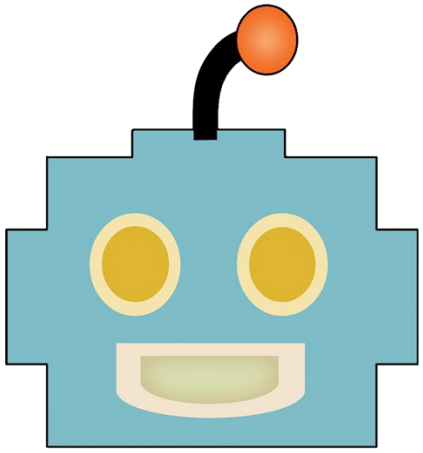
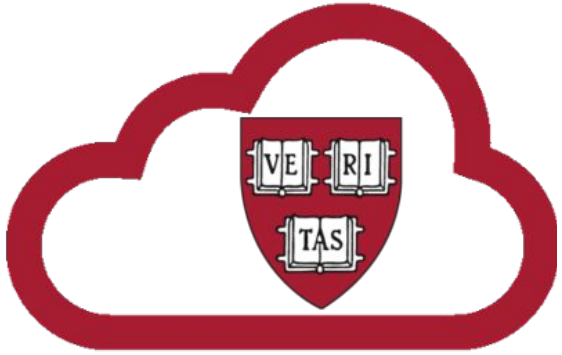
High Performance computing On Odyssey @ FASRC





Francesco Pontiggia - Research Consultant

Feb 2019
AC290R

Research Computing Resources

- 82k cores HPC
- Over 2 Petaflops GPUs (NVidia)
- 40.0PB of storage (Lustre, XFS/NFS, Isilon, gluster, Ceph, GPFS)
- 400+ virtual machines (KVM/OpenNebula)
- 2MW of research computing equipment in 3 data centers
- 23 FTE in 5 groups
  - RC Support Services
  - Data Science & Research Facilitation
  - Research Software Engineering
  - Cloud/Software as Infrastructure
  - Systems Engineering & Data Center Operations
  - Supporting 600 research groups and 4200 active users

Cluster Terminology

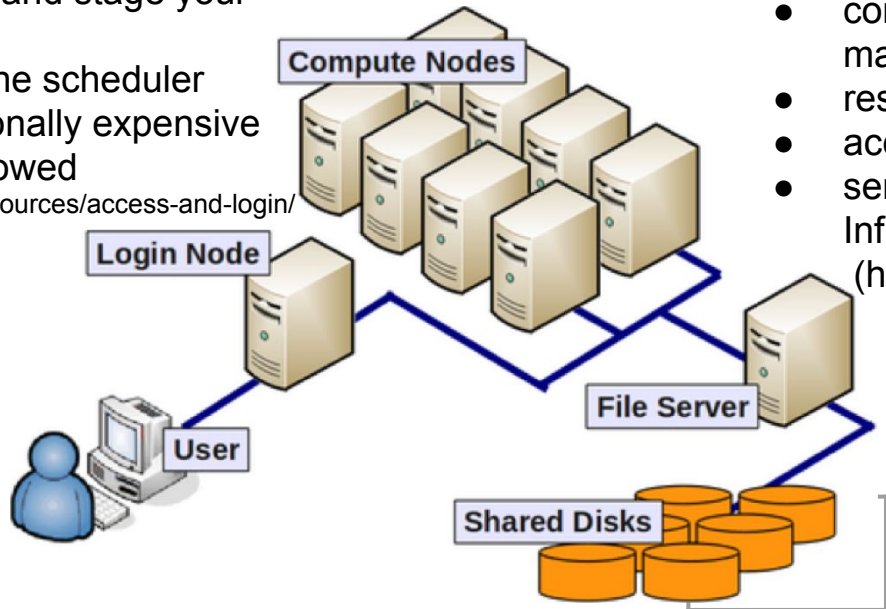
- Supercomputer/High Performance Computing (HPC) cluster: A collection of similar computers connected by a high speed interconnect that can act in concert with each other.
- Server, Node, Blade, Box, Machine : An individual motherboard with CPU, memory, network, and local hard drive.
- CPU (Socket): Central Processing Unit, a single silicon die that can contain multiple computational cores
- Core: Basic unit of compute that runs a single instruction of code
- GPGPU/GPU: General Purpose Graphics Processing Unit, a GPU designed for supercomputing.
- InfiniBand (IB): A near zero latency high bandwidth interconnect used in Supercomputing
- Serial: Doing tasks/instructions in sequence on a single core
- Parallel: Doing tasks/instructions on multiple cores simultaneously
- I/O: Input/Output, a general term for reading and writing files to/from storage whether local or remote.

