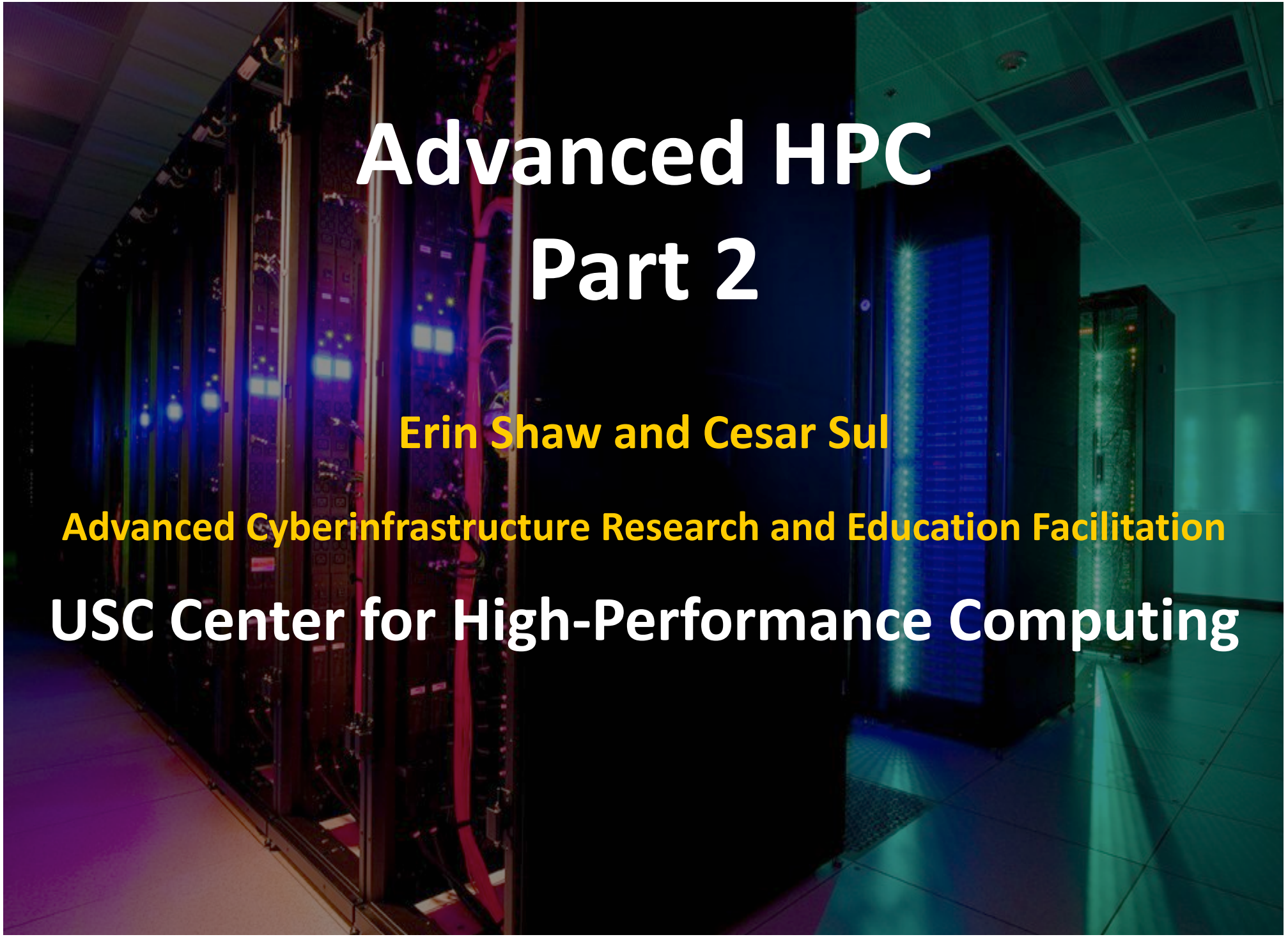

Advanced HPC
Part 2

Erin Shaw and Cesar Sul

Advanced Cyberinfrastructure Research and Education Facilitation

USC Center for High-Performance Computing

Outline

- File System
- Queues and partitions
  - Scavenge partition
  - Checkpointing
- Node allocation table
- Handy Slurm commands
  - Analyzing your job

HPC File Systems

- Permanent directories
  - Home: /home/rcf-xx
  - Project: /home/rcf-proj
  - Note:
    - Backed up every night
    - Nightly: For ONE week only!
    - Weekly: For 4 weeks only!
    - Send URGENT email to
- Temporary disk space
  - /staging
  - $TMPDIR (per node)
  - Note:
    - No backups!

