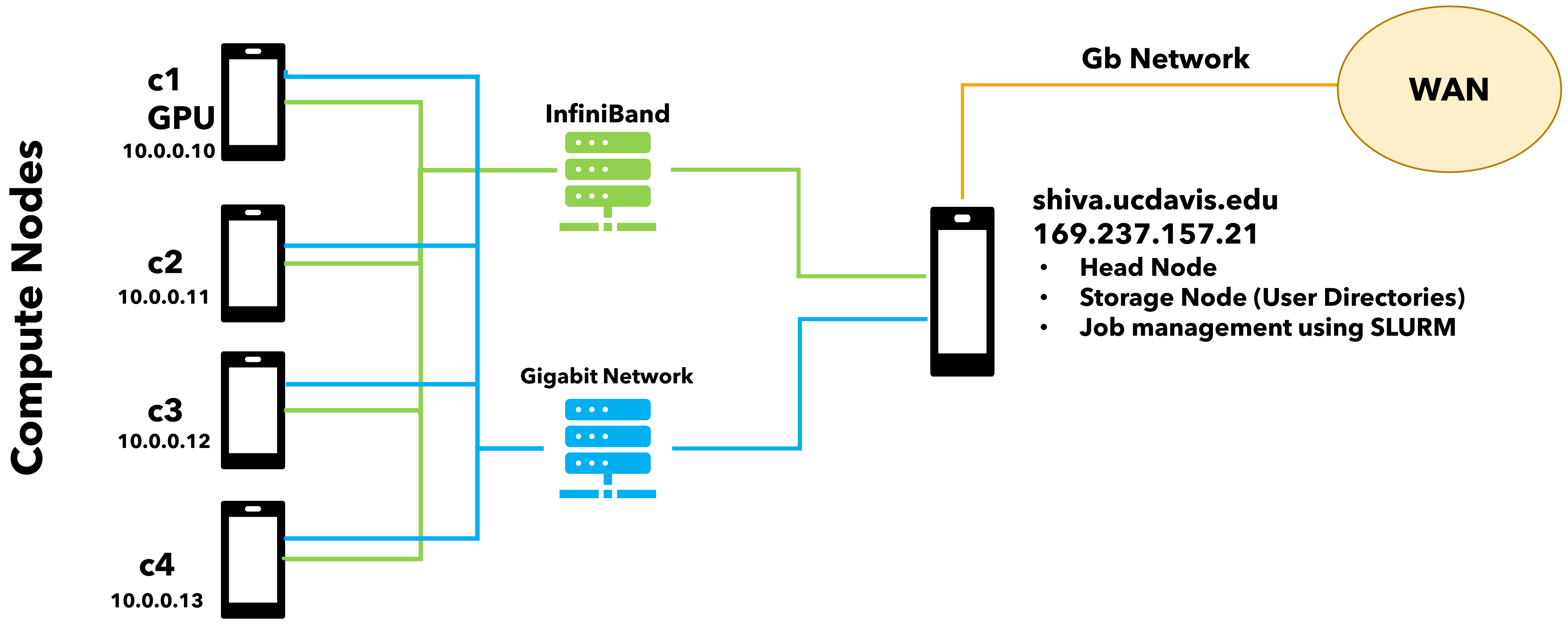
Shiva: Environment and Tools

- Install packages locally, if possible.
- Ideal for parallel computing
  - OpenMPI
  - Infiniband
  - Discrete Nvidia Tesla V100 and Cuda Toolkit
- System maintenance happens regularly, will notify.
- **No backups. Please plan accordingly.**

Maintained Packages & Tools:

- SLURM
- R
- R Studio
- Python3
  - mpi4py
- C/C++
- Vi / Emacs
- Veracrypt
- OpenMPI
- Infiniband
- Cuda Toolkit (GPU node)

Shiva: Architecture



Shiva: What is SLURM?

SLURM: Open source, fault-tolerant, and highly scalable cluster management and job scheduling system.

You must learn the basics of SLURM to use Shiva.

All work must be submitted via SLURM. Do not run jobs on head node!

SLURM QuickStart: https://slurm.schedmd.com/quickstart.html

Key Functions of SLURM:

1) It allocates exclusive and/or non-exclusive access to compute nodes to users for some duration of time so they can perform work
2) It provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes
3) It arbitrates contention for resources by managing a queue of pending work.

