Stanford Research Computing

ONBOARDING TO THE SHERLOCK
High Performance Computing Cluster
October 5 2022

https://srcc.stanford.edu/

Agenda

- Intro to SRCC and HPC
- Sherlock's layout, storage, partitions, limits
- Submitting jobs
- Modules
- Installing software
- Typical HPC workflow
- Estimating your job's resources
- Questions?

Download slides here- https://srcc.stanford.edu/events/series/sherlock-boarding-session
Video of this presentation- https://youtu.be/iqq7GGqMRg8

SRCC- Our Group

Stanford Research Computing Center

We manage and support:

- Sherlock
- SCG Genomics Cluster
- Farmshare
- Carina Secure Computing Environment
- ICME
- Oak HPC Storage
- PHS

Enabling Research and Discovery
SRCC
Stanford University srcc.stanford.edu

What is High Performance Computing (HPC)

"HPC generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get from a typical desktop computer or workstation in order to solve large problems."
– Inside HPC

- When Will I Need It?

  Almost every field of *research where simulations, large computation or data is needed*: Astrophysics, Social Sciences, Biology, Chemistry, Economics. Most common software run on Sherlock: R, Matlab, and Python

  For computing needs above and beyond what your laptop/desktop can handle, in terms of *CPUs, RAM, time, storage* and *I/O*

Personal Computers vs. High Performance Computers

Mac Book Pro Laptop

- 2 cores (1 CPU)
- 16 GB RAM
- 512GB Solid State Disk

Typical Sherlock Node

- AMD EPYC 32-Core Processors, others have 24 CPUs in two sockets, Intel 2.4GHz Xeon Skylake CPU, up 256 CPUs can be run at once on Sherlock, 8,192 CPUs for owners.
- 192GB RAM
- Some nodes have 128 CPUs, 1TB RAM
- 100TB scratch storage, 1TB group home, 200GB local Solid State Disk
- Infiniband connection 100GB/s between nodes and storage (Scratch, Oak and Home)
- GPU nodes (NVIDIA Kepler K80, K40, Volta V100)
- Big memory nodes (512GB, 1.5 and 4TB RAM)

A key difference – Laptop vs. HPC Cluster

Running on a cluster with SLURM is different than running your code on a laptop. If you run an R script on your laptop, R will take as much of whatever CPU/time/RAM it needs from the operating system. You may notice that all other programs will grind to a halt.

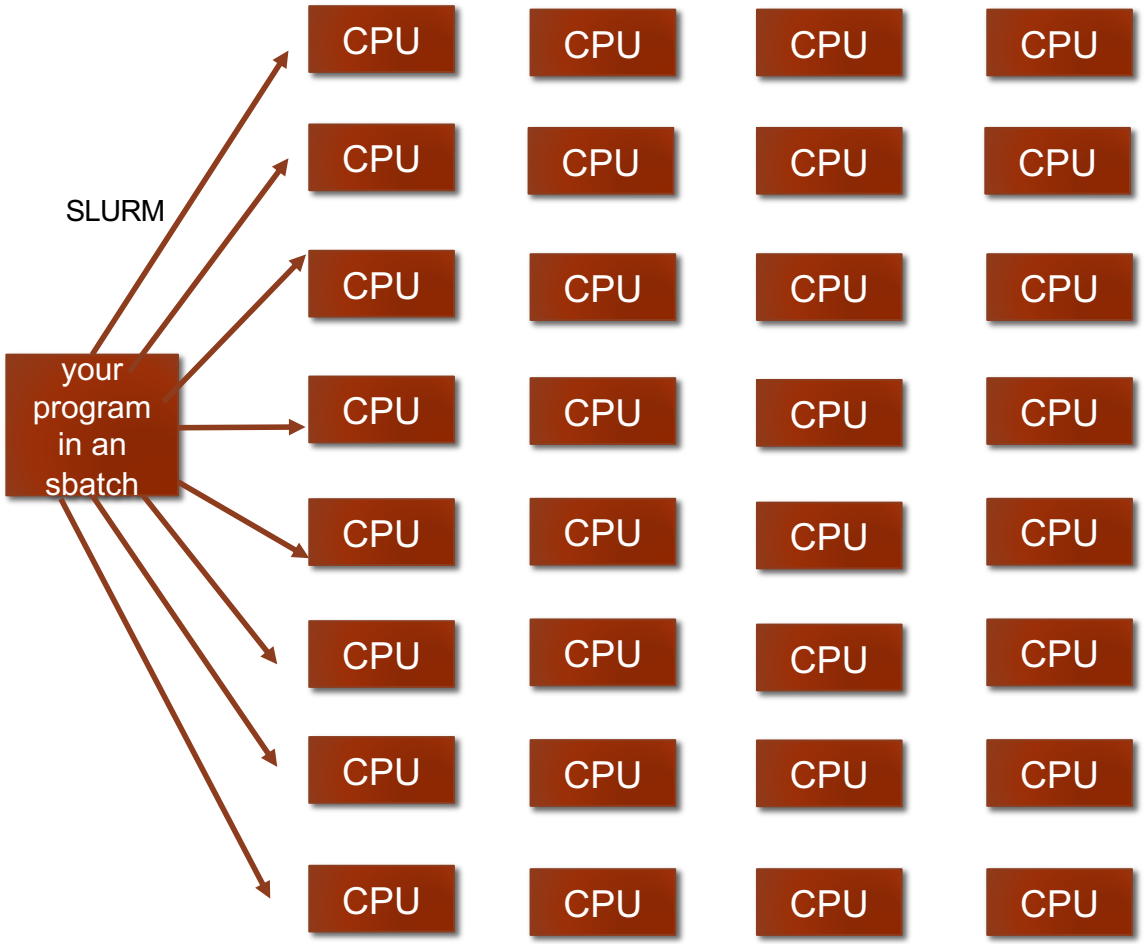
So those 2 cores and 16GB of RAM on your laptop are pretty much allocated.

However: on a cluster you have access to thousands of CPUs, a large parallel filesystem and lots of RAM. *But we need to use a job scheduler (SLURM) to allocate, limit and control those resources*. We also have a queue to deal with user contention for those resources.

You need to explicitly ask for those resources from the scheduler in an **sbatch** file with #SBATCH directives. And sometimes…you must wait.

Just remember your laptop does not have 512 CPUs, a gigabit network connection and a 100TB hard drive.

Parallel Processing



On a cluster multiple tasks (instances of your code) can be submitted via a job scheduler to many CPUs and servers at once

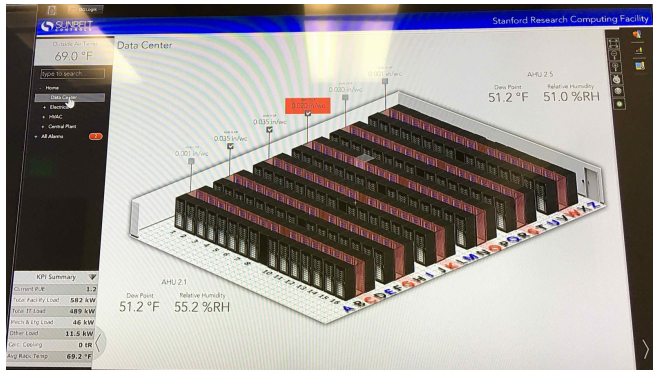
Pass multiple arguments to your code at once

For example, if you need to process 500 files with your script, these can be processed at once rather than serially

Multiple instances of your code can run simultaneously across the cluster

No need to wait for 2 or 3 cores and limited RAM on a laptop or desktop to be available

