SHARCNET
General Interest Webinar Series

Conquering the Scheduler
Starting @ ~12:05pm

Tyler Collins
High Performance Computing Consultant
SHARCNET, Brock University
Feb 9th, 2022

Compute • Calcul Ontario

SHARCNET™

compute canada | calcul canada

The Wiki

Before we go any further…

Always always always check the wiki

No, really… please

https://docs.computecanada.ca/wiki/Compute_Canada_Documentation

If you can’t find something, contact us!

support@computecanada.ca

Today’s Outline

Goal: understand how to get better throughput from the scheduler

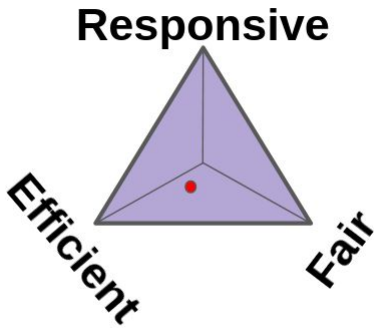
1. Definitions
    a. Core Equivalent / Billing
    b. Fairshare and Priority
    c. Partitioning
2. Assumptions
3. Conquering!
    a. Easy wins
    b. MPI, GLOST, META, etc
4. Final Considerations
5. Open Discussion

Definitions: Core Equivalent and Billing

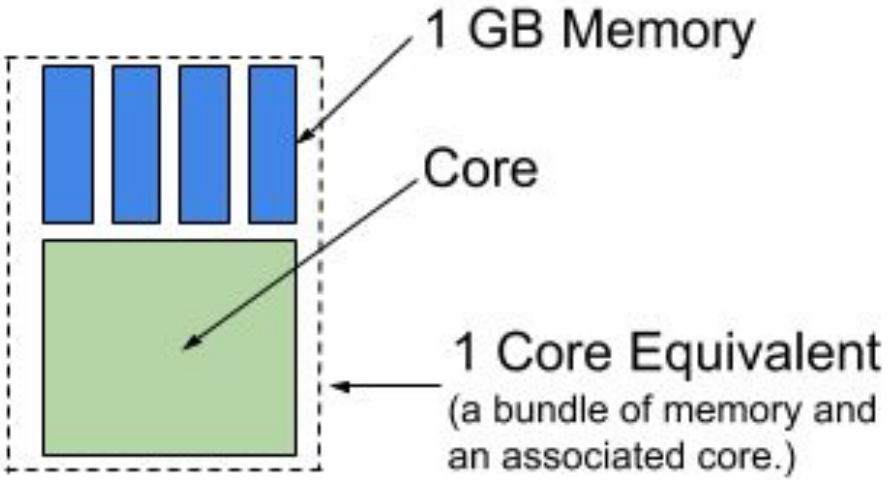
Formal definitions and ratios can be found on the wiki

Rationale:

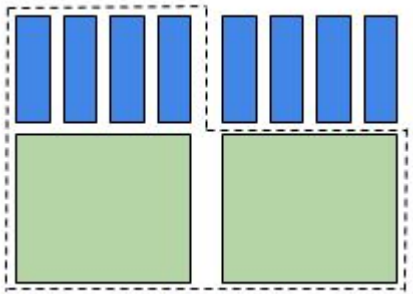
- If a job specifies an entire node's memory but only a single core, it has **isolated all other jobs from running on that node**
- The job should be billed accordingly
- This function is known as the **Core Equivalent**
- Can be seen in slurm as "billing"
  - Does **not** imply any monetary cost!
- Some examples to follow



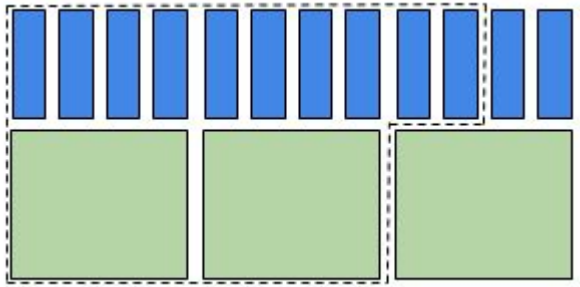
Definitions: Core Equivalent and Billing



Definitions: Core Equivalent and Billing



2.0 CE

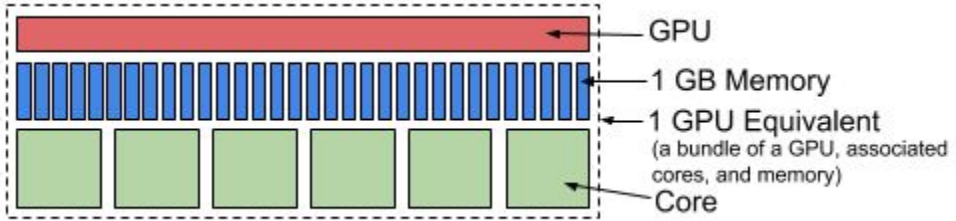


2.5 CE

GPU Equivalent?