Cluster Basics

Login Nodes :

- prepare input and stage your calculations
- interact with the scheduler
- no computationally expensive processes allowed

www.rc.fas.harvard.edu/resources/access-and-login/



Compute nodes:

- computational resource monitored and managed by the SLURM scheduler.
- resources are organized in partitions
- access only via scheduler
- servers are interconnected by Infiniband fabric
  (high throughput , low latency)

Storage:

	Home Folders	**Local Scratch**	**Global Scratch**
mount	$HOME	/scratch	/n/regal/ac290r/
size limit	100G	~ 270G/node	quota 50T/group
backup / retention	Hourly snapshots	only available during job	**90days retention** no backup
performance	Not suitable for intense I/O	Suited for small file I/O intensive jobs	Suited for large file I/O intensive jobs

Documentation: www.rc.fas.harvard.edu

Here you will find all our user documentation.
Of particular interest for you :

- Access and Login :
  https://www.rc.fas.harvard.edu/resources/access-and-login/
- Running Jobs :
  https://www.rc.fas.harvard.edu/resources/running-jobs/
- Software modules available :
  https://portal.rc.fas.harvard.edu/apps/modules
  or use the command module-query
- singularity containers
  https://www.rc.fas.harvard.edu/resources/documentation/software/singularity-on-odyssey/
- gpu computing
  https://www.rc.fas.harvard.edu/resources/documentation/gpgpu-computing-on-odyssey/

How to get help :
https://www.rc.fas.harvard.edu/resources/support/

Login & Access

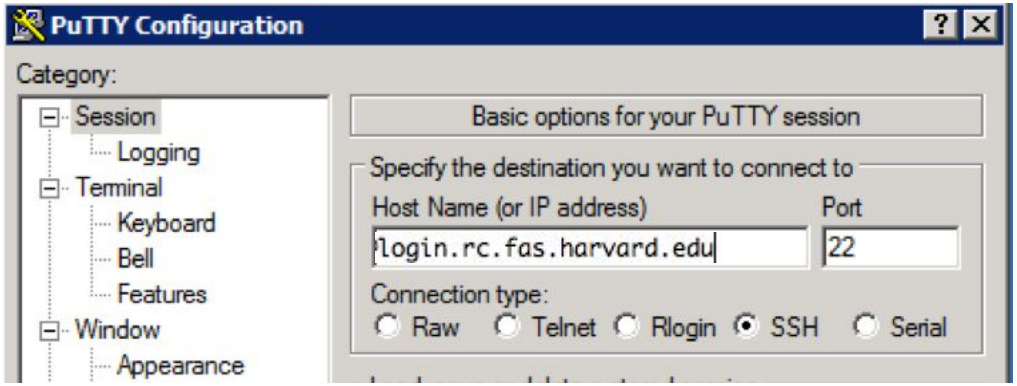
- Terminal application is needed to connect via secure shell (SSH)
  - Mac: Terminal.app on Mac/Linux
  - Linux: Xterm

```
> ssh username@login.rc.fas.harvard.edu
```

```
Odyssey2
Login issues? See https://rc.fas.harvard.edu/resources/support/

Password:
Verification code:
```

- Windows: Putty



Verification Code?

- OpenAuth is 2-factor authentication separate from HarvardKey and updates the token every 30 seconds
- Download OpenAuth from: https://software.rc.fas.harvard.edu/oa/
- NOTE: OpenAuth token requires that your computer time be correct. If you have problems logging in this is one of the first things you should check.

Access Issues?

- Accounts are locked for 15 minutes after 5 failed login attempts in a row.
- Password Reset: https://portal.rc.fas.harvard.edu/pwreset/

Transfer Files

- Secure File Transfer: SFTP Client

  - GUI client FileZilla for all platforms
  - Configure according to http://fasrc.us/configfilezilla to avoid 2FA problems

- command-line from local terminal application
  - scp: secure *copy*

```
scp file1 username@login.rc.fas.harvard.edu:directory2/
```