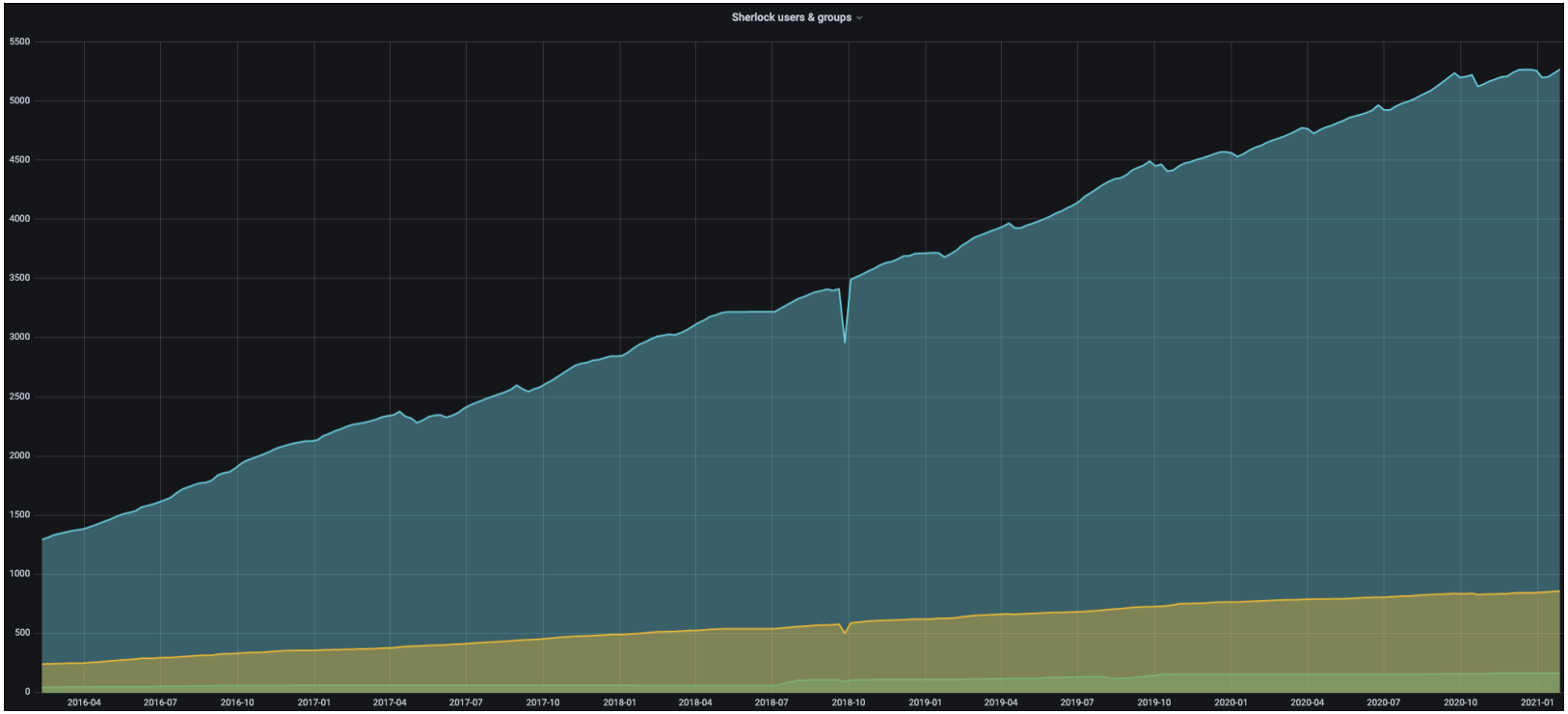
Where is Sherlock? - SRCF Data Center @SLAC



An example day-
IT load 489 kW
Facility load 582 kW

Racks
Servers(nodes)
UPS
Generators
Networking/Fiber
Cooling
Electrical

Sherlock's user growth since 2016



5,726 users from 916 research groups, **176 owner partitions**

6 PB scratch, 28 PB long term Oak storage

Stanford Data Risk Classifications

https://uit.stanford.edu/guide/riskclassifications

Low Risk

Data and systems are classified as Low Risk if they are not considered to be Moderate or High Risk, and:

1. The data is intended for public disclosure, or
2. The loss of confidentiality, integrity, or availability of the data or system would have no adverse impact on our mission, safety, finances, or reputation.

Moderate Risk

Data and systems are classified as Moderate Risk if they are not considered to be High Risk, and:

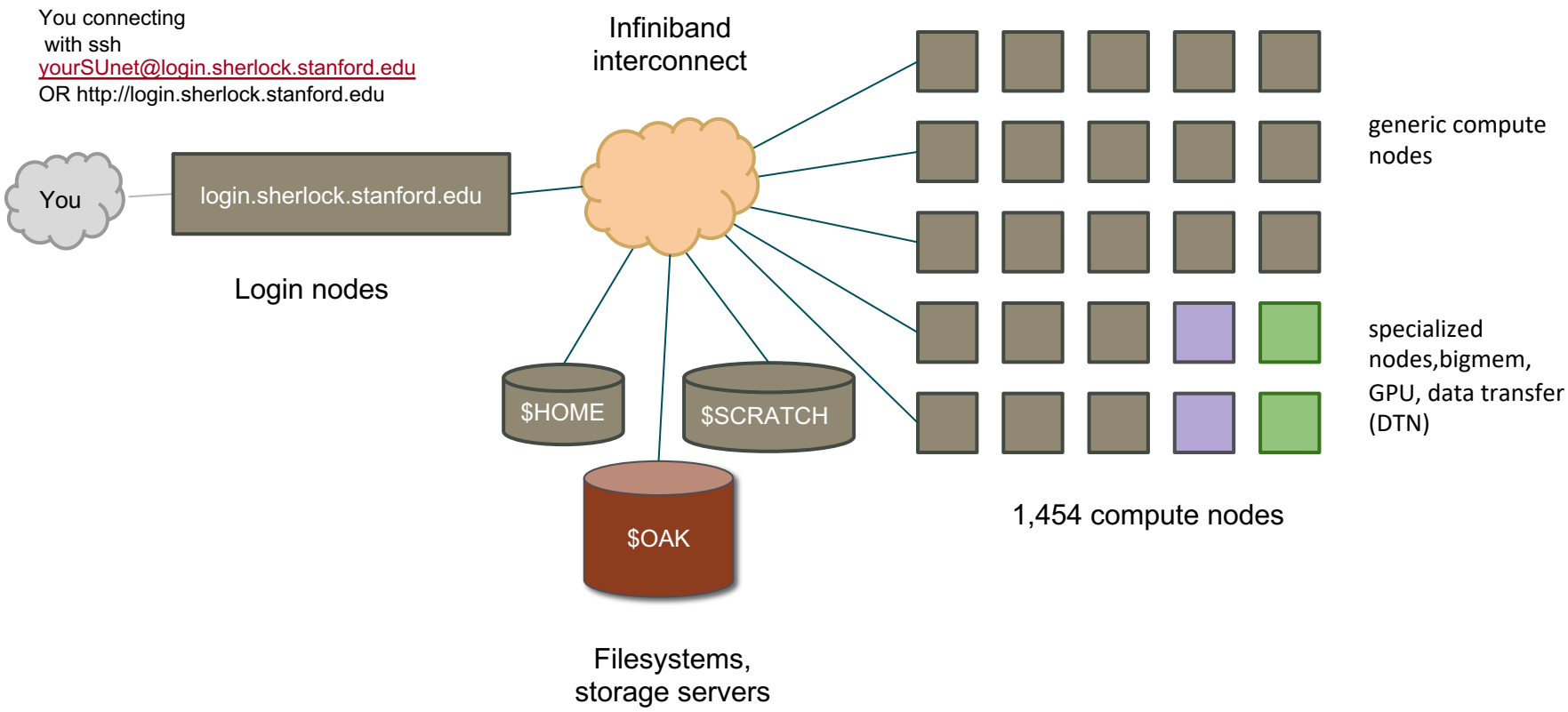
1. The data is not generally available to the public, or
2. The loss of confidentiality, integrity, or availability of the data or system could have a mildly adverse impact on our mission, safety, finances, or reputation.

High Risk

Data and systems are classified as High Risk if:

1. Protection of the data is required by law/regulation,
2. Stanford is required to self-report to the government and/or provide notice to the individual if the data is inappropriately accessed, or
3. The loss of confidentiality, integrity, or availability of the data or system could have a significant adverse impact on our mission, safety, finances, or reputation.