Short answer: yes

Long answer:



Definitions: Fairshare and Priority

Formal definitions can be found here

- Fairshare:
  - The “slice” of the system your group is entitled to
  - This value **fluctuates** based on recent usage
- Priority: Function of an **group’s** fairshare combined with job size
  - RAC accounts typically have larger fairshare’s
  - Time in queue **does** increase priority, however it is almost a totally negligible amount
- In general: the more you have used lately, the less priority you have
  - Gives other users their “turn”
  - A half-life decay exists that enables recovery of fairshare
  - **NOT A BANK ACCOUNT**

Definitions: Fairshare and Priority

How do we see these values and make decisions based on them?

```
[prof1@gra-login4 ~]$ sshare -l -A def-prof1_cpu -u prof1,grad2,postdoc3
       Account       User  RawShares  NormShares  RawUsage  ... EffectvUsage  ...    LevelFS  ...
-------------- ---------- ---------- -----------  --------  ... ------------  ... ----------  ...
def-prof1_cpu                 434086    0.001607   1512054  ...     0.000043  ...  37.357207  ...
 def-prof1_cpu      prof1          1    0.100000         0  ...     0.000000  ...         inf  ...
 def-prof1_cpu      grad2          1    0.100000     54618  ...     0.036122  ...    2.768390  ...
 def-prof1_cpu   postdoc3          1    0.100000    855517  ...     0.565798  ...    0.176741  ...
```

LevelFS is the most important column; see [wiki] for formal definitions

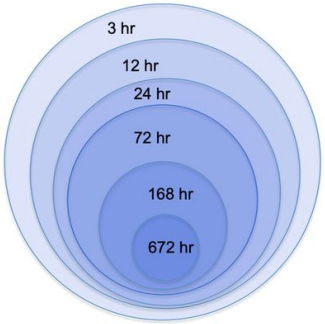
- Value **>1**: under served, should see fast responsiveness
- Value **0 < x < 1**: well served, may see wait times
- As the value approaches **0**, more wait times will be seen

Definitions: Partitioning

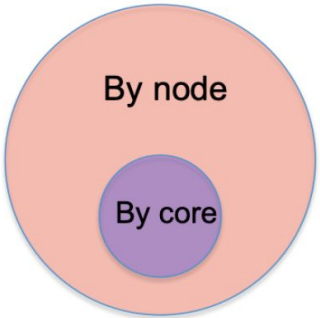
The general purpose systems contain different resource amounts dedicated to serving particular types of job requests

