HPC Training

version

HPCO

January 20, 2026

Contents

GPSC Science Documentation

Using our HPC

User Accounts

In order to access HPC resources, Linux accounts are assigned to each user.

Science account

A user account for accessing GPSC HPC systems on the science network. The username follows the naming convention `aaa123`, where the characters are derived from the client's initials. Each user belongs to a primary group that is determined by the user's department and group``<department>_<group>``.

Please refer to our portal page for more details here.

The form to be filled to get a user account can be found here.

Collaborator account

GPSC-C, a collaboration cluster facing the internet, allows external collaborators to have an account. It is also available for actual GPSC users.

The username is identical to your science account username; however, these two accounts are distinct and therefore require individual authorizations.

User representative

A user representative (userrep) is appointed to each partner (i.e. department) Linux group. In general, userreps represent the interests of users in meetings with SSC support staff, and have elevated privileges for managing their group's HPC resource allocations. They are your first point of contact when it comes to questions, issues and changes.

A user representative's duties and responsibilities include:

- managing account requests
- supporting the user setup and configuration (profile, environment, group/project specific stuff)
- ensuring quality communication (clarity, completeness) in service requests and issues
- managing storage resource usage: quotas, allocations, permissions/acls
- managing compute resource usage: priorities, allocations, consumption
- notifying users about system changes, system times, etc.
- participating in regular userrep meetings to discuss technical issues and needs with SSC staff and other userreps

Getting Started

Interactive access

To log in to the GPSC with a science account credential, an ssh session is needed. Use a terminal, Putty, Powershell, VSCode or any applications that support an ssh connections. Examples:

```
$ ssh <science_account>@inter-<department>-<os-version>.science.gc.ca
```

```
$ # example
$ ssh anl987@inter-nrcan-ubuntu2204.science.gc.ca
```

To login to the GPSCC with your collab account credentials:

```
$ ssh <science_account>@inter-c-<department>-<os-version>.collab.science.gc.ca

$ # example
$ ssh anl987@inter-c-nrcan-ubuntu2204.collab.science.gc.ca
```

Having issue reaching the login nodes?

- Make sure you are connected to a Goverment of Canada network (VPN) if GPSC is the target.
- Sometimes, buildings or VPN's have their own firewalls. Check with your colleagues, can they connect? Port 22 needs to be open.
- Try reaching *inter-<department>-lp.science.gc.ca* or *inter-<department>-lp-gccloud.science.gc.ca*. These are referred to as "landing pads", which are nodes located at the edge of the network. They are used as "proxy" to hop into the proper login node mentionned above.

Environment configuration

Upon connecting through an ssh session, you'll land inside a container where your personal folders and departement storages are available. That session can be configured the way you see fit: shortcuts, colors, personal organizations, routines, etc. This is called a shell environment.

Here's how to set it up on GPSC or GPSCC.

.profile vs .bashrc

Bash sessions are configured by special scripts called startup files. There are two standard ones that allow users to define personal settings: `~/.profile` and `~/.bashrc`. It is important to note that these files are loaded into bash shells under different circumstances, and that they hold different types of settings.

When bash is invoked as a login shell, like when logging into an interactive cluster, the `~/.profile` is automatically loaded into the shell environment. This file contains personal bash-nonspecific environment variable settings.

In comparison, the `~/.bashrc` is only loaded into non-login shells; that is, for bash commands executed locally or remotely over ssh (e.g. `ssh <hostname> date`). This startup file is intended for bash-specific settings as well as command aliases and shell functions.