Sherlock System Overview



Connecting to Sherlock

From your local system (laptop/desktop)

$ ssh <sunetid>@login.sherlock.stanford.edu

Mac- use the Terminal app

Windows
Cygwin
"Windows Subsystem for Linux" (WSL)

Linux
ssh

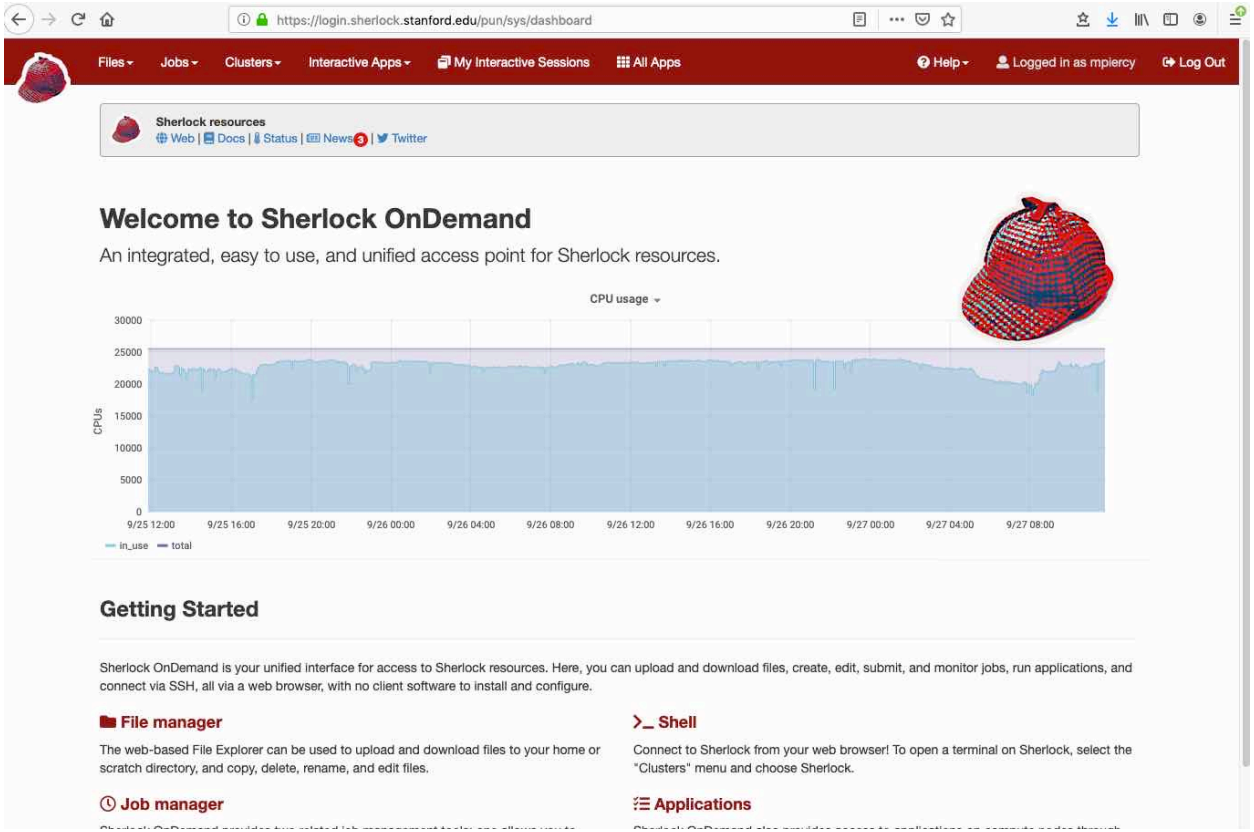
More info: Connecting to Sherlock

Web browser access to Sherlock

Access Sherlock with your browser - https://login.sherlock.stanford.edu/

- open a shell session
- move, copy files from your laptop to Sherlock
- edit files

OnDemand Documentation



Sherlock Partitions

Partition- a logical and physical set of nodes (servers, computers) in a cluster

Partition	Number of Nodes
normal	154
owners	1,203
hns	95
bigmem (512GB-4TB RAM)	3
gpu	26
dev	2
Your PI's or groups nodes	**?**

What partitions can I run on? Run: **sh_part**

or

scontrol show partition | egrep -B1 "AllowGroups=.*$(id -gn $user).*" | awk -F= '/PartitionName/ {print $2}'

What nodes are in the partition, CPUs/RAM/node?

sinfo -Nlp normal

Sherlock hns partition

Open to all users from the School of Humanities and Sciences (Political Science , Statistics, Sociology, Communication, Economics Departments, etc.)

95 Compute nodes (servers)

3,216 CPUs

2 Large memory nodes .5 TB RAM

4 128 CPU nodes with 1TB RAM

To use add

#SBATCH -p hns

to your submission scripts

Check hns layout:

sinfo -Nlp hns

Sherlock **serc** partition for users from School of Sustainability

Open to all users from the School of Earth

132 Compute nodes (servers)

5,456 CPUs

64 GPUs

14 128 CPU nodes with 1TB RAM

To use add

#SBATCH -p serc

to your submission scripts

To see if you can run on serc use the **sh_part** command

Check serc's layout:

sinfo -Nlp serc

The HPC Condo Model

Sherlock faculty (PIs) can buy 1 or more nodes (starting at about $7,500)

- The PI's group members will have exclusive use of these nodes
- Access to idle resources in the owners partition of 1,158 nodes
- An owner's group members will be able to use up to 8,192 CPUs at once in the owners partition
- However, jobs in the owners queue are preemptible, if the owner of the node you are running on wants to use those resources, your job is terminated. So these jobs need to be checkpointed in some way. Or at least you need to be able to logically aggregate the data at that state and restart processing. On Sherlock preempted jobs in owners are automatically re-queued 5 times.

Sherlock node orders

Sherlock Filesystems

Home and Group Home, backed up, snapshotted and replicated, no purge policy

$HOME 15 GB

$GROUP_HOME 1 TB

Scratch- fast i/o Lustre parallel filesystem, your jobs should write/read here (3 month purge policy- if a file is not modified after 3 months it is deleted)

$SCRATCH	100 TB
$GROUP_SCRATCH	100 TB
$OAK	$42.95 per 10TB/month, no purge policy

$L_SCRATCH ~200GB (some nodes will have more)

Local to your job's node, even faster, but gets deleted at the end of your job- move your results/data to your $SRATCH or $GROUP_HOME when job ends, just add the mv/cp at the end of your sbatch script. Most users don't need to use $L_SCRATCH

http://www.sherlock.stanford.edu/docs/user-guide/storage/filesystems/

run **sh_quota** to see what is available to you

Use the rclone command to connect to your cloud storage accounts, great for back-ups, submit back-ups as a recurring job

I/O on Sherlock's Filesystems

- When running jobs always try to use the $SCRATCH or $LOCAL_SCRATCH as much as possible instead of $HOME and $GROUP_HOME
- Do not submit large number of jobs (your R, Python, Matlab, C++ code etc.) which open, read and close the same file in your $HOME/$GROUP_HOME directories several times per second from each of your code's threads.
- This can lead to thousands of processes opening and closing the same file simultaneously.
- This may put a huge strain on the whole filesystem affecting all users on Sherlock. We may need to pause/end these jobs
- Check your code's loops, test on command line in a dev session first with the sdev or srun commands

Oak Storage for HPC

Oak:

An affordable storage for HPC- $42.95 per 10TB / month, billed monthly. The Oak storage system is mounted on Sherlock. SFTP and Globus connections are available. Oak is a parallel, capacity-oriented HPC storage system designed for long term storage . Aliased as $OAK on Sherlock.

Data on Oak is not backed-up. You can use cloud transfer tools such as rclone to send data to your Google Drive accounts. Note: there may be transfer/allocation limits for your Google accounts.

More info-

https://oak-storage.stanford.edu/

Sherlock User Limits

- **How long can I run?**
- **How many jobs can I submit?**
- **How many CPUs can I use at once?**

These limits can change, so view partition limits with **sacctmgr** or **sh_part** commands

sacctmgr show qos format=Name,MaxTRESPerUser,MaxSubmitJobsPerUser,MaxJobsPerUser,MaxTresPerAccount,MaxWall

Name	MaxTRESPU	MaxSubmitPU	MaxJobsPU	MaxTRESPA	MaxSubmitPA	MaxWall
normal	cpu=256	1000		cpu=512	2000	2-00:00:00
dev	cpu=4,mem=16G	2		cpu=99999	32	02:00:00
long	cpu=32	20	16		40	7-00:00:00
bigmem	cpu=128,mem=4T	10		mem=6T	20	1-00:00:00
gpu	gres/gpu=16	50		gres/gpu=24	100	2-00:00:00
owner*	cpu=99999	3000		cpu=99999	5000	7-00:00:00
owners	cpu=8192	3000		cpu=16384	5000	2-00:00:00

When you see srun: error: Unable to allocate resources: Requested node configuration is not available

It's because your job request went over these limits. Note that some limits apply group-wide (Max CPUs/account) In SLURM PI group=Account

Tip- Minimizing jobs in the queue

For jobs >2 days use QOS long (#SBATCH --qos=long) However, no needed if you're in an owner's group,

*If your PI is an owner, you will see this with sh_part command

User Limits cont.

