

Estimating job resources

When trying to estimate how many resources to submit for you can use the interactive srun command.

In a terminal connected to a login node, or in a new terminal launch a 5 core interactive job run via:

```
srun --cpus-per-task=5 --pty /bin/bash
```

Once you have landed on your new node. Go ahead and activate your newly created conda environment:

```
conda activate discovery_class
```

To get started, please copy a small zip folder containing some python code and a sample submit script.

```
cp /dartfs-hpc/admin/Class_Examples.zip . && unzip Class_Examples.zip
```

The above command will copy the .zip file called Class_Examples from the location /dartfs-hpc/admin . The . instructs the copy to your current working directory. The && instructs to run the next command, which is to unzip the contents of the folder into the directory you are in.

When estimating your resource utilization you can use a program like top to monitor current utilization.

In this tutorial lets open two terminals side by side.

In one terminal we will launch our python code from the folder we unzipped. In the other terminal we will run the command top -u <username> to look at resource utilization.

In your first terminal take notice of the host your srun job landed on. You can see this by the change in the prompt:

```
[john@t04 ~]$
```

In my case my interactive job landed on t04.

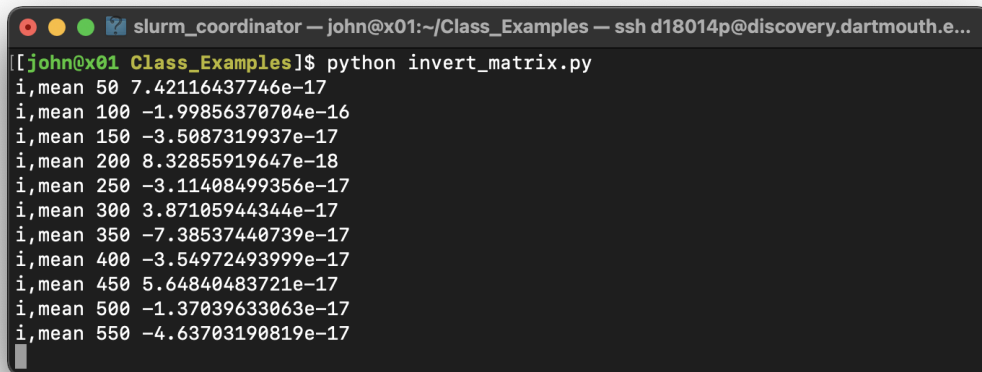
In my second terminal i'll ssh to t04 directly.

```
ssh t04
```

Now that we are setup with two terminals side by side on the same host. (one terminal a job, the other terminal direct ssh). Next, within the Class_Examples folder is a basic python script we will use for estimating resources. The script is called invert_matrix.py . Lets run the script from our first terminal, the one we created the interactive job in, and see what it does.

```
cd Class_Examples
```

```
time python3 invert_matrix.py
```



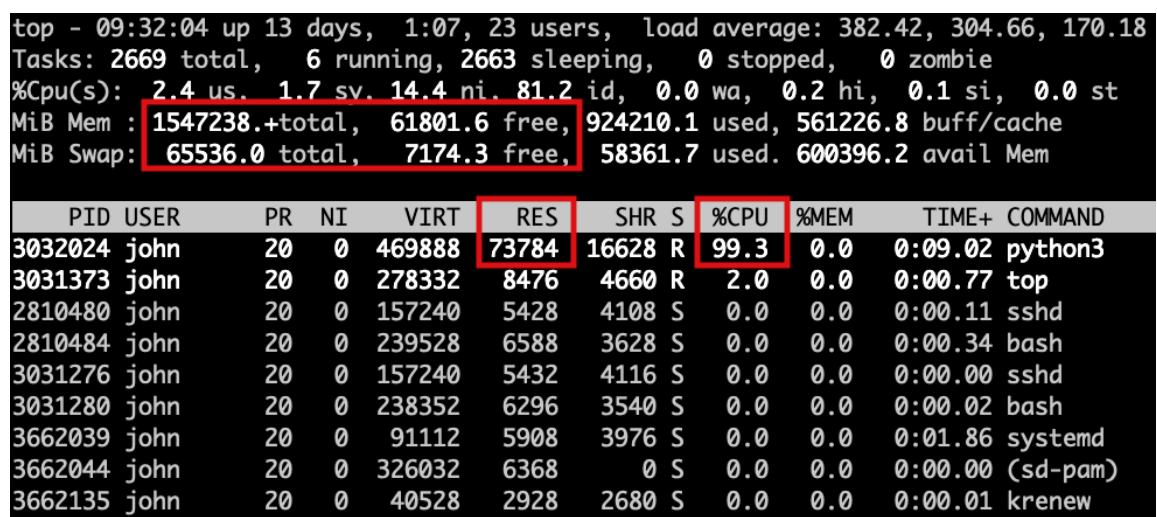
```
slurm_coordinator — john@x01:~/Class_Examples — ssh d18014p@discovery.dartmouth.e...  
[john@x01 Class_Examples]$ python invert_matrix.py  
i,mean 50 7.42116437746e-17  
i,mean 100 -1.99856370704e-16  
i,mean 150 -3.5087319937e-17  
i,mean 200 8.32855919647e-18  
i,mean 250 -3.11408499356e-17  
i,mean 300 3.87105944344e-17  
i,mean 350 -7.38537440739e-17  
i,mean 400 -3.54972493999e-17  
i,mean 450 5.64840483721e-17  
i,mean 500 -1.37039633063e-17  
i,mean 550 -4.63703190819e-17
```

Once your python command is executing like the above image, use the second terminal you opened to run the top command. My username is `john` so my command will be:

```
top -u $(whoami)
```

You can get out of top by simply hitting the letter `q`.