.profile setup

source `~/.bashrc`

As previously mentioned, the `~/.bashrc` is not sourced when an interactive login session has been created. Typically these bash-specific settings are wanted in login sessions, but they need to be loaded in the `~/.profile`:

```
[ -f ~/.bashrc ] && . ~/.bashrc
```

Load pre-configured profiles (ssm)

In order to load packages and environment settings that are particular to the partner and group you belong to, several profiles need to be loaded:

```
export ORDENV_SITE_PROFILE=
export ORDENV_COMM_PROFILE=
export ORDENV_GROUP_PROFILE=
. /fs/ssm/main/env/ordenv-boot-<datestamp>.sh
```

`ordenv` is a tool that loads environment settings and tools into your bash session. The following environment variables are used by `ordenv` to load settings tailored to each user:

- `ORDENV_SITE_PROFILE`: environment settings applicable to all users
- `ORDENV_COMM_PROFILE`: community-specific settings
- `ORDENV_GROUP_PROFILE`: group-specific settings

These profiles are managed by an administrator. In order to get the latest values, ask your user representative.

Good practices

Redirect output

It's almost always a bad idea to include commands that output to stdout or stderr. It is best to redirect the output somewhere else, which leaves stdout clean and easy to work with when using command pipelines.

Instead, send the stdout and stderr to `/dev/null`.

```
aprogram > /dev/null 2>&1
# or
aprogram > login_session.txt 2>&1
```

Avoid lengthy commands

Another potential problem is if `.bashrc` runs something that takes a non-trivial amount of time to complete. Because this file is loaded everytime a command is run, it is best to ensure the `.bashrc` loads quickly in order to have a smoother experience.

ssh configuration

Several ssh configurations are required for submitting jobs and accessing some services.

Passwordless authentication

As part of the execution of some commands, some tools run commands on remote servers using ssh connections. These commands cannot be disrupted by credential prompts, so passwordless ssh login needs to be set up.

1. Create `.ssh` directory in `$HOME` directory:

```
$ mkdir -p ~/.ssh
```

1. Disable all other users' access:

```
$ chmod -R 700 ~/.ssh
```

1. Add your public key to your authorized keys file:

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Settings

Add the recommended syntax to your `~/.ssh/config` file:

```
Host *
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
```

These settings are required to disable the prompt for adding host keys to `~/.ssh/known_hosts` that would otherwise disrupt some scripts or commands that execute commands over SSH.

Checking Storage Limits

To check storage space usages and limits, the `sqm` tool can be used. For example, here's how to list the storage limits in your home:

```
$ sqm list -u -d .
Quota Path: /fs/vnas_Hnrcan/amspm
Last Updated: 2021-03-02 19:15:10 UTC
Update Frequency: 5 minutes
------------------------------------------------------------
User Name  Used Space (GB)  Soft Limit (GB)  Hard Limit (GB)
```

```
sdum001                0.0                -             10.0
--------------------------------------------------------------
```

Launching a Simple Slurm Job

Once your environment is properly setup, submitting jobs is simple. Some compute clusters manage jobs with the Slurm job scheduling system. Here's how to prepare, submit, and monitor a job using native Slurm tools.

Prepare the jobfile

jobfile that defines directives for the scheduler, including compute resources requests and the commands to run.

Here is a simple jobfile called `sort-numbers.job`:

```
#!/bin/bash
#
#SBATCH --job-name=sort-numbers
#SBATCH --output=sort-numbers.%j.out     # stdout file (sort-numbers.<jobid>.out)
#SBATCH --error=sort-numbers.%j.err      # stderr file (sort-numbers.<jobid>.err)
#SBATCH --account=nrcan_amspm  # account tag REQUIRED
#SBATCH --partition=standard   # partition (queue)
#SBATCH --nodes=1              # number of nodes
#SBATCH --cpus-per-task=1      # number of cores
#SBATCH --mem-per-cpu 200M     # memory per core
#SBATCH --time 00:00:60        # wallclock time (HH:MM:SS) REQUIRED
## optional, default to interactive containers version
#SBATCH --comment="image=nrcan/nrcan_all_default_ubuntu-24.04-amd64_latest"

# commands to run
srun hostname
srun df

for i in {1..6}; do
srun echo $RANDOM >> /tmp/random-numbers.txt
done
srun sort -n /tmp/random-numbers.txt

srun ls -la /tmp
```

Submit

Submit the jobfile to the scheduler:

```
$ sbatch sort-numbers.job
Submitted batch job 67140430
```

Once submitted, the job scheduler places the job in the specified queue based on a calculated priority value.