MaxTRESPU Maximum number of CPUs/GPUs a user can use at once
MaxSubmitPU Maximum number jobs a user can submit at once
MaxTRESPA Maximum number number of CPUs/GPUs your PI group (account) can use at once
MaxSubmitPA Maximum number jobs your PI group (account) can submit at once
MaxJobsPU Maximum number jobs a user can have running at any given time
MaxWall Maximum time a job can run

Sometimes upon submitting your job or when viewing your squeue output you see:

MaxCpuPerAccount

This means that others in your group are currently running on all the CPUs that are available for them to use.

To see who is running these jobs in your group run-

$squeue -A **ruthm** -o "%.18i %.9P %.8j %.8u %.2t %.C %.L"

(where **ruthm** is your PI/Faculty sponsor/group name)

Find your PI group name with:

$id -gn

Check your CPU/memory/time/job limits with the **sh_part** command

Error messages

SLURM/sbatch/application error messages can be a bit hard to understand.

- Always try to Google the error messages, helpful to add “sbatch” or “SLURM” to the search. For example “tensorflow sbatch”
- When sending an email to srcc-support@stanford.edu always include relevant info, commands used, error messages, your sbatch file
- Don’t hesitate to ask us for help

Sherlock troubleshooting tips

Fairshare

Basically the more resources you use- CPU/RAM/time/nodes in a 2 week sliding window the lower your Fairshare score is which means the more likely your jobs will wait in the queue when other user's jobs are running.

or

A resource scheduler ranks jobs by priority for execution. Each job's priority in queue is determined by multiple factors, among them the user's fairshare score. A user's fairshare score is computed based on a target (the given portion of the resources that this user should be able to use) and the user's effective usage, *i.e.* the amount of resources (s)he effectively used in the past. As a result, the more resources past jobs have used, the lower the priority of the next jobs will be. Past usage is computed based on a sliding window and progressively forgotten over time. This enables all users on a shared resource to get a fair portion of it for their own use, by giving higher priority to users who have been underserved in the past.

Sherlock also uses *backfill*, smaller jobs can go in front of larger jobs, often regardless of the users Fairshare factor, thus increasing our clusters utilization. So if you can, use less (CPU/RAM/nodes/time)

Fairshare

Fairshare scores and your job's place in the queue is a moving target- Jobs end for various reasons:

SLURM has no idea when a user's application launched by srun/sbatch will exit.

It can *only* know what time was requested with --time/-t in the srun/sbatch. If an application ends before the time allocation request then SLURM obviously can't predict or know this and your place in the queue will change accordingly.

A user may cancel jobs, this will free up resources, sometime many resources (CPUs/memory) are freed when a user scancel's jobs or the application/script called in that job's sbatch ends

Thus sshare -a -A <your group name> changes constantly along with squeue output

Common software pre-installed on Sherlock

We take care of a lot of installations-
Matlab
R
Python
Stata-mp
K-nitro
Gurobi

Sherlock provides 431 software packages, in 7 categories, covering 77 fields of science

A complete listing of all modules on Sherlock

Search for modules you need with the module spider command

All stored as modules-
$**module avail**
--- math -- numerical libraries, statistics, deep-learning, computer science ---

R/3.4.0		py-keras/2.3.1_py36	(g)		
R/3.5.1	(D)	py-numpy/1.14.3_py27	(D)		
R/3.6.1		py-numpy/1.14.3_py36			
armadillo/8.200.1		py-numpy/1.17.2_py36			
arpack/3.5.0		py-numpy/1.18.1_py36			
caffe2/0.8.1	(g)	py-onnx/1.0.1_py27			
cgal/4.10		py-pytorch/0.2.0_py27	(g)		
cudnn/5.1	(g)	py-pytorch/0.2.0_py36	(g)		
cudnn/6.0	(g)	py-pytorch/0.3.0_py27	(g,D)		
cudnn/7.0.1	(g)	py-pytorch/0.3.0_py36	(g)		
cudnn/7.0.4	(g)	py-pytorch/1.0.0_py27	(g)		
cudnn/7.0.5	(g)	py-pytorch/1.0.0_py36	(g)		

Module Commands

Module documentation on Sherlock

module load -"loads" the software, it temporarily updates your path ($PATH) so that when you call the code, it will execute
module purge to start fresh
module list to see what you have loaded
module info
module keyword
module spider to search
$module spider numpy*
will find all modules that match the numpy pattern

Note that modules are in *categories*. For example if you want to use numpy in your Python code you will load it with the math category-

$module spider numpy

For detailed information about a specific "py-numpy" package (including how to load the modules) use the module's full name.

$module spider py-numpy/1.18.1_py36

$module load math py-numpy/1.18.1_py36

Sometimes you don't want certain dependent modules that are loaded with the one you want, so use **module -** (module with a "minus sign)

module load python/2.7
$ pip2.7 install --user numpy==1.11.0

unload the py-numpy module that is automatically loaded as a dependency:
$ module load -py-numpy

Scheduling Jobs*

Why Do We Need to Schedule a Job?

Resource contention between users needs to be balanced. So, the compute resources are managed and workloads are balanced using a job scheduler- SLURM.

How Easy Is It to Schedule a Job?

Basic concept - tell the scheduler:

1. What resources you need- CPUs, RAM, time, partition
2. What it should do- load modules, run your code
3. Need to request as few resources as you need so your jobs pend for as little time as possible, profile jobs with top, htop sacct

*Job= an instance of your program submitted to the scheduler (SLURM)

Sample Batch Job

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --time=10:00
#SBATCH -p normal
#SBATCH -c 1
#SBATCH --mem=8GB
# below you run/call your code, load modules, python, Matlab, R, etc.
# and do any other scripting you want
# lines that begin with #SBATCH are directives (requests) to the scheduler-SLURM
module load python/3.6.1
module load py-keras/2.2.4_py36
python3 mycode.py
```

Edit with vim/nano/vi/OnDemand file manager , save the file as test.sbatch
To run:
$**sbatch** test.sbatch
To watch:
$ **squeue** -u $USER
Many ways to control jobs as they run, scontrol pause/update, scancel

Output and error files will be placed the same directory that your sbatch script was run in.
slurm-916753.out
slurm-916753.err
Look in these files while debugging

https://youtu.be/GXmnS_rY_88

sbatch file format is important

Slurm will ignore all #SBATCH directives after the first non-comment line (the first line in the script that doesn't start with a # character).

- **Always put your #SBATCH parameters at the top of your batch script.**
- **Spaces in parameters will cause #SBATCH directives to be ignored**

Slurm will ignore all #SBATCH directives after the first white space. For example directives like these:

#SBATCH --job-name=big job

#SBATCH --mem=16 GB

#SBATCH --partition=normal, owners

will cause all following #SBATCH directives to be ignored thus the job is submitted with the default parameters (6GB, 1 CPU, 2 hours, normal partition). More info

sbatch flags, full and abbreviated

SLURM gives you a choice for some options; full or abbreviated. For example if you want the job to run for exactly 3 hours, in your sbatch file:
#SBATCH --time=3:00:00
or
#SBATCH -t 3:00:00

Full Option	Abbreviated	Description
--job-name=	-J	Give your job a name
--partition=	-p	Partition to run on
--nodes=	-N	Total number of nodes (use for MPI or codes that communicate across nodes)
--ntasks=	-n	Number of "tasks". Use if your code can do distributed parallelism
--cpus-per-task=	-c	# of CPUs allocated to each task. For use with shared memory parallelism.
--ntasks-per-node=		Number of "tasks" per node, use with distributed parallelism.
--time=	-t	Maximum walltime of the job in the format D-HH:MM:SS (e.g. --time=1- for one day or --time=4:00:00 for 4 hours)
--constraint=	-C	specific node architecture (if applicable)
--mem-per-cpu=		Memory requested per CPU (e.g. 10G for 10 GB)
--mail-user=		Mail address (alternatively, put your email address in ~/.forward)
--mail-type=		Control emails to user on job events. Use ALL to receive email notifications at the beginning and end of the job.

sbatch vs srun

sbatch command

- submits your job to the queue for later execution.
- non-blocking, you can submit with sbatch and logoff the cluster
- $sbatch jobscript.sbatch

srun command

- srun is used to submit a job in real time, useful for debugging
- used to get a interactive session and resources
- $srun -c 2 --mem=32GB --time=3:00:00 -p normal --pty bash
- used within an sbatch for job arrays
- sdev is a type (wrapper) of srun

srun in the submission script will create SLURM job steps. srun is used to launch the processes. If your program is a parallel MPI program, srun takes care of creating all the MPI processes. If not, srun will run your program as many times as specified by the --ntasks= option.

Bottom line: unless your application is multithreaded or MPI enabled don't bother calling srun in your sbatch script. Read the docs on your applications to check if they can support multithreading or MPI.

More on SLURM commands

Parallel example with SLURM Job Arrays

Job arrays offer a mechanism for submitting and managing collections of similar jobs quickly and easily; job arrays with thousands of tasks can be submitted in milliseconds (subject to configured size limits). All jobs must have the same initial options (e.g. size, time limit, etc). Array jobs are usually limited to 1000 steps.

Here only one job is submitted to the schedular with 24 array steps. Note the files are named to match array task ID ($SLURM_ARRAY_TASK_ID)

