Making best use of your NeSI HPC allocation

Tips for optimising job configuration and reducing your queue time

Overview

In this talk you should learn the basics of:

- Which Slurm commands can help you view how efficiently your code ran
- What does an efficient job look like, and
- How to improve your jobs efficiency

What do I mean by Job Efficiency and why Does it Matter?

An Efficient Job

- Should utilise all resources available to it.
  - If a job has 10 CPUs for an hour but only utilises one hour of CPU time it is not efficient
- Should efficiently use the resources allocated to it.
  - A job could appear to be efficiently utilising CPUs and memory, but if the code is not well written it could still be inefficient
  - Code using a less efficient method to calculate something might still effectively utilise resources but take longer or use more than if it used a more efficient method
- We will mostly look at avoiding the first type of inefficiency

The HPCs are a Shared Resource:

- The larger a job is in any dimension the longer it can expect to wait.
- We are less inclined to provide allocation increases to projects with inefficient jobs.
- Inefficient work means your Fair Share score drops faster, which decreases your job priority
- It can affect the queue time of others on the platforms.

It is therefore necessary to accurately estimate your job's resource requirements or you risk wasting both time and resources.

But to do this you need to know if your jobs are inefficient in the first place.

Sacct

That’s where sacct comes in:

By default show you all jobs queuing, running, or has run since midnight. The most important default sacct columns are “Elapsed” (how long the job ran for), “Alloc” (the amount of allocated, logical CPUs), “MaxRSS” (the maximum amount of memory detected for that job step) and “TotalCPU” (the total non-idle CPU time).

- To look at a specific job use the -j flag
- To look at a jobs from an earlier time use the -S flag, followed by the earlier date in the YYYY-MM-DD format

```
sacct
        JobID    JobName      Elapsed   TotalCPU  Alloc     MaxRSS      State
------------ ---------- ---------- ---------- ---- ---------- ----------
12345678        Job          2-22:01:02 9-00:50:25   32                 COMPLETED
12345678.batch  batch        2-22:01:02 9-00:50:25   32   1956756K      COMPLETED
12345678.exte+  extern       2-22:01:02 00:00:00     32   0             COMPLETED
```

How CPU Efficient is my Job?

- A perfectly CPU efficient job will have TotalCPU equal to Elapsed multiplied by Alloc (we do not expect your jobs to be perfectly efficient)
- 2-22:01:02 multiplied by 32 is about 93 days.
- This jobs TotalCPU is 9 days, so it is only 10% efficient
- This indicates it might be reasonable to ask for fewer CPUs
- There is no set rule for what is CPU efficient enough, do your best to make them as efficient as possible.

```
sacct
       JobID    JobName        Elapsed   TotalCPU  Alloc     MaxRSS      State
------------ ---------- -------------- ---------- ------ ---------- ----------
12345678         Job       2-22:01:02 9-00:50:25    32              COMPLETED
12345678.batch   batch     2-22:01:02 9-00:50:25    32   1956756K   COMPLETED
12345678.exte+   extern    2-22:01:02 00:00:00      32   0          COMPLETED
```

What is causing the Inefficiency?

- The job could be serial, or hard coded to utilise fewer cores than requested
- The job could have parallel and serial parts
- The jobs CPUs could site idle waiting on I/O
- CPUs could be sitting idle waiting on other CPUs to complete tasks (load imbalance)

Be aware of Hyperthreading

- Hyperthreading is enabled on the NeSI machines, so for each physical CPU core, there are two logical CPUs.
- Hyperthreading can be turned off with `#SBATCH --hint=nomultithread`, but doing so will make your jobs appear half as CPU efficient.
- So a non-hyperthreaded job with 50% CPU efficiency is as efficient as a hyperthreaded job with 100% CPU efficiency.
- You should test your job both with and without hyperthreading, as some jobs perform better with it disabled.
- Learn more about hyperthreading on the NeSI platforms here: https://support.nesi.org.nz/hc/en-gb/articles/360000568236

How Memory Efficient is this Job?