The job will then be ran on a compute node (or more). Since these compute nodes have access to the same filesystems available on the interactive container, the job will be able to read and write files.

Monitor

To see the current state of the job, list your submitted jobs:

```
$ squeue -u sdum001

   JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
67140430  standard sort-num  sdum001  R       0:00      1 ib10-fx02e42-be01
```

sacct also works, once the job id is known, more details can be obtained with:

```
$ scontrol show job <jobid>
```

More advanced Slurm submission

Multi steps

```
#!/bin/bash
# Multi-step job example using srun within the same allocation
# This script requests a larger allocation up-front and runs several sequential srun steps
# with different per-step resource usage. The first two srun calls are done at the same time
# and step 3 waits for them to finish
# How to use:
#  - edit the overall SBATCH resource request to cover the maximum resources needed
#  - adjust the srun calls for per-step allocations
#  - submit with: sbatch multistep_example.sbatch

#SBATCH --job-name=multistep_example
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=32
#SBATCH --time=02:00:00
#SBATCH --partition=large
#SBATCH --account=nrcan_geobase
#SBATCH --output=multistep_example.%j.out

# Compile intel
# Version might change please check with portal.science.gc.ca
# . ssmuse-sh -x main/opt/intelcomp/inteloneapi-2023.2.0/

# Compile and run time (OpenMPI)
# Version might change please check with portal.science.gc.ca
. ssmuse-sh -x main/opt/openmpi/openmpi-4.1.5rc2--hpcx-2.15-mofed-5.x--gcc/

# Step 1: large distributed MPI work across many tasks
echo "Step 1: MPI work starting on $(hostname)"
srun -N2 -n60 ./stage_one_mpi_work &
echo "Step 1 complete"

# Step 2: a smaller single-node, multi-threaded job (runs after step 1)
echo "Step 2: single-node post-processing"
# Limit this step to 1 node and use the remaining 4 CPUs available
srun -N1 --ntasks=1 --cpus-per-task=4 ./process_single_node &
echo "Step 2 complete"

# wait for background (&) processes to be finishes
wait

# Step 3: cleanup step that uses a single task
echo "Step 3: final cleanup"
srun -N1 --ntasks=1 --cpus-per-task=1 ./cleanup_task

echo "All steps finished"

# Alternative: If you prefer separate Slurm jobs, you can submit step scripts and chain them
# using sbatch --dependency=afterok:<jobid>. Example (from shell):
# job1=$(sbatch --parsable step1.sbatch)
# job2=$(sbatch --parsable --dependency=afterok:${job1} step2.sbatch)
# This will ensure step2 only runs after step1 completes successfully.
```

Job arrays

```
#!/bin/bash
# Array job example
# How to use:
#  - set the array range in the SBATCH header (--array)
#  - use $SLURM_ARRAY_TASK_ID inside the script to select input per-task
#  - submit with: sbatch array_example.sbatch

#SBATCH --job-name=array_example
#SBATCH --array=1-100%10           # tasks 1..100 with max 10 concurrent
#SBATCH --time=00:10:00
#SBATCH --account=<your account>
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --partition=standard
#SBATCH --output=array_example.out

# each file output if needed
# #SBATCH --output=array_example.%A_%a.out

# Example: pick an input file based on the array index
mkdir array_folder

INPUT_DIR="./array_folder"
INPUT_FILE="$INPUT_DIR/input_${SLURM_ARRAY_TASK_ID}.dat"

echo "Running array task ${SLURM_ARRAY_TASK_ID} on $(hostname)"

# create a file for this array index
# usually you would read from them in a map/reduce fashion
echo "content ${SLURM_ARRAY_TASK_ID}" > "$INPUT_FILE"

# Notes:
# - Use %A for master job id and %a for array index in filenames.
# - %10 in --array limits concurrency (good for throttling shared resources).
```

