Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Slurm Job Management and Applications on Discovery

Content

- Connecting to Discovery
- Cluster File System
- Partitions and Compute Nodes
- Data transfer
- Loading Module
- Linux Fundamentals on Discovery Cluster
- Using Slurm
- Running jobs: interactive mode and batch mode; Job scripts
- Learn how to compile and run OpenMP and MPI programs via interactive mode and batch mode

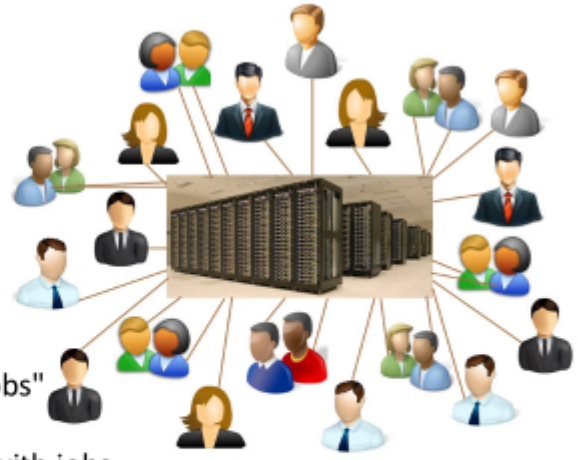
Abstract

- Slurm is a workload manager systems that has been used to *schedule* and *manage* user jobs run on *HPC clusters/supercomputers*.

What is a Workload Manager?

Commonly called a Workload Manager. May also be referred to (sometimes loosely) as:

- *Batch system, Workload scheduler, Job scheduler*



Tasks commonly performed by a Workload Manager:

- Provide a means for users to specify and submit work as "jobs"
- Evaluate, prioritize, schedule and run jobs
- Provide a means for users to monitor, modify and interact with jobs
- Manage, allocate and provide access to available machine resources
- Manage pending work in job queues
- Monitor and troubleshoot jobs and machine resources
- Efficiently balance work over machine resources; minimize wasted resources

Slurm: Workload Managers at Discovery

- Slurm is an open-source cluster management and job scheduling system for Linux clusters.
- Slurm is short for Simple Linux Utility Resource Management.
- SchedMD® is the core company behind the Slurm workload manager software, a free open-source workload manager designed specifically to satisfy the demanding needs of high-performance computing.
- Slurm is in widespread use at government laboratories, universities and companies worldwide and performs workload management for many systems in the TOP500.

https://www.schedmd.com/

Jobs

- To a user, a job can be simply described as a request for compute resources needed to perform computational work.
- Jobs typically specify what resources are needed, such as type of machine, number of machines, job duration, amount of memory required, account to charge, etc.
- Jobs are submitted to the Workload Manager by means of a job script.

SLURM Commands: System Information

- sinfo – Report system status (nodes, queues, etc.)
  - `$ sinfo -Nle -p debug`
- smap – Report system, job or step status with topology
  - `$ smap`
- scontrol – View status of system, job, partition or reservation
  - `$ scontrol show res=csye7105`

Job Submission

- $ srun – obtain a job allocation (as needed) and execute an application
  -- interactive mode

- $ sbatch – submit a batch script for later execution
  -- batch mode

Jobs Management

- squeue – View information about jobs
  - `$ squeue -u $USER`
  - `$ squeue -j job_id`
  - `$ squeue -t state`
  - `$ squeue -u $USER -p partition -t PENDING`
- scancle – Cancel your jobs
  - `$ scancel -u username`
  - `$ scancel <job_id>`
  - `$ scancel --state=PENDING --user=bob --partition=debug`

- seff <job_id> - Reports the computational efficiency of your calculations

Jobs Management

- Job State Code
  - The typical states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED.
  - **PD R S CG CD ……**

Running jobs

- Single node;
- Multiple nodes.
- Interactive mode
- Batch mode
- Move to a compute node by using `srun` or `sbatch`
- You should **NEVER** launch any jobs from the login nodes login-00 or login-01. Any job launched from the login node will be terminated.

Running jobs – Interactive

- Allocate resource, log onto compute node, run the job and exit
- To move to a compute node, at the command prompt type:

```
$ srun -p debug --pty /bin/bash => for debug
$ srun -p reservation --reservation=csye7105
--pty /bin/bash  => for project
```

- Easy /small / debug
- Interactive

Running jobs – Batch

- Submit to the cluster for later execution

```
$ sbatch configuration file => for project
```

- Configuration file
  - Parameters, what resource do you want
  - Commands, instructions to be executed

- A little more effort
- For longer and bigger jobs

Submitting a job

- See the RC document.
- The general format for submitting a job to the scheduler is as follows:

```
$ sbatch example.sbatch
```