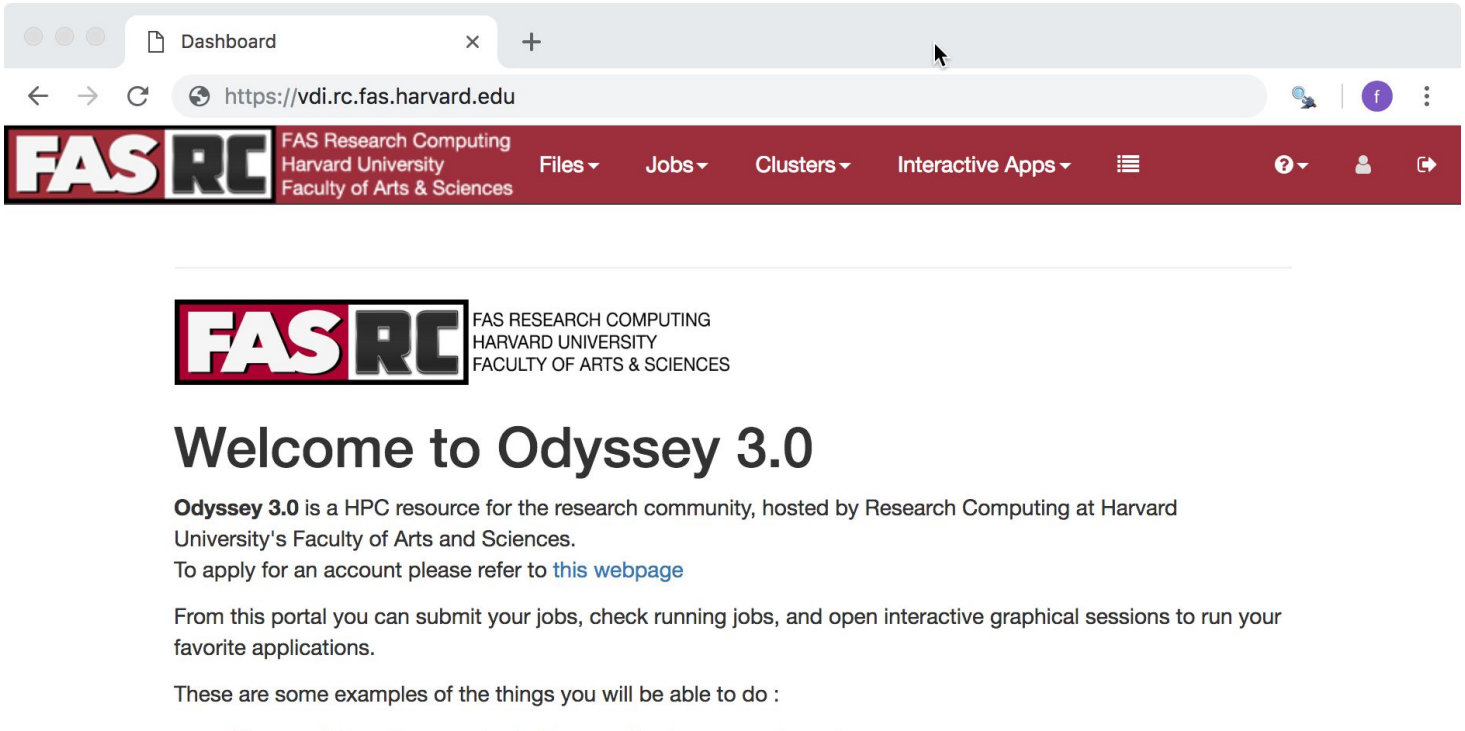
```
scp -r directory1 username@login.rc.fas.harvard.edu:directory2/
```

  - rsync: remote *sync*

```
rsync -av --progress directory1/ username@login.rc.fas.harvard.edu:directory2/
```

Interactive Computing website

You can login and schedule jobs and interactive sessions from the portal
https://vdi.rc.fas.harvard.edu



Documentation available at
https://www.rc.fas.harvard.edu/resources/documentation/virtual-desktop/

Scientific Software

- Software Packages installed by RC staff can be accessed loading the corresponding software modules.
- Basic module subcommands:
  - load : loads a module
    >$ module load gcc/7.1.0-fasrc01
  - unload : unloads a module
    >$ module unload gcc/7.1.0-fasrc01
  - list : lists the currently loaded modules
    >$ module list
  - purge : unload all modules
    >$ module purge

Available modules can be searched at https://portal.rc.fas.harvard.edu/apps/modules or using the command “module-query”.

- Users can build and install any software in your user folder or lab folder.

- Singularity can be used to run docker containers or singularity images.

