

# OSCER Supercomputer

Andrew H. Fagg  
Symbiotic Computing Laboratory  
School of Computer Science

# Logging In

Primary entry point to the “login nodes”: schooner.oscer.ou.edu

- Will automatically connect you to one of schooner1, schooner2 or schooner3 (automatic load balancing)
- You can also directly ssh to any of these
- Large scale data transfer (e.g., with scp or sftp): use dtn2.oscer.ou.edu

```
ssh schooner.oscer.ou.edu -l OSCER_USER_NAME
```

# Login Nodes vs Compute Nodes

## Login:

- Primary use: configuring, launching and monitoring experiments
- Can be used for limited testing and debugging
- But: NEVER do real experiments on these login nodes

## Compute:

- This is where the real experiments are done
- Never directly login to these nodes to launch experiments
  - we will use SLURM for this

# Key Directories

- Your home directory: /home/USER\_NAME/
  - Persistent storage
  - Limited size (~ 20GB)
- My home directory: /home/fagg
  - Some materials, including datasets will be left here
- Scratch space: /scratch/USER\_NAME/
  - May have to request that a directory be created for you ([support@oscer.ou.edu](mailto:support@oscer.ou.edu))
  - Large-scale temporary storage
  - Old files are automatically deleted (you will receive a warning email first)

# Managing Your Own Files

- Suggest: develop and test on your own computer
- When ready, move to the supercomputer for the bigger experiments
  - CPUs on the supercomputer are generally less capable than your modern laptop (but the supercomputer has *\*a lot\** of CPUs)
- Strong suggestion: use a source control management system to share code and config files between your own computer and the supercomputer (e.g., svn or git)

# Managing Your Own Files

Supercomputer will produce many results files. Bring these back to your own computer for analysis and visualization

- Don't check these into your repository. Instead, transfer with scp or rsync
  - Big transfers should be done through dtn2.oscer.ou.edu
- Results files can be really big:
  - Don't keep them on the supercomputer for very long
  - You might want to store in /scratch

# SLURM

SLURM = the job control system for the supercomputer

- Documentation:  
[https://www.ou.edu/oscer/support/running\\_jobs\\_schooner](https://www.ou.edu/oscer/support/running_jobs_schooner)
- Process of executing an experiment:
  - Submit the experiment to the appropriate job partition (sbatch command)
  - When there is a compute node available and you are at the front of the queue, the compute node will pick up the job and begin execution
  - During execution: data will be written to log files
  - Once your job is complete, its status will be updated and its record will be removed from the job control system

# Supercomputer Partitions

- Can examine the partitions and their states using commands such as:

```
sinfo
```

```
sinfo | grep normal
```

- Useful partitions:

- debug\_5min: quick pick up; cannot run for more than 5 minutes
- debug: relatively quick pick up; 30 min limit
- normal: where many experiments will be; 48 hour limit
- debug\_gpu: test gpu code; 30 min limit
- gpu: processor has a GPU

# Batch Files

- Resource request:
  - Which partition
  - How many CPUs/cores
  - How much memory
  - Maximum running time
  - How many experiments to run at once
- What to execute:
  - Configure python environment
  - Python program to run
- See example batch file

# Batch File: Resource Request

```
#!/bin/bash
#
#SBATCH --partition=debug_5min
#SBATCH --ntasks=1
# memory in MB
#SBATCH --mem=1024
# The %j is the job ID
#SBATCH --output=results/xor_%j_stdout.txt
#SBATCH --error=results/xor_%j_stderr.txt
#SBATCH --time=00:02:00
#SBATCH --job-name=xor_test
#SBATCH --mail-user=INSERT_YOUR_EMAIL_ADDRESS_HERE
#SBATCH --mail-type=ALL
#SBATCH --chdir=/home/fagg/aml/demos/basics
```

# Batch File: What to Execute

```
# Set up python environment (can copy directly)
. /home/fagg/tf_setup.sh
conda activate tf

# Change this line to start an instance of your experiment
python xor_base.py --exp 0 --epochs 10
```

# Configuring Your Batch File

- “cd” to your experiment directory
- Create the file (assuming batch.sh in this example)
- Set the parameters for your particular context
- Set the batch file to be “executable”. At the command line:

```
chmod a+x batch.sh
```

- Make sure that all of the necessary directories have already been created (our example uses a local ‘results’ directory)

# Testing Your Batch File

You can test the execution part of your batch file on the login node by:

- “cd” to the directory where your experiment is located
- Execute the following in your shell:  
./batch.sh
- Note that this will execute on the login node itself. You should only perform very quick experiments here for testing purposes

# Queueing Your Experiment

- Assuming that your current working directory is your experiment directory (where the batch file is located)
- Type:

```
sbatch batch.sh
```

- If there are no errors, then you are good to go
- Check status of your jobs:

```
squeue -u USER_NAME
```

# After the Job is Done

Two log files can be very helpful:

- stderr:
  - Any errors or warnings that were generated by your program
  - Can have a lot of useful detail (but many things can be ignored)
- stdout:
  - Anything printed by your program

Note: you will generally be producing your own files (e.g., logs of learning curves; full copies of the learned models, etc.)

# Executing a Set of Jobs at Once

Often, we want to execute the same program with small variations

- Multiple, independent learning runs of the same model architecture
- Variations in architecture
- Variations in hyper-parameters

SLURM allows us to queue up a set of experiments (can be numbered 0...999)

# Executing a Set of Jobs at Once

SLURM allows us to queue up a set of experiments (can be numbered 0...999)

- Python code then translates this integer into a specific experiment
- Today: we will use this to just execute repetitions of the same experiment
- Can also use this to also select other hyper-parameters

# Batch File for Multiple Jobs

Add to the batch file:

```
#SBATCH --array=0-3
```

Then, for each job, the shell variable `$SLURM_ARRAY_TASK_ID` will encode the integer (in this case, between 0 and 3, inclusive). You can use this in the execution part of your batch file:

```
python xor_base.py --epochs 10 --exp $SLURM_ARRAY_TASK_ID
```

# Supercomputer Etiquette

This is a resource that is shared by *\*many\** people, so “play nice”

- Limit use of login nodes to configuring/starting jobs and doing very basic tests
- Try your best to properly estimate your resource needs
  - Minimizing the requests will mean that your jobs get picked up quicker
  - But if you underestimate your execution time, you will have to start over
- Don’t execute servers (e.g., jupyter) on the login nodes
- Plan your big experiments carefully

# Notes

- Don't change the underlying code if you have jobs queued up – when they start, they will pick up the most recent version of the code (whether it works or not)
- The log files can be really helpful for debugging ... you just have to sort through the noise
- Test before you do big jobs (but keep the tests very small)
  - Your local machine
  - Login node