Shiva: Using SLURM

Job Submission

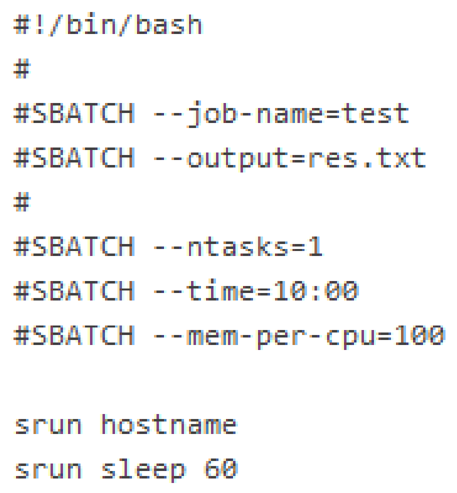
salloc – Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished.

sbatch – Submit a batch script to Slurm.

srun – Run parallel jobs

Simple script using srun

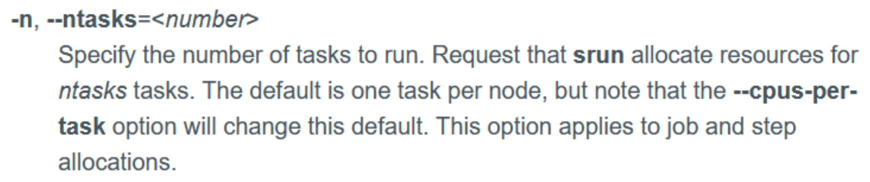
-------------- *submit.sh*



```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

srun hostname
srun sleep 60
```

← would request one CPU for 10 minutes, along with 100 MB of RAM, in the default queue. When started, the job would run a first job step srun hostname, which will launch the UNIX command hostname on the node on which the requested CPU was allocated. Then, a second job step will start the sleep command. Note that the --job-name parameter allows giving a meaningful name to the job and the --output parameter defines the file to which the output of the job must be sent.

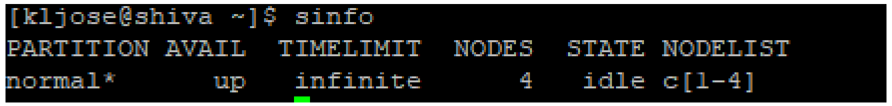


-n, **--ntasks**=<*number*>

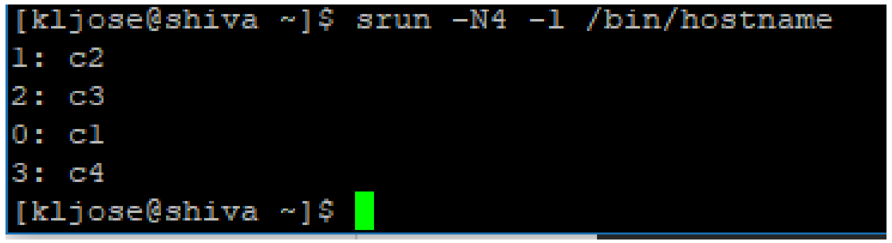
Specify the number of tasks to run. Request that **srun** allocate resources for *ntasks* tasks. The default is one task per node, but note that the **--cpus-per-task** option will change this default. This option applies to job and step allocations.

All options for **srun:** https://slurm.schedmd.com/srun.html

Shiva: Slurm Examples



```
[kljose@shiva ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   infinite      4   idle c[1-4]
```



```
[kljose@shiva ~]$ srun -N4 -l /bin/hostname
1: c2
2: c3
0: c1
3: c4
[kljose@shiva ~]$
```

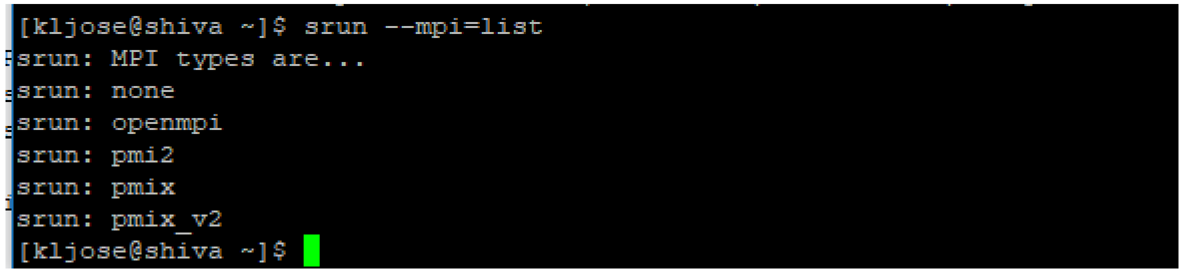
```
adev0: cat my.script
#!/bin/sh
#SBATCH --time=1
/bin/hostname
srun -l /bin/hostname
srun -l /bin/pwd

adev0: sbatch -n4 -w "adev[9-10]" -o my.stdout my.script
sbatch: Submitted batch job 469

adev0: cat my.stdout
adev9
0: adev9
1: adev9
2: adev10
3: adev10
0: /home/jette
1: /home/jette
2: /home/jette
3: /home/jette
```