MPI Jobs

```
#!/bin/bash
# MPI job example
# How to use:
#  - edit the job parameters below (nodes, ntasks-per-node, time, partition)
#  - load the required MPI package
#  - submit with: sbatch mpi_example.sbatch

#SBATCH --job-name=mpi_example
#SBATCH --nodes=2                    # number of nodes
#SBATCH --ntasks-per-node=32         # tasks (ranks) per node
#SBATCH --time=01:00:00               # HH:MM:SS
#SBATCH --account=ssc_support         # Your account, generally the same as your group
#SBATCH --partition=standard          # partition/queue name
#SBATCH --output=mpi_example.%j.out
#SBATCH --error=mpi_example.%j.err

# Load OpenMPI
# Version might change, check with your user rep
. ssmuse-sh -x main/opt/openmpi/openmpi-4.1.5rc2--hpcx-2.15-mofed-5.x--gcc/

# Recommended: use srun for MPI launches (it integrates with Slurm's PMI)
# -n (total tasks) is computed automatically by srun when used within allocation
# If you prefer mpirun, ensure it is the site-recommended launcher.

# Run the MPI executable (replace with your binary)
# srun will launch 2 nodes * 32 tasks-per-node = 64 ranks

# Look up mpi type
srun --mpi=list

srun ./mpihello #srun will dispatch the MPI job itself

# Same thing, using mpirun
# Bind to core: when the number of processes is <= 2
# Bind to package (socket): when the number of processes is > 2 ## physical processor socket
# Bind to none: when oversubscribed

# We recommened binding to cores, some issues can arise
# when a single node is being shared without this option
mpirun -n 2 --bind-to core ./mpihello

# Notes:
# - If your application uses environment variables, export them before srun.
# - To request specific cores per task use --cpus-per-task, e.g. srun --cpus-per-task=2
```

OpenMP

```
#!/bin/bash
# OpenMP job example
# How to use:
#  - set OMP_NUM_THREADS to the number of CPU cores per task you want
#  - request cpus-per-task in SBATCH so Slurm reserves CPU cores for your threads
#  - submit with: sbatch openmp_example.sbatch

#SBATCH --job-name=openmp_example
#SBATCH --nodes=1
#SBATCH --ntasks=1                    # single process (multi-threaded)
#SBATCH --cpus-per-task=16             # number of OpenMP threads
#SBATCH --account=ssc_support
#SBATCH --time=00:30:00
#SBATCH --partition=standard
#SBATCH --output=openmp_example.%j.out

export OMP_NUM_THREADS=16

# Run the OpenMP-enabled binary
./omphello

# Tips:
# - Match OMP_NUM_THREADS to --cpus-per-task.
# - For hybrid MPI+OpenMP jobs, request ntasks >1 and set --cpus-per-task accordingly.)
```

