DS 7347
High-Performance Computing (HPC) and Data Science
Session 9

Robert Kalescky
Adjunct Professor of Data Science
HPC Research Scientist

May 24, 2022

Research and Data Sciences Services
Office of Information Technology
Center for Research Computing
Southern Methodist University

Outline

Lab Peer Review

Lab Peer Review

Group Discussion

- Assigned pairs will go to breakout rooms
- Discuss:
  - Progress
  - Problems
  - Ideas

Comments

- Provide a summary of discussion concerning your lab progress and report in `assignments/lab_02.md`; note your peer reviewers name
- Commit `assignments/lab_02.md` to your class repo
- Due 12:00 AM Central, Thursday, May 26, 2022

Session Question

Session Question

Why do we need job schedulers?

ManeFrame II (M2)

ManeFrame II (M2) Node Types

Type	Quantity	Cores	Memory [GB]	Additional Resources
Standard-Memory	176	36	256	
Medium-Memory-1	35	36	768	
Medium-Memory-2	4	24	768	3 TB SSD local scratch
High-Memory-1	5	36	1,536	
High-Memory-2	6	40	1,536	3 TB SSD local scratch
GPGPU-1	36	36	256	NVIDIA P100 GPU has 3,584 CUDA cores and 16 GB CoWoS
MIC-1	36	64	384	16 GB of high bandwidth (400 GB/s) stacked memory
VDI	5	36	256	NVIDIA Quadro M5000 GPU
v100x8	3	36	768	8 NVIDIA V100 GPUs with 5,120 CUDA cores and 32 GB CoWoS
Faculty Partner Nodes	3			Various research specific NVIDIA GPU configurations
ManeFrame II	354	11,276	120 TB	2.8 PB storage and InfiniBand network

ManeFrame II (M2) Partitions (Queues)

Partition	Duration	Cores	Memory [GB]
development	2 hours	various	various
htc	1 day	1	6
standard-mem-s	1 day	36	256
standard-mem-m	1 week	36	256
standard-mem-l	1 month	36	256
medium-mem-1-s	1 day	36	768
medium-mem-1-m	1 week	36	768
medium-mem-1-l	1 month	36	768
medium-mem-2	2 weeks	24	768
high-mem-1	2 weeks	36	1538
high-mem-2	2 weeks	40	1538
mic	1 week	64	384
gpgpu-1	1 week	36	256
v100x8	1 week	1	20
fp-gpgpu-2	various	24	128
fp-gpgpu-3	various	40	384

ManeFrame II File Systems

$HOME	• Default file system when logging into M2, e.g. `/users/$USER`. • Space should be used to write, edit, compile programs, and job submission scripts, etc. • Restricted by quotas (200 GB) and backed-up.
$WORK	• Long term storage at `/work/users/$USER`. • Restricted by quotas (8 TB) and not backed-up.
$SCRATCH	• Scratch space at `/scratch/users/$USER`. • Treat $SCRATCH as a volatile file system that is not backed-up.

Slurm

Common Classes of Applications

Serial

- Users run short-lived single-threaded calculations in batches
- Not uncommon to see batches of greater than 10K
- Batches can be submitted individually, via SLURM's array feature, or bundled to use a whole node

Shared-Memory

- Users run long-lived computationally intensive jobs with various memory requirements
- Not uncommon to see run times of several months

Distributed-Memory

- Users run computationally intensive jobs that span nodes
- Not uncommon to see hundreds of nodes used for single job

Common Classes of Applications

Serial

- Simple calculations frequently done *en masse*
- Usually done to get statistics over many trials
- R, MATLAB, Python, ROOT, Autodock

Shared-Memory

- Complex calculations with demanding compute requirements
- Quantum chemistry and molecular dynamics packages (e.g. CFOUR, Gaussian, NAMD, CHARMM)

Distributed-Memory

- Distributed programs usually via MPI
- Many parallel computing research codes
- Some packages such as CFOUR, CHARMM, LAMMPS

Resource Management

- Compute resources are shared by all users
- Queue systems manage access to specific resources
- There are many queue systems around: Moab, Torque, LoadLeveler, Condor, Oracle Grid Engine, Argent Job Scheduler, Platform LSF, etc.
- ManeFrame uses Simple Linux Utility for Resource Management (SLURM)