SLURM

- Simple Linux Utility for Resource Management
  User tasks (jobs) on the cluster are controlled by slurm and isolated in cgroups so that users cannot interfere with other jobs or exceed their resource request (cores, memory, time)
- Basic SLURM commands:
  - sbatch: submit a batch job script
    >$ sbatch [options for resource request] myscript
  - srun: submit an interactive test job
    >$ srun --pty [options for resource request] /bin/bash
  - squeue: contact slurmctld for currently running jobs
    >$ squeue
  - sacct: contact slurmdb for accounting stats after job ends
    >$ sacct
  - scancel: cancel a job(s)
    >$ scancel somejobnumber

https://www.rc.fas.harvard.edu/resources/documentation/convenient-slurm-commands/
https://www.rc.fas.harvard.edu/resources/running-jobs/

Slurm Partitions

https://www.rc.fas.harvard.edu/resources/running-jobs/#Slurm_partitions

Partition	Nodes	Cores per Node	CPU Core Types	Mem per Node (GB)	Time Limit	Max Jobs	Max Cores	MPI Suitable?	GPU Capable?
shared	448	32	Intel "Broadwell"	128	7 days	none	none	Yes	No
general	133	64	AMD "Abu Dhabi"	256	7 days	none	none	Yes	No
serial_requeue	varies	varies	AMD/Intel	varies	7 days	none	none	No	Yes
test*	16	32	Intel "Broadwell"	128	8 Hours	5	64	Yes	No
gpu_requeue	varies	varies	Intel (mixed)	varies	7 days	none	none	No	Yes
gpu	1	24	Intel "Ivybridge"	30	7 days	none	none	No	Yes (8 K20Xm)
fas_gpu	88	varies	varies	varies	7 days	none	none	No	Yes (128 K80, 16 K20Xm, 16 K20m)
bigmem	6	64	AMD "Abu Dhabi"	512	none	none	none	No	No
unrestricted	8	64	AMD "Abu Dhabi"	256	none	none	none	Yes	No
PI/Lab nodes	varies	varies	varies	varies	none	none	none	varies	varies
holyseasgpu	**13**	**48**	**Intel**	**96**	**none**	**none**	**none**	**yes**	**yes (2x K40m / node)**
seas_dgx1	4	40	Intel	512	3 days*	none	none	yes	yes (8x V100 / node)

SLURM Scheduler

Jobs can be run in batch mode

>$ sbatch [options with requested resources] myscript

or interactive mode

>$ srun --pty [options with requested resources] /bin/bash

Typical options include :

-t : time in minutes or D-HH:MM:SS I am requesting for the job

-n : number of tasks I intend to run

-c : number of cores per task (number of cores allocated= tasks * cores/task)

-N : number of nodes on which distribute the tasks

--mem : memory allocated per node (by def unit is M)

-p : partition

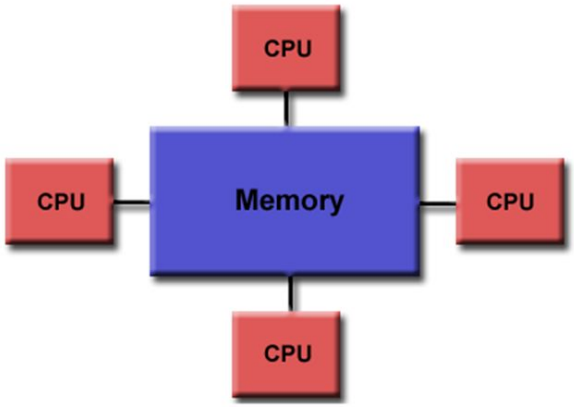
--gres : additional resources, for example --gres=gpu:k to allocate k gpus

--reservation=ac290r to use the reserved nodes for the class in “shared”.

--account=ac290r to use the slurm account for the class instead of your lab one

SLURM Scheduler

Parallel shared memory (single node) jobs



Examples:

- OpenMP (Fortran, C/C++)
- MATLAB Parallel ComputingToolbox (PCT)
- Python (e.g., threading, multiprocessing)
- R (e.g., multicore)

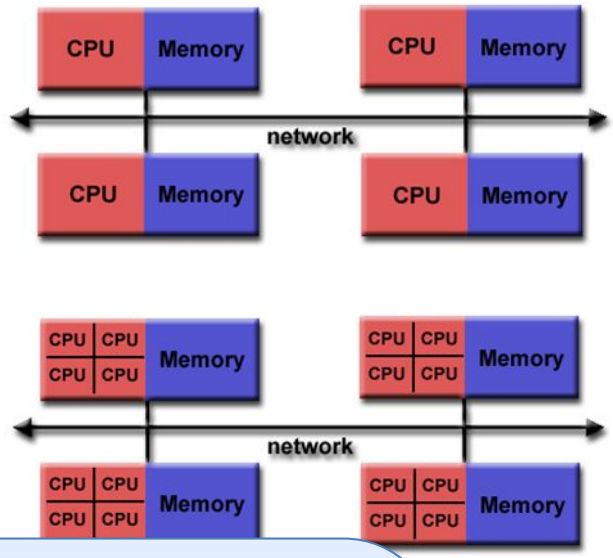
```
#!/bin/bash
#SBATCH -p shared                # Partition
#SBATCH -t 0-02:00               # Runtime in D:HH:MM
#SBATCH -c 4                     # Number of cores/tasks
#SBATCH -N 1                   # Number of nodes
#SBATCH --mem=4000           # total memory per node

module purge
module load  some_program
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun -c $SLURM_CPUS_PER_TASK some_program  PROGRAM_OPTIONS
```