- A job is generally considered memory efficient if at any time one of its job steps comes close to the requested memory, in this case 250 MB per CPU.
- Given there are 32 CPUs, and this is a shared memory job it is requesting 8,000 MB of memory.
- The maximum detected memory usage is ~1910 MB, about ~25% efficient
- A good rule of thumb is to ask for ~20% more memory than you think you need, so this job should be asking for ~2.3 GB, or 72 MB per CPU.
- Note: sacct is not infallible, memory is only detected every 30 seconds.

```
sacct
           JobID        JobName       Elapsed   TotalCPU  Alloc    MaxRSS        State
---------------- -------------- ------------- ---------- ------ ---------- ------------
12345678            Job           2-22:01:02 9-00:50:25   32               COMPLETED
12345678.batch      batch         2-22:01:02 9-00:50:25   32   1956756K    COMPLETED
12345678.exte+      extern        2-22:01:02 00:00:00     32   0           COMPLETED
```

How Time Efficient is this Job?

- A job is generally considered time efficient if its elapsed time is close to its requested walltime, 3 days in this example.
- Like memory requests it is a good idea to ask for ~20% more wall time than you expect to use.

```
sacct
          JobID       JobName        Elapsed  TotalCPU  Alloc     MaxRSS        State
-------------------- --------------------- -------------------- --------------- ------- --------------- -------------------
12345678             Job              2-22:01:02  9-00:50:25    32                COMPLETED
12345678.batch       batch            2-22:01:02  9-00:50:25    32   1956756K    COMPLETED
12345678.exte+       extern           2-22:01:02  00:00:00      32   0           COMPLETED
```

Custom Formatted Sacct Commands

- Default sacct assumes pre-existing knowledge of jobs
- This is usually fine, but it can be helpful to have all relevant information in sacct output.
- So why not format your own sacct command?
- You can find additional sacct options than those we show by using the “man sacct” command.

Custom Formatted Sacct Commands

sacct --format=JobID,JobName,Timelimit,Elapsed,Alloc,TotalCPU,ReqMem,MaxRSS,nTasks,State,account,partition

```
       JobID    JobName  Timelimit    Elapsed  AllocCPUS   TotalCPU     ReqMem     MaxRSS   NTasks      State    Account  Partition
------------ ---------- ---------- ---------- ---------- ---------- ---------- ---------- -------- ---------- ---------- ----------
12345678     Job        3-00:00:00 2-22:01:02         32 9-00:50:25     250Mc                        COMPLETED  nesi99999      large
12345678.ba+ batch                 2-22:01:02         32 9-00:50:25     250Mc   1956756K        1  COMPLETED  nesi99999
12345678.ex+ extern                2-22:01:02         32 00:00:00       250Mc          0        1  COMPLETED  nesi99999
```

You can set a custom formatted sacct command as your default by exporting it in your home directories .bashrc

- `nano /home/user001/.bashrc`
- Paste and save:
  export SACCT_FORMAT=JobID,JobName,Timelimit,Elapsed,Alloc,TotalCPU,ReqMem,MaxRSS,nTasks,State,account,partition