Resource Management

- ManeFrame resource allocation is governed by a fair-share factor
  - Preference is given to those who have submitted the fewest jobs
  - Prevents a single user from “blocking” subsequent job submissions of other users
- Allocations can be requested interactively or as a batch job
- Queues (or partitions) are used to distinguish resources by type and the maximum run time

Basic Slurm Commands

Command	Description	Example Usage
`sinfo`	Displays partition and node state information	`sinfo`
`squeue`	Displays queue state information	`squeue -u $USER`
`sbatch`	Submits batch script	`sbatch job.submit`
`srun`	Requests resources for interactive use	`srun -p development --exclusive --pty $SHELL`
`scancel`	Cancel jobs	`scancel 12345678`

Serial Python Script: π Monte Carlo

```python
import random, sys

def monte_carlo_pi(points):
    return 4 * sum(1 for _ in range(points) if random.random()**2 +
    ↪  random.random()**2 < 1) / float(points)

if __name__ == "__main__":
    print(monte_carlo_pi(int(sys.argv[1])))

```

Listing 1: Serial algorithm to estimate the value of π.

Parallel Python Script: π Monte Carlo

```python
import random, sys
import multiprocessing as mp

def check_point(points):
   return sum(1 for _ in range(points) if random.random()**2 + random.random()**2 < 1)

def parallel_monte_carlo_pi(points, cores):
   points_per_core = int(points / cores)
   n = [points_per_core] * cores
   n[0] = points_per_core + (points - (points_per_core * cores))
   pool = mp.Pool(processes = cores)
   results = pool.map(check_point, n)
   return 4 * sum(results) / float(points)

if __name__ == "__main__":
   print(parallel_monte_carlo_pi(int(sys.argv[1]), int(sys.argv[2])))

```

Listing 2: Parallel algorithm to estimate the value of π.

Interactive Sessions

```
module load python
srun -p htc --mem=6G --pty $SHELL

```

Listing 3: Using `srun` to log into a compute node to run commands interactively.

Using `srun`

```
srun -p htc --mem=6G  python pi_monte_carlo.py 1000
```

Listing 4: Using `srun` to run commands directly on a compute node.

Using `sbatch --wrap`

```
sbatch -p htc --mem=6G  --wrap "sleep 30; time python pi_monte_carlo.py 1000"
```

Listing 5: Using `sbatch --wrap` wrap a commands in an `sbatch` script that is then submitted to the queue can run non-interactively.

Using sbatch

```bash
#!/bin/bash
#SBATCH -J python
#SBATCH -o python_%j.out
#SBATCH -p htc
#SBATCH --mem=6G

module purge
module load python

time python pi_monte_carlo.py 1000

```

Listing 6: Using `sbatch` run serial computations via an `sbatch` script.

Using sbatch

```
#!/bin/bash
#SBATCH -J pi
#SBATCH -o pi_%j.out
#SBATCH -p development
#SBATCH -N 1
#SBATCH --cpus-per-task=2
#SBATCH --mem=6G

module purge
module load python

time python pi_monte_carlo_shared.py 10000000 ${SLURM_CPUS_PER_TASK}

```

Listing 7: Using `sbatch` run parallel computations via an `sbatch` script.

Using `sbatch --array`

```bash
#!/bin/bash
#SBATCH -J pi_array
#SBATCH -o pi_array_%a-%A.out
#SBATCH --array=1-4%2
#SBATCH -p development
#SBATCH --mem=6G

module purge
module load python

time python pi_monte_carlo.py $((100**${SLURM_ARRAY_TASK_ID}))

```

Listing 8: Using `sbatch --array` run parallel jobs via a single `sbatch` script.

Readings and Assignments

Readings and Assignments

Readings

None

Readings and Assignments

Assignment

- Write and submit an Slurm job script that does the following:
    1. Uses the “htc“ queue with one node, one core, and 6 GB of memory.
    2. Sends output to `assignment_05.out`.
    3. Runs the Docker image, via Singularity, from the previous assignment `assignments/assignment_04.dockerfile`.
    4. Prints the hostname of the compute node running the job, job ID number of the job, contents of `/proc/cpuinfo`, the number of nodes, number of cores, the amount of memory, and the output of `free -g`, all using Slurm-specific environment variables where possible.
- Commit to your class repo:
    1. `assignments/assignment_05.sbatch`.
    2. `assignments/assignment_05.out`.
- Due 12:00 AM Central, Tuesday, May 31, 2022