10 ways to improve your workflow

Number 5 will shock you!

This is not a FAQ

I’m trying to make this presentation different from our FAQ and documentation site, because those are excellent!

PLEASE read docs.computecanada.ca - it has a wealth of information.

(It’s a Wiki, and you can even make contributions! We know it’s got so much content that it’s sometimes unapproachable - we sometimes talk internally about ways to reorganize or theme the site to provide discipline-specific entrypoints)

Help us help you

Support tickets (email support@computecanada.ca) are always your first and primary way to get help!

- Tell us which cluster or resource
- Provide us with jobids, specific filenames, etc
- Paste in relevant text, not screenshots or videos (!)

Also consider our training sessions, summerschools, periodic intro seminars, youtube channel(s)

Check status.computecanada.ca

If you think something’s down, please check the status page first.

We try to announce planned outages and known problems there.

(This is likely to get renamed at some point, but is also likely to become more featureful)

Ponder the mystery of life(cycle)

Account lifecycle is intended to follow a researcher's academic progression

- Joining another project
- Changes of role (PhD -> Postdoc, etc)
- Changes of sponsor
- PI should establish policy/expectations for data and code
  - By default, file owner controls, NOT sponsor
  - Yes, even though sponsor has to pay the bills (quota, allocations)
- Think about what happens when a student leaves
- Suggestion: allow g+rX for project and nearline
  - Hopefully no one is abusing scratch for precious files
  - Leaves home if the sponsored account wants some things private

Use the right cluster

Narval is newest and has the fastest GPUs.

Cedar is biggest (and tens of thousands of other users know it).

Niagara is best for large MPI jobs (but you'll have to restart).

Graham is the "just-right" cluster - still fairly big, with a range of different resources and very friendly staff!

Use multiple clusters

Lots of code can run on any of the clusters. Even a lot of GPU code will work fine on a wide range of nodes and GPU models! You can get 3-4x more work done by using them all.

- Moving data around
- Different cpu and GPU models
- Different scheduling policies (mainly timelimit)
- Talk to us about data repositories
- Containers might provide useful insulation

Small files are the devil's playground

Think about your files - how many you're generating, whether they're of sensible size.

Small files are a problem because there is per-file overhead, not just bandwidth.

- Consider using tar to bundle files
- Consider using the SQL DBaaS on each cluster
- Consider using smarter file formats (sqlite, HDF)
- How about squashfs?
- Talk to us about repositories of well-known datasets

Jobs should last at least 15 minutes.

- Each job requires some setup by the system.
- Everyone likes a big cluster, but if everyone brings a lot of jobs, the cluster's scheduler will be slow and overloaded
- Job arrays are good, but they're just a submission syntax (job array elements should last > 15 minutes)
- If you have short jobs, bundle them together

Try to use a whole node

There are a lot of advantages to using a whole node

- The scheduler is deliberately biased toward whole-node usage.
- No noisy neighbors
- If you max out one kind of resource on a node, you might as well take them all. So if the node has 4 GPUs, you might as well use all memory+CPUs
- Systems like Meta and GNU Parallel let you start lots of small/short jobs

Don’t use whole nodes unless you can keep a whole node busy.

Don’t use salloc all the time

Both salloc and sbatch do the same thing: allocate dedicated resources for your job. The difference is you wait for salloc, but sbatch is fire-and-forget. (Depending on the cluster and other users, salloc may wait less because it might be satisfied from a small number of nodes set aside for “interactive” use.)

If you’re doing a lot of interactive or GUI stuff, you should probably be using a VDI node or Jupyterhub. JH isn’t just for Python!

Test your code for performance

Whenever you start using some new tool, please set aside some time to do performance testing.

- Fixed input/config that will run long enough (maybe half an hour, probably not more than a couple hours)
- Maybe try it on several clusters - there's a fair amount of differences in CPU, storage, interconnect and GPU performance
- Parallel scaling: single job that allocates a whole node and runs on 1,2,4,8, 16,32 cpus.
- Don't do multiple nodes till you understand single-node scaling. Do 2 nodes, then 4, etc

Pay attention to memory

Don’t ask for too much

- High memory-per-core will be forced onto “high memory” nodes (fewer)
- Asking for less memory will reduce waiting time
- But ask for enough - difficult to predict, so do it experimentally
- Once you get something to run, check how much memory it actually used
- Modest amounts of “extra” memory will speed up IO (but nothing else)
- Java is a stupid pig
- If your memory use is “spikey”, consider splitting it into multiple jobs
- Newer Slurm does support heterogenous jobs

Use the right filesystem

Home for source mainly, maybe tools

Project for important data, configs, maybe tools

Scratch for inter-job temporary files

Localscratch for intra-job temporary files (per-node, removed at job end!)

Nearline for large files (or tars) likely to remain unaccessed for weeks (tape)

Consider using a sharing group and ACLs

A Unix user is a member of an arbitrary number of groups

- We use this to encode access to a sponsor’s allocation/quota
- Sponsored account can access many projects this way
- We can make groups solely for the purpose of providing access
  - They don’t introduce anything new for Slurm or quotas - just permissions
  - CCDB provides a self-serve management interface (once created)
  - You can choose the name, mostly
  - Put arbitrary users together
  - ACLs provide arbitrary access via any of a user’s groups

How to move files around

Globus. A lot of people love it - it's fairly GUI/app-oriented.

scp/sftp/rsync/sshfs are all viable and in some ways much nicer

Beware: our storage quotas (especially on project and nearline) depend on you maintaining relevant group.

Network issues

COMPUTE NODES AREN'T ON THE INTERNET

There are ways to get to the internet from a job

- For license access, we make specific exceptions in firewalls, provide tunnels, even run the license manager for you
- For database-driven workflows, you might like the DBaaS (local SQL server)
- We provide proxies in some places, generic NAT in others
- For production loads that touch the internet, perhaps stay in the cloud
- We do block internet access to login nodes when it looks like the incoming IP is trying to brute-force credentials. Open a ticket and include your IP.

Consider using containers

You may be familiar with Docker - it’s an older but widely-used mechanism for bundling processes into private file namespaces.

Docker can’t be used in a shared environment, but we have Apptainer (formerly known as Singularity). We may add Podman.

Why? Reproducibility, but also portability.

VMs or batch/cluster?

- VMs are exactly right for anything like a webserver
  - “Persistent” nodes are very CPU-overcommitted
- Clusters are right for heavy compute loads: big/fast storage, MPI interconnect, GPUs, and bursting
  - Compute nodes are never overcommitted (except network, localscratch)
- VMs might also be right for steady, production workloads that are still compute-oriented.
- Special cases are interesting (for instance, a workload that needs significant resources for 8 hours every Sunday).
- Talk to us about it!