```
# !/bin/bash
#SBATCH --job-name=array_zip # Job name
#SBATCH -p owners
#SBATCH --ntasks=1 # Run a single task
#SBATCH --mem-per-cpu=1gb # Memory per processor
#SBATCH --time=00:10:00 # Time limit hrs:min:sec
#SBATCH --output=array_%A-%a.out # Standard output and error log
#SBATCH --array=1-24 # Array range, how many steps or times you want to run your app, in this case gzip
#Do your work here
gzip SRR062634.$SLURM_ARRAY_TASK_ID.filt.fastq
```

Same thing can be done with parameter values or other arguments to your code- map arguments to your code with $SLURM_ARRAY_TASK_ID

Note: The --ntasks parameter is only useful if you have commands that you want to run in parallel within the same batch script, i.e. your code is multithreaded or MPI enabled.

This example and the next slide's are examples of an embarrassingly parallel problem- little or no effort is needed to separate the problem into a number of parallel tasks. There is no dependency or need for communication between the parallel tasks or for the results between them. These are some of the most common types of jobs run on Sherlock.

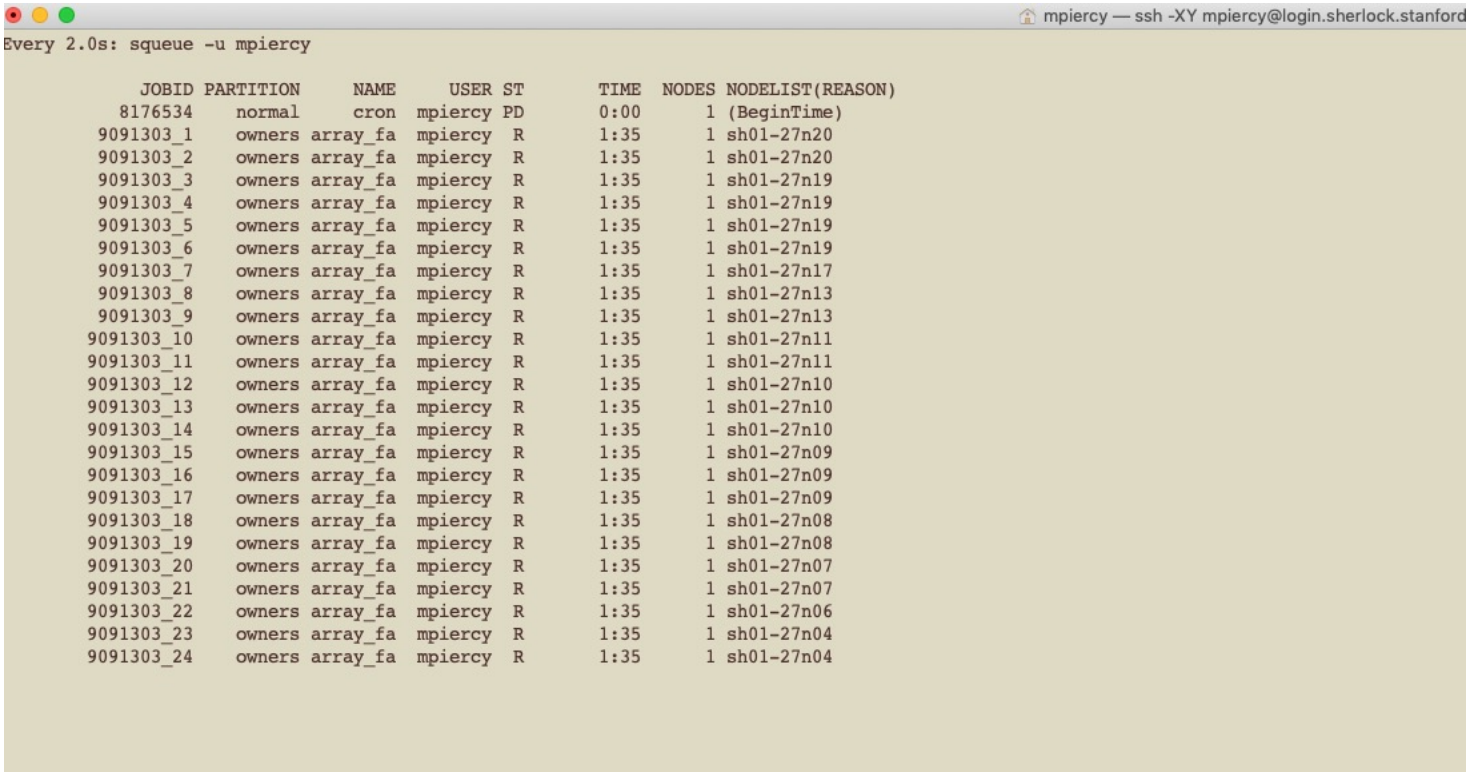
Job Arrays- creating array indexes

Here I have named my input files to match $SLURM_ARRAY_TASK_ID so they can be indexed by the job and each file (task) launched independently. You can do the same with any arguments or parameters to your code, python/MATLAB/R script

```
[mpiercy@sh01-ln01 login ~/tasks_test]$ ls -ltr
total 2261248
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.1.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.2.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.3.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.4.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.5.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.6.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.7.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.8.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.9.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.10.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.11.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.12.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.13.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.14.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.15.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.16.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.17.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.18.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.19.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.20.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.21.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.22.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.23.filt.fastq
-rw-r--r-- 1 mpiercy ruthm 80755971 Sep 23  2019 SRR062634.24.filt.fastq
-rw-r--r-- 1 mpiercy ruthm      176 Oct  9  2019 sruntasks.sbatch
-rw-r--r-- 1 mpiercy ruthm      370 Oct  9  2019 tasksets.sbatch
-rw-r--r-- 1 mpiercy ruthm      149 Oct  9  2019 tasks.sbatch
-rw-r--r-- 1 mpiercy ruthm      122 Oct  9  2019 ctasks.sbatch
-rw-r--r-- 1 mpiercy ruthm      267 Oct  1 15:38 array_fastq.sbatch
[mpiercy@sh01-ln01 login ~/tasks_test]$
```

Job arrays- squeue output

Below you can see the 24 array steps all launching across the cluster at once on different nodes



Embarrassingly Parallel Example with Job Arrays

A very simple example, you have 384 files to zip-