The next top screen will display information about the state of the system but also information like the number of CPUs, amount of system memory, and other useful information. In this case we are looking at two fields in particular. `CPU%` & `RES` short for reserved memory.



```
top - 09:32:04 up 13 days, 1:07, 23 users, load average: 382.42, 304.66, 170.18  
Tasks: 2669 total, 6 running, 2663 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 2.4 us, 1.7 sv, 14.4 ni, 81.2 id, 0.0 wa, 0.2 hi, 0.1 si, 0.0 st  
MiB Mem : 1547238.+total, 61801.6 free, 924210.1 used, 561226.8 buff/cache  
MiB Swap: 65536.0 total, 7174.3 free, 58361.7 used. 600396.2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3032024	john	20	0	469888	73784	16628	R	99.3	0.0	0:09.02	python3
3031373	john	20	0	278332	8476	4660	R	2.0	0.0	0:00.77	top
2810480	john	20	0	157240	5428	4108	S	0.0	0.0	0:00.11	sshd
2810484	john	20	0	239528	6588	3628	S	0.0	0.0	0:00.34	bash
3031276	john	20	0	157240	5432	4116	S	0.0	0.0	0:00.00	sshd
3031280	john	20	0	238352	6296	3540	S	0.0	0.0	0:00.02	bash
3662039	john	20	0	91112	5908	3976	S	0.0	0.0	0:01.86	systemd
3662044	john	20	0	326032	6368	0	S	0.0	0.0	0:00.00	(sd-pam)
3662135	john	20	0	40528	2928	2680	S	0.0	0.0	0:00.01	krenew

From the above the `CPU%` column is showing `97-99%`. That is equivalent to 1 CPUs. With this information I know to submit my job for 1 CPU in order for it to run efficiently.

In the other col RES we can see that we are not using quite a full GB of memory. We know from this output that requesting the minimum for a job of 8GB will be sufficient for our job. (or lower)

The next resource you should consider estimating before subming your job is `walltime` walltime is used to determine how long your job will run for. Estimating accurate walltime is good scheduler ettiquite.

```
slurm_coordinator — john@x01:~/Class_Examples/Class_Examples — ssh d18014p@discovery.dartmouth.edu...
[john@x01 Class_Examples]$ time python invert_matrix.py
i,mean 50 3.0809518348e-16
i,mean 100 -3.7160949375e-17
i,mean 150 -1.05117087569e-18
i,mean 200 8.4178085509e-17
i,mean 250 -1.05018255377e-17
i,mean 300 6.77238142087e-17
i,mean 350 5.09817948181e-17
i,mean 400 1.8705306401e-17
i,mean 450 3.8989132092e-17
i,mean 500 1.6705945904e-16
i,mean 550 1.51966715328e-17
i,mean 600 -2.13054171624e-17
i,mean 650 -5.63506852332e-16
i,mean 700 1.30304028112e-17
i,mean 750 1.80525314651e-17
i,mean 800 1.84999105167e-16
i,mean 850 3.50647660866e-17
i,mean 900 -9.99593418259e-18
i,mean 950 5.77903989178e-17
i,mean 1000 -3.32272525466e-17

real    1m22.611s
user    3m26.028s
sys     0m17.299s
[john@x01 Class_Examples]$
```

From the output above you will want to look at the `real` field. This is the time passed between pressing the enter key and the termination of the program. At this point, we know that we should submit for at least 5 minutes of walltime. That should allow enough time for the job to run to completion.

Note

Determining walltime can be tricky. To avoid potential job loss it is suggested to add 15-20% more walltime than jobs typically need. This will ensure jobs have enough walltime to complete the task. So if your job takes 8 minutes to complete, submit for 10.

Now that we have all of this information about the job we are ready to build our first submit script for submitting in batch to the scheduler.

```
#!/bin/bash

# Request 5 CPU's for the job
#SBATCH --cpus-per-task=1

# Request 8GB of memory for the job
#SBATCH --mem=8GB
```

```
# Walltime (job duration)
#SBATCH --time=00:05:00

# Then finally, our code we want to execute.
time python3 invert_matrix.py
```

Before we move to the next portion of submitting the job via sbatch, lets adjust the script to use 5 cores instead of 1.

So that you do not have to open an editor an updated version of the script is in the folder. Go ahead and take a look at the file `invert_matrix_5_threads.py` :

```
cat invert_matrix_5_threads.py
```

```
#!/usr/bin/python3
import os

# Set the number of threads to 5 to limit CPU usage to 5 cores
os.environ["OPENBLAS_NUM_THREADS"] = "5" # For systems using OpenBLAS

# Now import NumPy after setting environment variables
import numpy as np
import sys

# Main computation loop
for i in range(2, 1501):
    x = np.random.rand(i, i)
    y = np.linalg.inv(x)
    z = np.dot(x, y)
    e = np.eye(i)
    r = z - e
    m = r.mean()
    if i % 50 == 0:
        print("i,mean", i, m)
        sys.stdout.flush()
```

Notice the line at the top it is set to:

```
os.environ["OPENBLAS_NUM_THREADS"] = "5"
```

Lets go ahead an run that script now to see if adding 5 cores speeds it up:

```
time python3 invert_matrix_5_threads.py
```

Was it faster?