- Example Job Scripts

Sample batch script

```
#!/bin/bash

## Both in single-letter and whole-word formats, e.g. -N 1 and --nodes=1
## Normal configurations
#SBATCH --job-name=test
#SBATCH --output=test.out
#SBATCH --error=test.err
#SBATCH --time=00:15:00     # not necessary

## Parallel configurations
#SBATCH -p express
#SBATCH -N 1
##SBATCH -n 8
```

https://rc-docs.northeastern.edu/en/latest/using-discovery/sbatch.html

Constraint options (not necessary)

#SBATCH --mem=100Gb

#SBATCH --nodelist=c1[234-238]

#SBATCH --exclude=c1234

#SBATCH --constraint="E5-2680v4@2.40GHz" => broadwell

Load modules

Change directory

Start the job

Updated node feature names (February 2021) for the flag "-- constraint":
https://rc-docs.northeastern.edu/en/latest/hardware/hardware_overview.html

Parallel Examples

How to compile and run an OpenMP Program

Compiler	Compiler Options	Default behavior for # of threads (OMP_NUM_THREADS not set)
GNU (gcc, g++, gfortran)	-fopenmp	as many threads as available cores
Intel (icc ifort)	-openmp	as many threads as available cores
Portland Group (pgcc,pgCC,pgf77,pgf90)	-mp	one thread

GNU Compiler Example:

$ **gcc -o** omp_helloc **-fopenmp** omp_hello.c
$ export OMP_NUM_THREADS=4
$ ./omp_helloc

Intel Compiler Example:

$ **icc -o** omp_helloc **-openmp** omp_hello.c
$ export OMP_NUM_THREADS=4
$ ./omp_helloc

How to Compile and Run a MPI Program

The table below lists OpenMPI compiler wrapper scripts for Linux clusters.

Language	Script Name	Underlying Compiler
C	**mpicc**	C compiler for loaded compiler package
C++	**mpiCC** **mpic++** **mpicxx**	C++ compiler for loaded compiler package
Fortran	**mpif77**	Fortran77 compiler for loaded compiler package. Points to mpifort.
	mpif90	Fortran90 compiler for loaded compiler package. Points to mpifort.
	mpifort	Fortran 77/90 compiler for loaded compiler package.

$ mpicc -o mpi_hello mpi_hello.c
$ mpiexec -np 8 –oversubscribe mpi_hello

Compile and run the programs you have parallelized with OpenMP

```
#!/bn/bash

#SBATCH --job-name=omp_test_01
#SBATCH --output=omp_test_01.out
#SBATCH --error=omp_test_01.err
##SBATCH --time=00:15:00

## Parallel configurations
#SBATCH -p debug
#SBATCH -N 1
#SBATCH --exclusive
```

```
## Cntd'

work = $HOME/csye7105/hw1
cd $work

export OMP_ NUM_THREADS=8
./omp_hello
```

Compile and run the programs parallelized with MPI

```
#!/bn/bash

#SBATCH --job-name=mpi_test_01
#SBATCH --output=mpi_test_01.out
#SBATCH --error=mpi_test_01.err
##SBATCH --time=00:15:00

## Parallel configurations
#SBATCH -p debug
#SBATCH -N 1
#SBATCH --exclusive
```

```
## Cntd'

module load openmpi/3.1.2

work = $HOME/csye7105/hw1
cd $work

mpirun -np 8  -oversubscribe ./mpi_hello
```

Run the Python programs in parallel
—— Embarrassingly Parallel for training

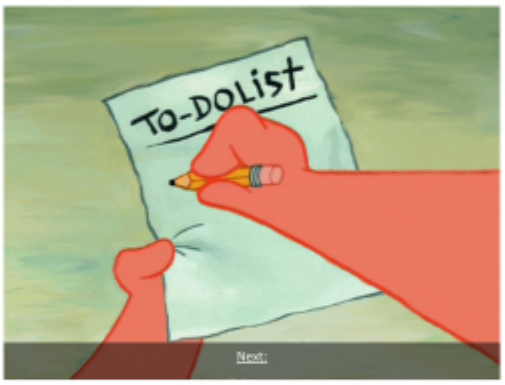
```
#!/bn/bash

#SBATCH --job-name=mpi_test_01
#SBATCH --output=mpi_test_01.out
#SBATCH --error=mpi_test_01.err
##SBATCH --time=00:15:00

## Parallel configurations
#SBATCH -p reservation
#SBTACH --reservation=csye7105
#SBATCH --mem=10GB
#SBATCH -N 1
#SBATCH -n 16
```

```
## Cntd'

module load anaconda3/3.7
source activate py37

work = $HOME/csye7105/project
cd $work

./cv01.py        # coded with embarrassingly parallel
```

Materials

- https://rc-docs.northeastern.edu/en/latest/get_started/get_access.html

- Stay safe!
- See you next class!



Next Lecture will Continue:

Parallel Processing in Python