- Using temporary disk space

```
# Start or submit job from staging
$ cd /staging/<proj>

# Copy data to staging
$ cp /home/rcf-proj/<proj>/<user>/datafiles .

# Run job

# Zip up and copy back (w/name, date, jobid)
$ tar cf results.tar myjob.sl results/
$ cp results.tar /home/rcf-proj/<proj>/<user>
```

File System Benchmarks

- Why use? Temporary disk areas are fast.
- Benchmark procedure

  Use **dd** command to measure the speed of a 1GB write on the project and staging directories. Benchmark requires 3g memory.

  $ salloc –p scavenge –n 1 --constraint=IB --time=1:00:00 --mem-per-cpu=3g

  $ man df #report file system disk space usage

  $ df

Filesystem	Used	Available	Use%	Mounted on
almaak-08:/export/samfs-proj2/proj	128683241472	43202379776	75%	/auto/rcf-proj
beegfs_nodev	1230624575488	120916893696	66%	/staging

  $ man dd #convert and copy a file

  $ rundd.sh #shell script with dd command (./rundd.sh)

File System Benchmarks

- Benchmark project directory

[hpc3529]$ cd /home/rcf-proj/ess/erinshaw/tmp
[hpc3529]$ dd if=/dev/zero of=fileOfzeros bs=1G count=1
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 10.8122 s, 99.3 MB/s

- Benchmark staging directory

[hpc3529]$ cd /staging/ess/erinshaw/tmp
[hpc3529]$ dd if=/dev/zero of=fileOfzeros bs=1G count=1
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB) copied, 0.735967 s, 1.5 GB/s

Queues and Partitions

Partition Name	Max Nodes/CPUs	Max Time	Max Memory	Max Jobs Running	Max Jobs Submitted	Available Resources *notes
main *(default)*	99 nodes	24 hours	–	10	2000	771 nodes (14,972 cpus)
quick *(default)*	4 nodes	1 hour	–	10	200	118 nodes (15,116 cpus) 7 dedicated nodes
large *(default)*	256 nodes	24 hours	–	1	10	771 nodes (14,972 cpus)
long *(default)*	1 node	14 days (336 hours)	–	1	30	9 nodes (108 cpus) 9 dedicated nodes
scavenge* *(–p scavenge)*	500 cpus	7 days (168 hours)	2500G	–	500	502 nodes (5268 cpus) ***nodes are preemptable** *nodes are shared *no HSDA jobs
largemem *(–p largemem)*	1 node	14 days (336 hours)	–	1	10	4 nodes (160 cpus)

Queues and Partitions

- Understand your partitions quiz!
  - When should you use largemem?
  - What are the pros and cons of using scavenge?
  - On which partitions can you run a job for over 24 hours?
  - What scheduling queues (partitions) will these jobs be submitted to?

```
#SBATCH --ntasks=2
#SBATCH --time=1:00:00
```

```
#SBATCH --ntasks=200
#SBATCH --time=23:00:00
```

```
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --time=10-00:00:00
```

```
#SBATCH --partition=largemem
#SBATCH --ntasks=60
#SBATCH --time=1:00:00
```

```
#SBATCH --partition=scavenge
#SBATCH --ntasks=500
#SBATCH --time=8-00:00:00
```

Scavenge Partitions

- See
  https://hpcc.usc.edu/support/documentation/scavenge/
- Why sweet?
  - Really fast allocation!
- Why important?
  - Sharing nodes will increase cluster utilization.
  - It's the way of the future for HPC.
- Why use cautiously?
  - Preemption will occur and needs to be planned for.
  - Time to understand checkpointing.