```
#!/bin/bash
#SBATCH --array=1-384%10
#SBATCH -n 1
#SBATCH -p owners
#SBATCH -t 5:00
gzip SP1_${SLURM_ARRAY_TASK_ID}.fq
```

Note that #SBATCH --array=1-384%10 will tell the schedular “submit my application- in this case gzip, 384 times in chunks of 10 jobs at a time. This is optional, you can leave out %10. One reason to limit the submissions is that you may encounter a limit for max # CPUs a user or group can run at once (MaxTRESPerUser, MaxTresPerAccount). See slide 20. Also, note that the files can be renamed to match the array step numbers with a shell loop. (for i in `seq 1 384`; do cp SP1.fq SP1_${i}.fq; done).

- Any parameter value, argument to your code can be used as an array step with ${SLURM_ARRAY_TASK_ID}.

- Rather than being run serially on 1 or 2 CPU’s on your laptop, on a cluster there are often thousands of CPUs so all 384 files (jobs) are processed (submitted to the scheduler with the sbatch command) at once. The scheduler needs to allocate jobs->resources.
- File I/O will be faster, clusters use a parallel filesystem, Lustre
- On Sherlock you can run on up to 256 (8,196 for owners) CPUs at once

srun SLURM tasks- an example of resource control

```
#!/bin/bash
#SBATCH --ntasks=8
#SBATCH --time=1:00
#SBATCH --mem=8GB
## can add more sbatch options above
echo hello from $SLURM_JOB_NODELIST
Output:
hello from sh-30-02
```

In SLURM terminology, a task is an instance of a running program.

If your program supports communication across computers (MPI) or you plan on running independent tasks in parallel, request multiple tasks with --ntasks= , the default value is set to 1.

Your program will require a certain amount of memory to function properly. To see how much memory your program needs, you can check the documentation or run it in an interactive session and use the **htop** command to profile it. To specify the memory for your job, use the **mem-per-cpu** option.

Sherlock has **defaults** to make job submission easier for users- 1 CPU, 6GB RAM, 2 hours. ***If unsure, always start with the defaults i.e. don't ask for any resources.***

srun SLURM tasks example cont.

change last line to:
srun echo hello from $SLURM_JOB_NODELIST
Output:
hello from sh-27-[17,20]
hello from sh-27-[17,20]
hello from sh-27-[17,20]
hello from sh-27-[17,20]
hello from sh-27-[17,20]
hello from sh-27-[17,20]
hello from sh-27-[17,20]
hello from sh-27-[17,20]

A task in SLURM analogous to a process in Unix, i.e. a running instance of a program with it's own memory and CPU allocation.

Task allocations are controlled by the user via SLURM.

You can see in the last example that the command was not only run 8 times (8 tasks) but run across the cluster on 2 different nodes (sh-27-17, sh-27-20). Used for multithreaded applications and MPI. SLURM allows **a lot** of resource granularity. For example if you want one process that can use 16 cores for multithreading use srun with:

--ntasks=1
--cpus-per-task=16

Matlab multicore example with MATLAB's parfor

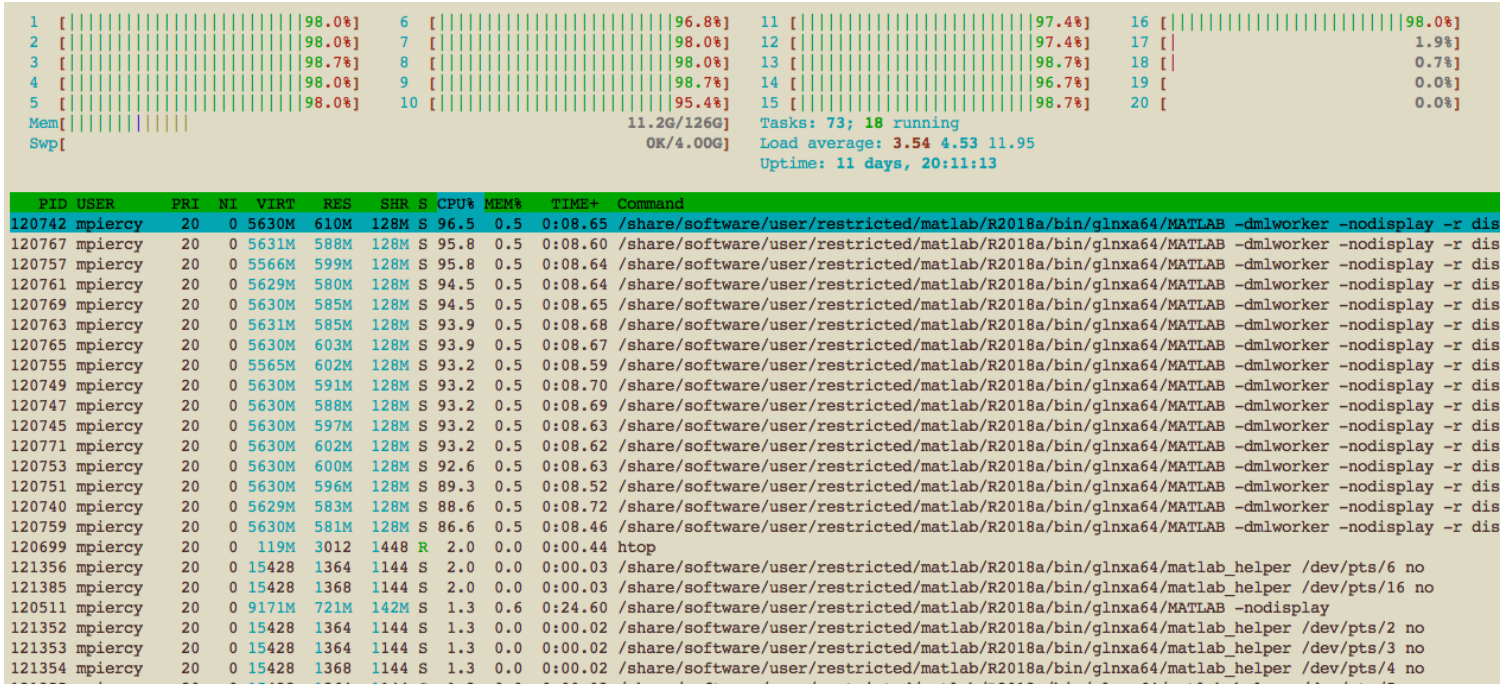
```
#!/bin/bash
#SBATCH -c 16
#SBATCH -t 15:00
#SBATCH -p hns
module load matlab
echo $SLURM_CPUS_PER_TASK
```

submit:

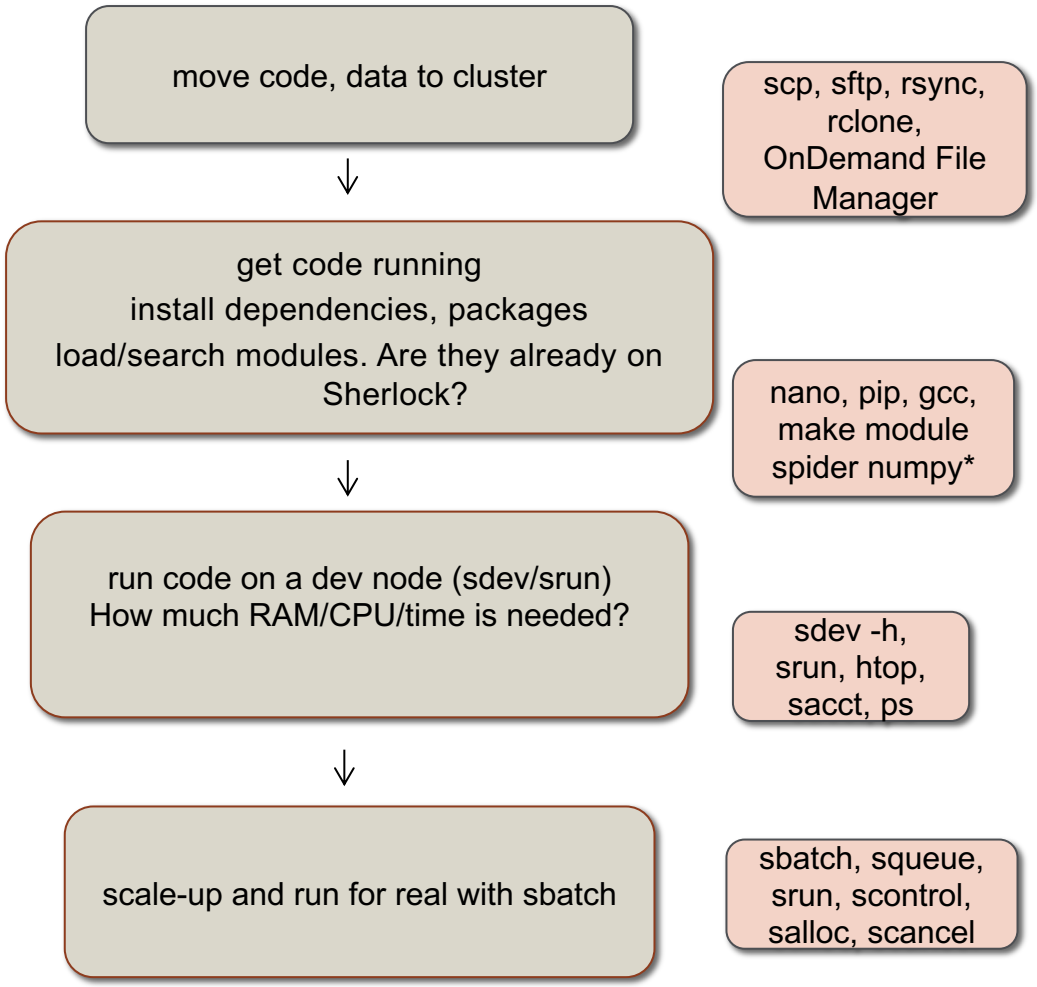
$srun -c $SLURM_CPUS_PER_TASK matlab -nosplash -nodesktop -r "pfor"

As a rule of thumb, the number of CPUs requested by a job should always match the number of threads or processes it will start.

So here we use -c 16 giving this job 16 CPUs



Typical HPC Cluster Workflow



Typical Sherlock workflow

1. move your code and data to Sherlock
2. test debug and install packages/software, load modules

ml load python
pip install <package> --user

ml load python/3.6.1
pip3 install <package> --user

Is the software already on Sherlock? Chances are if it's popular we have it.

module spider numpy*