Shiva: MPI / InfiniBand

Message Passing Interface



```
[kljose@shiva ~]$ srun --mpi=list
srun: MPI types are...
srun: none
srun: openmpi
srun: pmi2
srun: pmix
srun: pmix_v2
[kljose@shiva ~]$
```

https://slurm.schedmd.com/mpi_guide.html ← LOTS of information here!

Using pmix support:

$ srun **-mpi=pmix** –n 4 a.out

Shiva: MPI / InfiniBand

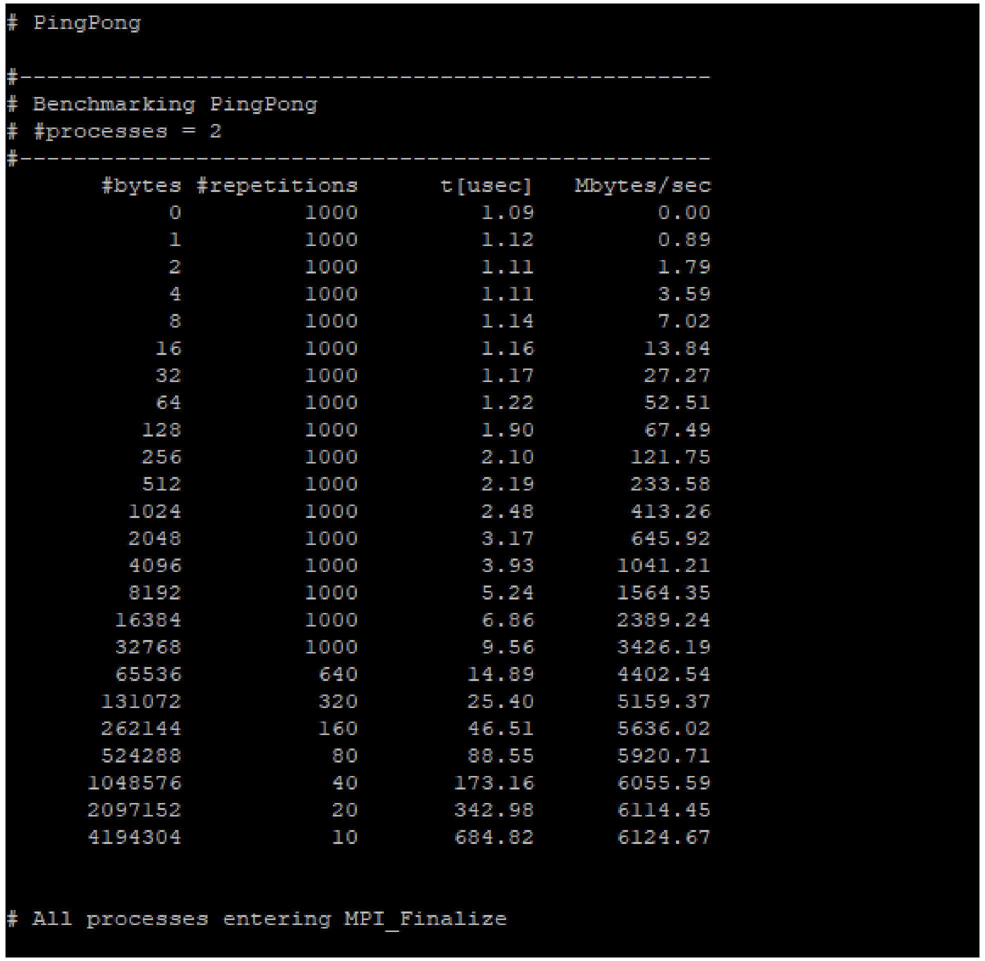
Quick benchmark to verify InfiniBand

```
kljose@shiva$ module load imb

kljose@shiva$ srun -N 2 -pty /bin/bash

kljose@c1$ prun IMB-MPI1 PingPong
```