Scavenge Checkpointing

- Long jobs run the risk of ending before completing
  - If job is longer than maximum wall time
  - If job is preempted
- Different ways to recover and requeue
  - Save state (save values of variables and reload each time)
    - Requires being able to change code
  - DMTCP: Distributed checkpointing
    - Saves the running state of any program
- See

  https://hpcc.usc.edu/support/documentation/checkpointing/

Scavenge Checkpointing

- Copy workshop example directory ckpt/

  cp –r /home/rcf-proj/workshop/adv-hpc/ckpt/ .

- Follows

  https://hpccpreview.usc.edu/documentation/checkpointing-restore/

Why isn't my job running?

- There are many reasons that your job may have to wait in the queue longer than you would like.
  - System load is high. It's frustrating for everyone!
  - Your project does not have sufficient ~~computing~~ or space resources
    - Use ~~$mybalance~~ and $myquota
- Your job is requesting
  - Specialized resources, such as the large memory queue or certain software licenses, that are limited or in high demand.
  - A lot of resources. It takes time for the resources to become available.
  - Incompatible or nonexistent resources (may never run).
  - Resource requests that restrict the nodes where the job can run, for example by requesting mem=25GB on a system where most of the nodes have 24GB.

Why isn't my job running?

- Job may also be unnecessarily stuck in batch hold because of system problems
  - Do not hesitate to email hpc@usc.edu
- Slurm will tell you why your job is pending

  $squeue | grep PD
  - Dependency
  - Priority
  - JobHeldUser
  - Resources
- See

  https://slurm.schedmd.com/squeue.html
  - Search for: JOB REASON CODES

Why isn't my job running?

- But there are soooo many idle nodes!

  $ sinfo --partition=main

- Remember that other jobs are waiting for resources, too
  - Large jobs often queue for a long time
  - Slurm internally reserves the resources the job will need
  - These resources remain in state "idle"
    - but they are prioritized for a previously-submitted job

Node Allocation Table

- See hpc_node_allocation_table_201907.pdf
  under /home/rcf-proj/workshop/handouts/

Handy Slurm Commands

- See handy_slurm_commands_201907.pdf
  under /home/rcf-proj/workshop/handouts/

Job utilization

- Two commands can help you analyze your utilization after it runs
  - Slurm's sacct and seff

$ sacct -a --format JobID,User,Group,State,ExitCode,AllocCPUS,NTasks,NNodes,TotalCPU,Elapsed,REQMEM, MaxRSS -j 3958756

JobID	User	Group	State	AllocCPUS	ReqMem	TotalCPU	Elapsed	MaxRSS	ExitCode	NNodes	Ntasks
3958756	tt	hpc	COMPLETED	8	60Gn	04:44:46	04:44:59		0:0	1	
3958756.bat+			COMPLETED	8	60Gn	04:44:46	04:44:59	256380K	0:0	1	
13958756.ext+			COMPLETED	8	60Gn	00:00.002	04:45:01	0	0:0	1	1

- How long did the job run?
- How much memory was used?
- Bonus: How do you monitor cpu and memory *while* running?

Job efficiency

- seff *job_id* #report efficiency of job
  - Efficiency is the amount [of X] that you actually used, relative to the amount [of X] that you requested resources for

```
$ seff 3958756
Job ID: 3958756Cluster: uschp
User/Group: jb_875/med-ar
State: COMPLETED (exit code 0
Nodes: 1
Cores per node: 8
CPU Utilized: 04:44:46CPU
Efficiency: 12.49% of 1-13:59
Job Wall-clock time: 04:44:59
Memory Utilized: 250.37 MB
Memory Efficiency: 0.41% of 6
```

Advice to this user:
=>The number of cpus used is dependent on the code, so this will not change unless you replace functions with parallel versions.
=>The memory used is dependent on the data or complexity of the calculations, so I expect this will increase with your data size.
=>This means that you need only one core but all the memory you can get, since slurm allows you to use all cores on a node, you can request:
--ntasks=1
--mem=60g #increase as needed, per MaxRSS value
--time=24:00:00 #increase as needed, per ElapsedTime and number of sequences.

End Part 2

- Wrap up…
- Questions…