This is to maximize the responsiveness of the cluster

Nodes reserved for short Jobs

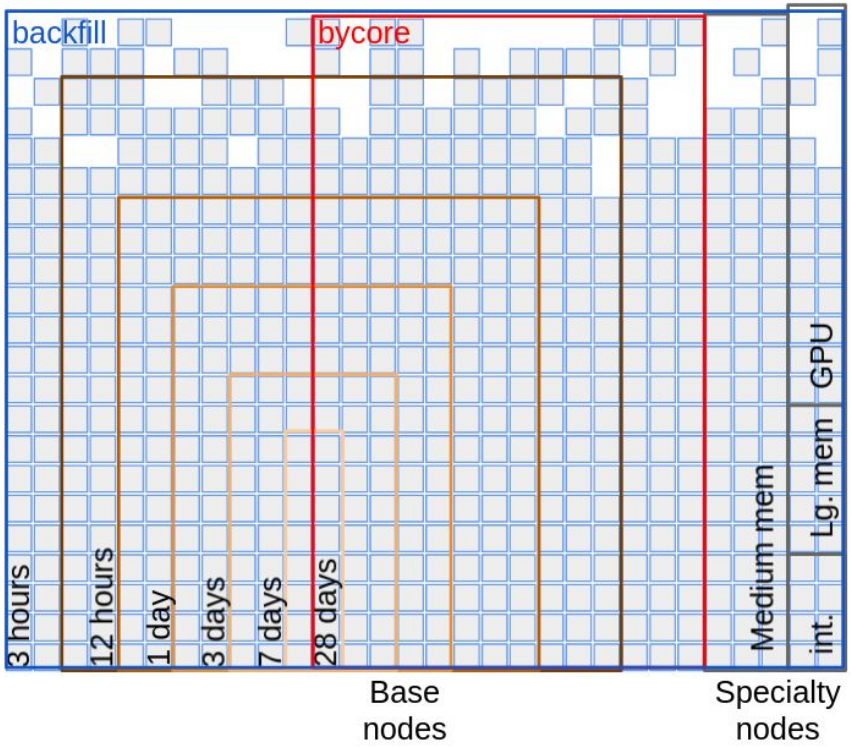


Nodes reserved for whole node Jobs



Definitions: Partitioning

System partitioning at a glance…



Observations:

- Look how few large memory resources!
- 28 days by core is really competitive…
- Backfill seems great!
  - Read the wiki
  - In general, smaller jobs that are used to “pack” the cluster so that there are less idle resources
  - Without backfill, jobs strictly wait in line as in previous figure

Definitions: Partitioning

How to view partition information on the systems?

Max walltime partitions are named: **b1-b7**

Again note the comparatively less resources in large memory and long duration partitions!

More information with [clusterstats]

[remote $]

```
partition-stats
```

[output]

```
Node type |                    Max walltime
          |   3 hr   |  12 hr  |  24 hr  |  72 hr  |  168 hr |  672 hr |
----------|-----------------------------------------------------------------
       Number of Queued Jobs by partition Type (by node:by core)
----------|-----------------------------------------------------------------
Regular   |   54:20  |  59:174 | 238:1855|  56:1209|  79:244 |  18:1943|
Large Mem |    0:0   |   0:1   |   1:0   |   0:12  |   0:21  |   0:22  |
GPU       |    0:0   |   0:0   |   2:644 |   3:215 |   1:1   |   0:9   |
----------|-----------------------------------------------------------------
      Number of Running Jobs by partition Type (by node:by core)
----------|-----------------------------------------------------------------
Regular   |    3:71  |  33:880 |  18:760 |  37:1215|  48:472 |  27:544 |
Large Mem |    0:2   |   0:54  |   5:6   |   1:36  |   1:9   |   0:7   |
GPU       |    0:9   |   1:29  |  19:66  |  17:82  |   0:39  |   0:17  |
----------|-----------------------------------------------------------------
       Number of Idle nodes by partition Type (by node:by core)
----------|-----------------------------------------------------------------
Regular   |    2:0   |   2:0   |   2:0   |   1:0   |   1:0   |   0:0   |
Large Mem |    0:0   |   0:0   |   0:0   |   0:0   |   0:0   |   0:0   |
GPU       |   13:4   |  11:2   |   6:2   |   2:0   |   0:0   |   0:0   |
----------|-----------------------------------------------------------------
       Total Number of nodes by partition Type (by node:by core)
----------|-----------------------------------------------------------------
Regular   | 1088:646 |1088:646 |1058:626 | 763:391 | 381:180 | 106:58  |
Large Mem |   27:19  |  27:19  |  24:16  |  20:4   |   5:4   |   3:2   |
GPU       |  200:122 | 194:116 | 182:110 | 139:81  |  42:39  |  29:27  |
----------|-----------------------------------------------------------------
```

Assumptions

- Researchers often isolate themselves to very few nodes without noticing
  - Jobs with hyper specific demands may wait longer
- The easier you make it for slurm to schedule your job, the sooner it will run
  - With some notable exceptions…
  - Note the relation to the above point
- The systems have a lot more bynode resources than anything else
  - Can we take advantage of this?
- Resource waste is bad and should be optimized away
  - There **will** be an impact to your priority/fairshare
- **Development time is often a cost worth paying to maximize throughput**

Conquering: Stage 1

Starting with the low hanging fruit:

1. Submitting as much work as possible at once
    a. Slurm job dependencies if needed; see “--dependency” in sbatch manual
2. Coordinating with others in your lab/group
    a. Recall how to check your group’s fairshare
3. Accuracy of job size
    a. Do **not** ask for things you don’t need like GPUs or an extra 20 days of run time
        i. You **will** increase your billing by requesting more resources and not using them
        ii. You **will not** decrease your priority by asking for extra time, but you wait time will go up
    b. Profile your execution with interactive jobs
    c. Make sure you are isolating yourself to the maximal amount of resources available
        i. Falling into the largemem partitions is dangerous

Conquering: Stage 2

Consider just the individual job:

- 27-day run time
- 6 cores, 128G of memory
- 1000 submissions

You are waiting for an **ENTIRE** large memory node to be free for 27 days 1000 times

Conquering: Stage 2 - Refinement

Say you have arrived at the following job shape via your own profiling and research requirements:

- 5-day run time
- 6 cores, 24G of memory
- 1000 submissions

This is infinitely better, but is it good enough if a reviewer asks for... **10,000** submissions?