Job Profiling

Profiling

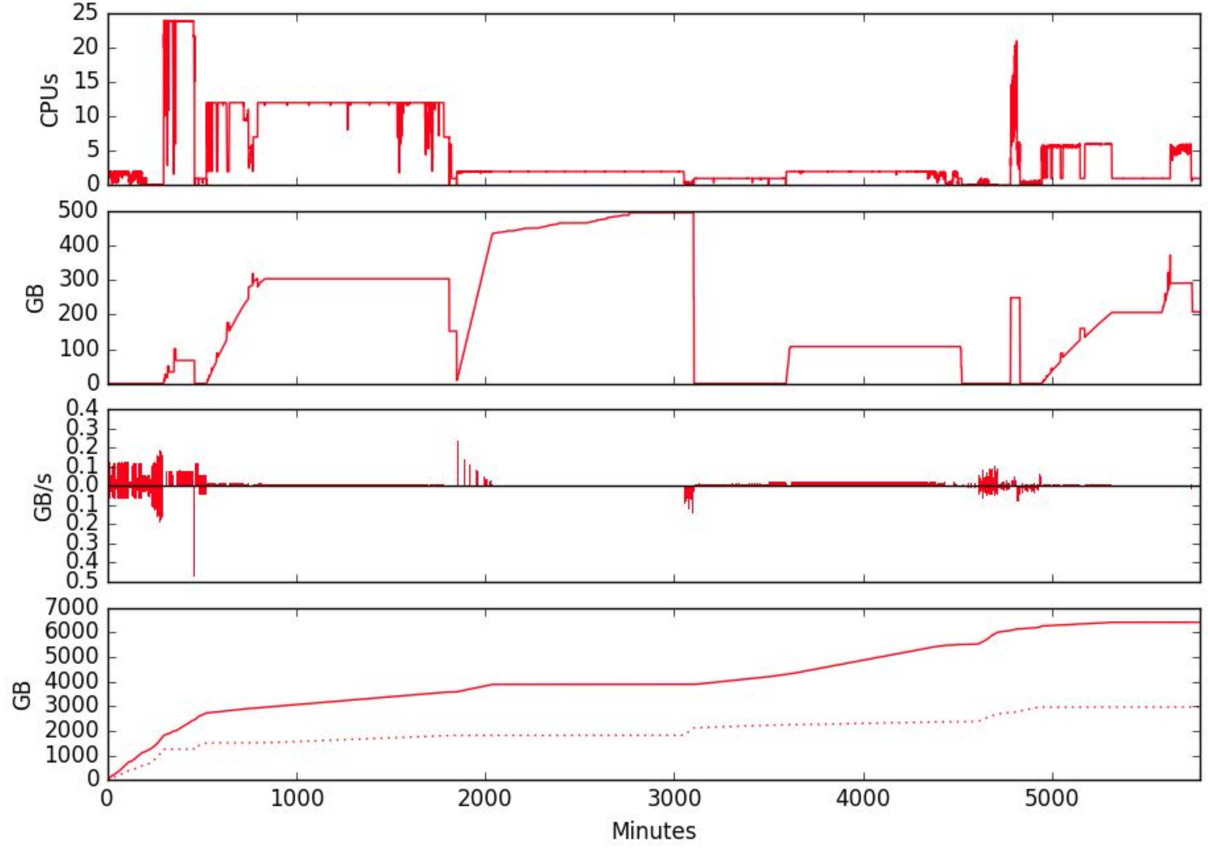
Profiling can detect things that sacct can't:

- sacct only gives the maximum detected memory and average CPU usage, profiling breaks it down over time.
- Breakdown of CPU and memory usage can be used to find job break points to make more efficient jobs.
- Profiling also gives read/write information.
- All this makes in depth assessment of your code efficiency easier.
- Additionally, software specific profilers like a MAP for C/C++, Fortran and Python code can even pinpoint which lines of code are inefficient or computationally intensive.

How to use the Slurm Profiler

1. Add the line `#SBATCH --profile task` to your Slurm script
2. After your job completes you will need to collate the profile data into a HDF5 file using the command `sh5util -j <jobid>`
3. Use the Python or MATLAB script found on our support page to plot your results: https://support.nesi.org.nz/hc/en-gb/articles/360000810616

Note: the Slurm profiler is not enabled on Maui job_60154472.h5



What do you do now?

- Investigate other profilers on NeSI:

  https://support.nesi.org.nz/hc/en-gb/articles/360000930396
  https://www.youtube.com/watch?v=b1cpCeksWXw

- Now that you know how to determine if a job you've already run is inefficient, learn how to predict jobs resource requirements before you run them:

  https://support.nesi.org.nz/hc/en-gb/articles/360000728016

  https://www.youtube.com/watch?v=CqATGcNbipo&feature=youtu.be

Questions