Shows rate over Infiniband →



```
# PingPong

#---------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#---------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         1.09         0.00
            1         1000         1.12         0.89
            2         1000         1.11         1.79
            4         1000         1.11         3.59
            8         1000         1.14         7.02
           16         1000         1.16        13.84
           32         1000         1.17        27.27
           64         1000         1.22        52.51
          128         1000         1.90        67.49
          256         1000         2.10       121.75
          512         1000         2.19       233.58
         1024         1000         2.48       413.26
         2048         1000         3.17       645.92
         4096         1000         3.93      1041.21
         8192         1000         5.24      1564.35
        16384         1000         6.86      2389.24
        32768         1000         9.56      3426.19
        65536          640        14.89      4402.54
       131072          320        25.40      5159.37
       262144          160        46.51      5636.02
       524288           80        88.55      5920.71
      1048576           40       173.16      6055.59
      2097152           20       342.98      6114.45
      4194304           10       684.82      6124.67


# All processes entering MPI_Finalize
```

Shiva: R & Slurm – Single CPU

Source: https://rcc.uchicago.edu/docs/software/environments/R/index.html

```
------ Rhello.sbatch

#!/bin/sh

#SBATCH --partition=broadwl

#SBATCH --tasks=1

# Load the default version of hello.

module load R

# Use R CMD BATCH to run Rhello.R.

R CMD BATCH --no-save --no-restore Rhello.R
```

```
----- Rhello.R

print("Hello World")
```

Shiva: R & Slurm – Multicore – Single Node

Source: https://rcc.uchicago.edu/docs/software/environments/R/index.html

```
-------- doParallel.sbatch

#!/bin/bash

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=16

module load R

R CMD BATCH --no-save –-no-restore doParallel.R
```

```
------ doParallel.R (Requires Module doParallel)

library(doParallel)
# use the environment variable SLURM_NTASKS_PER_NODE to set the number of cores
registerDoParallel(cores=(Sys.getenv("SLURM_NTASKS_PER_NODE")))
# Bootstrapping iteration example
x <- iris[which(iris[,5] != "setosa"), c(1,5)]
iterations <- 10000 # Number of iterations to run
# Parallel version of code
# Note the '%dopar%' instruction

parallel_time <- system.time({
    r <- foreach(icount(iterations), .combine=cbind) %dopar% {
    ind <- sample(100, 100, replace=TRUE)
    result1 <- glm(x[ind,2]~x[ind,1], family=binomial(logit))
    coefficients(result1)
  }
})

# Shows the number of Parallel Workers to be used
getDoParWorkers()

# Prints the total compute time.
parallel_time["elapsed"]
```

Shiva: R & Slurm – MultiNode

Source: https://rcc.uchicago.edu/docs/software/environments/R/index.html

```
----- Rmpi.sbatch

#!/bin/sh

#SBATCH --nodes=4

#SBATCH --time=1

#SBATCH --exclusive

module load R

# The openmpi module is not loaded by default with R.

module load openmpi3

# Always use -n 1 for the Rmpi package. It spawns additional
processes

# dynamically

mpirun -n 1 R CMD BATCH --no-save --no-restore Rmpi.R
```

```
------- Rmpi.R (Requires Module Rmpi)

library(Rmpi)

# initialize an Rmpi environment

ns <- 4

mpi.spawn.Rslaves(nslaves=ns)

# send these commands to the slaves

mpi.bcast.cmd( id <- mpi.comm.rank() )

mpi.bcast.cmd( ns <- mpi.comm.size() )

mpi.bcast.cmd( host <- mpi.get.processor.name() )

# all slaves execute this command

mpi.remote.exec(paste("I am", id, "of", ns, "running on", host))

# close down the Rmpi environment

mpi.close.Rslaves(dellog = FALSE)

mpi.exit()
```

Resources

Slurm: https://slurm.schedmd.com/documentation.html

OpenMPI: https://www.open-mpi.org/

Linux Reference Sheet: https://files.fosswire.com/2007/08/fwunixref.pdf

SAS Batch Mode: https://bit.ly/2T2RNMZ

R & Slurm Examples:
https://rcc.uchicago.edu/docs/software/environments/R/index.html