R packages you can install yourself
If you have a lot of R packages to add, try to install on the command line rather than with the R Studio GUI.

For some compiled applications, install, compile with **module load gcc**, so don't worry about error messages stating that you
need to be root/have sudo to install since you can install in directories you control.

Typical Sherlock workflow cont.

3. If your code runs fine, test it with some data on an actual compute node (so you are not limited by login node memory limits (cgroups):

run **sdev/srun** command

How much CPU/RAM/time do I need?

Always try the defaults (by not requesting CPUs/RAM/time in your sbatch/srun)

On Sherlock defaults are: 1 CPU, 6GB RAM and 2 hours , normal partition

See how much memory your code/job used

While job is running use **sstat** or **scontrol show job <jobID>**

sstat --format
JobID,NTasks,nodelist,MaxRSS,MaxVMSize,AveRSS,AveVMSize 66807122

After jobs completes use sacct

sacct -j 66808759 --format
JobID,NTasks,nodelist,MaxRSS,MaxVMSize,AveRSS,AveVMSize

A typical example of user installed and compiled from GitHub

Here I want to install and compile seqtk

1. git clone https://github.com/lh3/seqtk.git
2. cd seqtk
3. make

Permanently add the seqtk binary location to you path:

4. echo 'export PATH=$PATH:$GROUP_HOME/$USER/seqtk' >> $HOME/.bashrc
5. source $HOME/.bashrc
6. seqtk

A typical example of user installed and compiled software, modifying library path

Here I want to install and compile WFDB

1. wget https://archive.physionet.org/physiotools/wfdb.tar.gz
2. tar xfvz wfdb.tar.gz
3. cd wfdb-10.6.2

4. In your .bashrc add the path to the directory where you install it.
In this case I'm installing in $GROUP_HOME,
So update your LD_LIBRARY_PATH:

export LD_LIBRARY_PATH=$GROUP_HOME/$USER/wfdb-10.6.2
To permanently update your .bashrc -
echo export LD_LIBRARY_PATH=$GROUP_HOME/$USER/wfdb-10.6.2 >> $HOME/.bashrc

5. ml load gcc
6. (in my $GROUP_HOME I made the directory WFDB, you can choose your $HOME also)
7. ./configure --prefix=/home/groups/ruthm/mpiercy/wfdb-10.6.2/WFDB
8. make
9. make install
10. make check

Estimating your codes resources with sdev and htop

1. sdev (or srun --pty bash)
2. load modules, run code in background

$python mycode.py > /dev/null 2>&1 &

$htop

htop to see only your processes-

type u then your login name, hit return

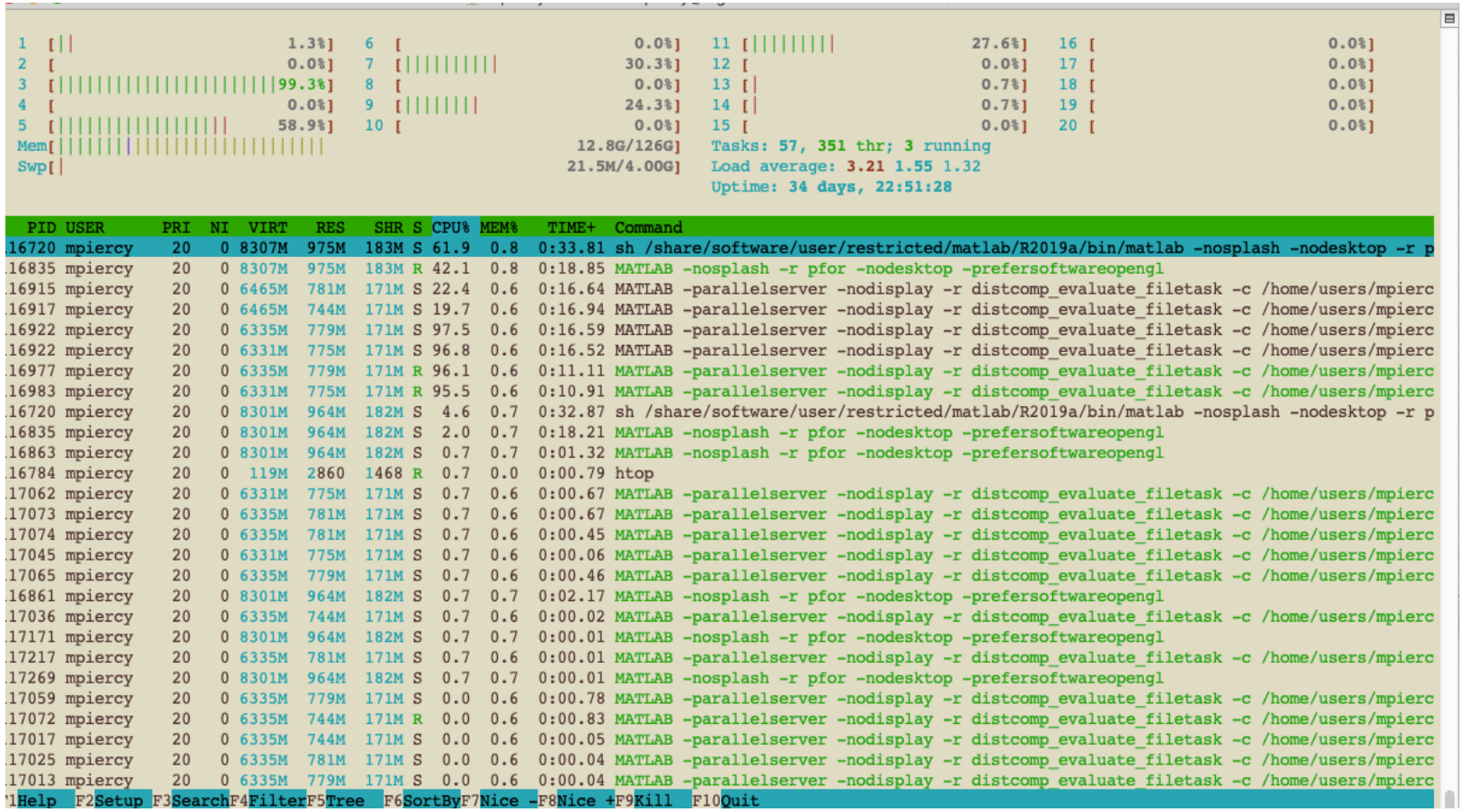
or just-

$htop -u <your user name>

You will see how many CPUs, threads and how much RAM your application is using in real-time.

htop example on a dev or compute node

$srun -c 4 matlab -nosplash -nodesktop -r "pfor" > /dev/null 2>&1
$htop



Estimate your job's resource requirements with sacct

sacct -o reqmem,maxrss,averss,elapsed -j 20222292

ReqMem	MaxRSS	AveRSS	Elapsed
1024Mn			00:00:10
1024Mn	579K	579K	00:00:10
1024Mn	90K	90K	00:00:10
1024Mn	524K	524K	00:00:05

ReqMem = memory that you asked from SLURM. If it has type Mn, it is per node in MB, if Mc, then it is MB per core
MaxRSS = maximum amount of memory used at any time by any process in that job. This applies directly for serial jobs. For parallel jobs you need to multiply with the number of cores (max 16 or 24 as this is reported only for that node that used the most memory)
AveRSS = the average memory used per process (or core). To get the total memory need, multiply this with number of cores
Elapsed = time it took to run your job

sacct -o reqmem,maxrss,averss,elapsed,alloccpu -j 426651

ReqMem	MaxRSS	AveRSS	Elapsed	AllocCPUS
4Gn			00:08:53	1
4Gn	3552K	5976K	00:08:57	1
4Gn	2921256K	2921256K	00:08:49	1

Here, the job came close to hitting the requested memory, 4 GB, 2.92 GB was used. Note that SLURM only samples a job's resources every few minutes, so this is an average. Jobs with a MaxRSS close to ReqMem can still get an out of memory (OOM event) error and die. When this happens request more memory in your sbatch with --mem=