SLURM Scheduler

Parallel distributed memory (multi-node) jobs



Examples:

- MPI (openmpi, impi, mvapich) with Fortran or C/C++ code
- MATLAB Distributed Computing Server (DCS)
- Python (e.g., mpi4py)
- R (e.g., Rmpi, snow)

```
#!/bin/bash
#SBATCH -p shared                    # Partition
#SBATCH -t 0-02:00                   # Runtime in D:HH:MM
#SBATCH -n 4                         # Number of cores/tasks
#SBATCH --mem-per-cpu=4000           # Memory per task

module purge
module load  some_mpi some_program

srun -n $SLURM_NTASKS --mpi=pmi2 some_program  PROGRAM_OPTIONS
```

Basic Workflow

Remember that the correct place to run your applications is the global scratch

SETUP STORAGE

```
WORK_DIR=/n/regal/ac290r/users/$USER/RUN1   # or WORK_DIR=/scratch/$USER/...
mkdir -pv $WORK_DIR
```

COPY INPUT FILES

```
cp $input $WORK_DIR
cd $WORK_DIR
```

load software and execute code

```
module purge
module load somemodule
./myprogram -i $input -o myresults
```

(or srun -c / srun -n as described earlier for openmp / mpi programs)

COPY OUTPUT BACK

```
cp -a myresults  some_safe_folder
```

GPGPU Computing @ FASRC

Submit GPGPU Job:

>$ sbatch -p holyseasgpu -n 1 --gres=gpu:2 gpu-job.sh

CUDA:

>$ module-query cuda

cuda/9.2.88-fasrc01
cuda/9.1.85-fasrc01
cuda/9.0-fasrc02
cuda/8.0.61-fasrc01
cuda/7.5.18-fasrc01

>$ module load cuda/9.0-fasrc02
>$ which nvcc
/n/helmod/apps/centos7/Core/cuda/9.0-fasrc02/bin/nvcc

https://rc.fas.harvard.edu/resources/documentation/gpgpu-computing-on-odyssey/

(Singularity) Containers @ FASRC

Singularity (https://www.sylabs.io/singularity/)

- Improved Security (no root-owned daemon)
- Allows HPC users to easily take advantage of containers
- Reproducible and mobile software stacks
- Easy conversion of docker containers to singularity
- You're "you" inside of the container
- $HOME folder and storage mounts visible inside the container

Refer to official documentation and on notes on

https://www.rc.fas.harvard.edu/resources/documentation/software/singularity-on-odyssey/

Creating Singularity Images

Option 1:

Using Docker Images: Docker -> Singularity Workflow

- Install docker (locally or in a VM)
- Develop docker image locally and push to docker registry (e.g. docker hub)
- Pull and convert image using singularity on Odyssey cluster
- Skip the first two steps if there's already a community container (e.g. tensorflow)

- You can export images from docker and import in singularity without using a registry, but you would need admin privileges to do that (so cannot be done on Odyssey)

Creating Singularity Images

Option 2 (only recommended to advanced users):

Native Singularity Images

- Custom image spec/format
- Single file vs docker “layers” (benefits/drawbacks)
- Have to be root to build containers (same as docker)

Building/Deploying Native Images

- Launch VM outside of Odyssey (e.g. VirtualBox)
- Install singularity
- Write Singularity image build script
- Build image
- Copy image file to Odyssey
- Optional: push image file to https://singularityhub.com then “pull” from Odyssey

Docker --> Singularity

$ singularity pull docker://tensorflow/tensorflow:1.6.0-gpu-py3

…

Done. Container is at: tensorflow-1.6.0-gpu-py3.img

$ singularity exec --nv tensorflow-1.6.0-gpu-py3.img python3 multigpu_cnn.py

…

Step 190: Minibatch Loss= 0.0180, Training Accuracy= 0.998, 27081 Examples/sec

Step 200: Minibatch Loss= 0.0129, Training Accuracy= 0.996, 27029 Examples/sec

Optimization Finished!

Testing Accuracy: 0.99450934

Simple Job Examples

You can find some simple job examples in the folder

/n/regal/ac290r/shared/

Big Thanks to ^^^

Francesco Pontiggia - Research Consultant

Feb 2019
AC290R