Conquering: Stage 2 - Review

Say you have arrived at the following job shape via your own profiling and research requirements:

- 5-day run time
- 6 cores, 24G of memory
- 1000 submissions

Okay; how bad is that really? This looks really nicely sized!

- Run time: competing against all other jobs in the **b4** (7-day) partitions
- Core and memory size: sharing the node and **finding** this shape/duration 1000 times
  - This job is only a fifth of a base memory node
- This job is **difficult** or **impossible** to backfill

Conquering: Stage 2 - Possibilities

From previous:

- 5-day run time
- 6 cores, 24G of memory
- 1000 submissions

What are our options?

- Can we cut up the duration?
- Can we package multiple together in a single node?
- Can it be distributed via MPI, etc?

Identifying the best way forward is RESEARCHER/SOFTWARE DEPENDANT

Conquering: Stage 3 - Tuning for Duration

Can we reduce execution time of a job by dividing it up into checkpointed stages?

When should we?

- If (*and only if*) there is no overhead in restarting a job
- There are natural stopping points
  - i.e. saving after 10 epoch, or saving after every hour of execution

What do we gain?

- Wider range of nodes to run on
- Better backfill opportunities
- If any part of the cluster goes down, your progress is saved!
  - This is an argument for **always** implementing checkpointing

Conquering: Stage 3 - Tuning for Duration

Where to start for checkpointing?

- Our wiki entry on it
  - A second wiki entry with a more specific example
- Container based strategies
  - Apptainer example (Singularity's new name)
- Implementation of your own serialized state
  - Can be as simple as writing files to scratch!
- BLCR
  - Old but potentially useful

There can be language and software specific solutions already out there!

Conquering: Stage 3 - Tuning for Packaging

When should we package jobs together to run on whole nodes?

- When it is **very difficult or impossible** to reduce the job size further
- “Noisy neighbours” must be eliminated

What do we gain?

- Access to **numerous** nodes
  - Federational bias towards wholenode work
- Additional backfill opportunities
  - Even more opportunities when used in conjunction with duration tuning!

Conquering: Stage 3 - Tuning for Packaging

Where to start for job packaging?

- GLOST
  - Bundle jobs together and submit; little overhead!
  - Simple syntax
- Gnu Parallel
  - More robust handling of failures and job states
- META
  - Inhouse developed tool which combines the best of GLOST and Gnu Parallel
  - Less overhead with account wide “farm” management
  - Webinar content available as well as a very robust wiki page

As previously mentioned, investigate wiki pages or reach out to support if curious!

Conquering: Stage 3 - Tuning for Distribution

What if investment is made into MPI and cores/memory are distributed across multiple nodes?

- You have become the grain of sand in the jar!

What do we gain?

- Extreme performance from backfilling
- Practically all idle resources are fair game!
- Mature software support on several platforms
- Sadly not always possible; ask us though!

Final Considerations (and Complications)

Combining any/all previous stages can lead to some amazing results!

Some things to consider if your particular workflow cannot be optimized or reshaped:

- Job dependencies
  - If you have one stage that uses a large amount of memory and then never uses it again, try investigating if it is at all possible to isolate that step to its own job
  - If different steps use different amounts of cores/memory breaking the job down into separate parts **will** see responsiveness increases
- The META package also can function as a manager
- **“bygpu”** also exists but is beyond the scope of a one hour talk. If you would like more information, my email is always open and will be posted!

Takeaways: Discussion / Questions

General principles from today:

- Read the CC wiki
- The easier you make it on the scheduler, the sooner your jobs can run
- Isolating yourself to resources in high demand and low supply hurts
- Exploiting the structure of the system by packaging your jobs, or distributing them across nodes is worth development time

Thank you for your attention!

Feel free to email me: tk11br@sharcnet.ca

Wiki: https://docs.computecanada.ca/wiki/Compute_Canada_Documentation