Estimate your batch job's resource requirements

sacct -o reqmem,maxrss,averss,elapsed,alloccpus -j 3413279

ReqMem	MaxRSS	AveRSS	Elapsed	AllocCpus
16000Mc			1-20:54:49	4
16000Mc	4771852K	4603220K	1-20:54:49	4

The first line is the parent job, second is the job step, the actual job.

You've requested 16GB per core, i.e. a total of 64 GB (4x16GB, everything in one node)

Your job has used a maximum of 4771852K i.e. 4.7 GB per core

You've requested more than 10 GB too much memory per core i.e. about 50 GB too much in total

So, ask for less memory for this kind of job, e.g. --mem=8GB

sstat, srun- monitor resource usage as a job runs

sstat --format JobID,NTasks,nodelist,MaxRSS,MaxVMSize,AveRSS,AveVMSize 20267805

JobID	NTasks	Nodelist	MaxRSS	MaxVMSize	AveRSS	AveVMSize
20267805.0	1	gpu-27-21	393953K	1912732K	393017K	1912732K

Compare these values to what you requested in your sbatch file or srun command

You can also quickly monitor your job's memory and CPU usage as it run with srun and top

1. Find your job id with-

squeue –u $USER

2.

srun --jobid=1002961 top -b -n 1 -u $USER

Protip: Sort pending jobs in the queue by priority, the higher the number the sooner it will run. Look for your name in the queue. For example on the normal partition-

squeue -o "%8i %8u %15a %.10r %.10L %.5D %.10Q" -p normal --state=PD | more

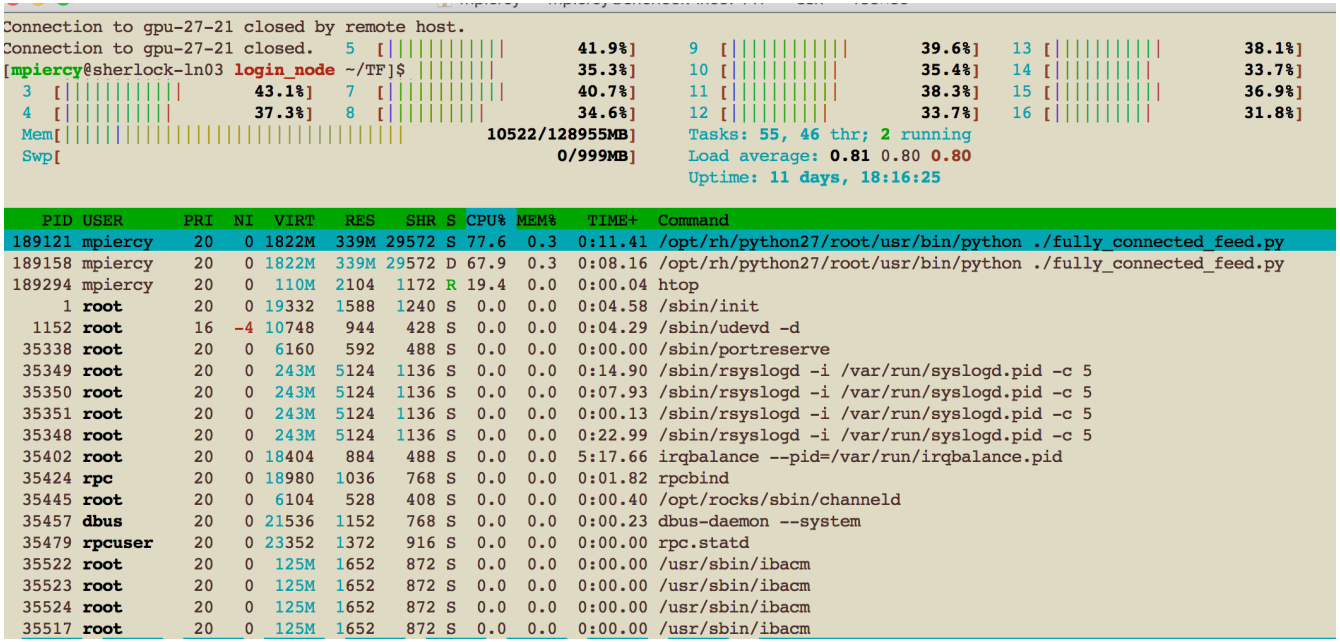
Estimating resources requirements, htop compute node while job is running

$ sbatch TF_mnist.sbatch
Submitted batch job 20244339
$ squeue -u $USER

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
20244339	hns_gpu	TF_mnist	mpiercy	R	0:04	1	gpu-27-21

$ ssh mpiercy@gpu-27-21

ssh to the server only of you have a job running on it, here it's gpu-27-21
$ ssh mpiercy@gpu-27-21
use **nvidia-smi -l 1** or **module load system nvtop** for GPU nodes



Videos

Create an sbatch file and submit it to the queue:
https://youtu.be/GXmnS_rY_88

Submit a MATLAB batch job:
https://youtu.be/ytAyF_KlpJc

Monitor a job:
https://youtu.be/89OK9pGnRJE

Show different Python environments:
https://youtu.be/DlqtQDriprQ

Installing and R package:
https://youtu.be/BshpaeJMAUs

Key Sherlock Links

Python
Matlab
R
submitting a job
Unix/Linux tutorials
modules
filesystems
Globus for large data transfers
Sherlock OnDemand: browser based access
moving data
Oak
job arrays
GPUs

Nice examples of –ntasks vs. --ntasks-per-node vs. --cpus-per-task
Creating a parallel environment with sbatch directives

Please Acknowledge SRCC

It is important that publications resulting from computations performed on Sherlock, Farmshare or SCG acknowledge this. The following wording is suggested for your Acknowledgements section:

"Some of the computing for this project was performed on the **Sherlock *(or XStream, Farmshare, Nero)*** cluster. We would like to thank Stanford University and the Stanford Research Computing Center for providing computational resources and support that contributed to these research results."

Many researchers have: SRCC acknowledged publications

Support

Documentation

Sherlock: http://www.sherlock.stanford.edu/

Sherlock Office Hours :
https://www.sherlock.stanford.edu/docs/overview/introduction/#office-hours

Office hours via Zoom, for the time being.
https://stanford.zoom.us/j/95962823750?pwd=cFM2U2ZRQ243Zkx0Ry83akdtWU9zUT09
Tuesday 10-11am
Thursday 3-4pm

Drop by or make an appointment : https://calendly.com/srcc-officehours/sherlock

Check out Events listings for more workshops:
https://srcc.stanford.edu/events/upcoming-events

Contact

Questions/Answers: srcc-support@stanford.edu

SRCC group: http://srcc.stanford.edu
Mark Piercy: mpiercy@stanford.edu