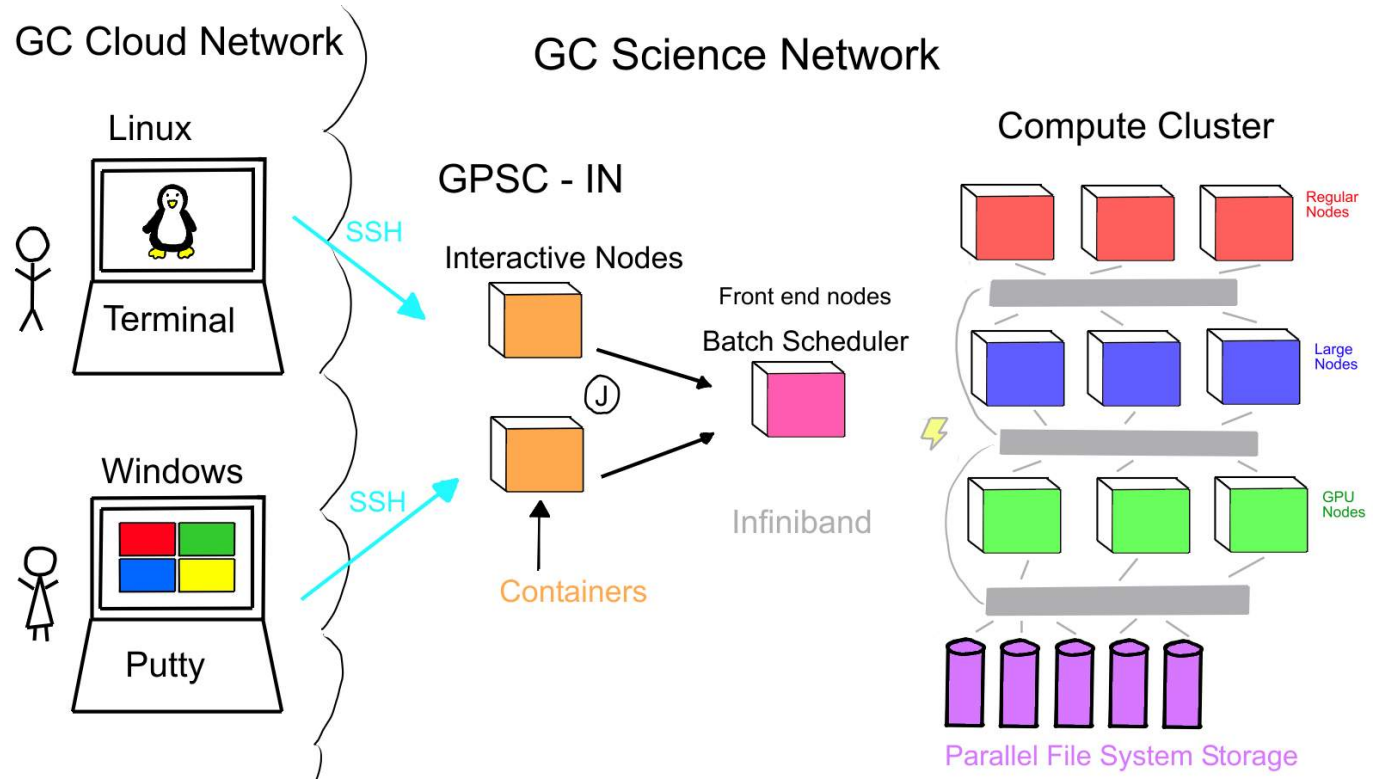
System Overview

System Overview

Introduction

GPSC stands for General Purpose Science Cluster; it’s a High Performance Computing solution delivered by the High Performance Computing Team of Shared Services Canada.

It has many components. There’s mainly the interactive cluster, the compute cluster, the high performance storage, the network interconnect and the use of Linux containers.



What's a scheduler?

"A Batch-Scheduler is a program running on a cluster that decides who can use which machines at what time."

It is there to coordinate between all the jobs, prioritize and decide where jobs are run and how they are dispatched. A scheduler is coupled with the idea of a cluster which is a set of machines meshed together. GPSC has many clusters for different needs and partners.

Interactive Nodes

Interactive nodes try to replicate the environment that will be available on the compute nodes (filesystems for storage, environment variables, available libraries, etc.). They are based on the same container image that will be used when work is sent to the scheduler.

Interactive nodes are to be used for low resources demanding work like creating job files, downloading, testing accesses to different storages and organizing files and folders. They should not be used to execute real work as they do not have a lot of resources.

These nodes would also be used for data transfer between storage spaces.

Visualization Nodes

Similar to the interactive nodes in terms of purpose, visualisation nodes or "vis" are nodes have GPUs for supporting graphical applications.

Front-end Nodes

These nodes are used by the scheduler to coordinate the resources and the jobs.

Compute Nodes

These come in different sizes and shapes, and they make up most of each cluster. Some have more memory, and others are built for GPU work, for example. Compute nodes should only be used for computing not for downloading or syncing data.

xfer Nodes

Used for a specific purpose when the interactive nodes are not sufficient for data transfer. Partners can ask for xfer nodes to be installed and use them (mid-term or long-term) through the scheduler job submissions.

Hardware

Please refer to our portal page.

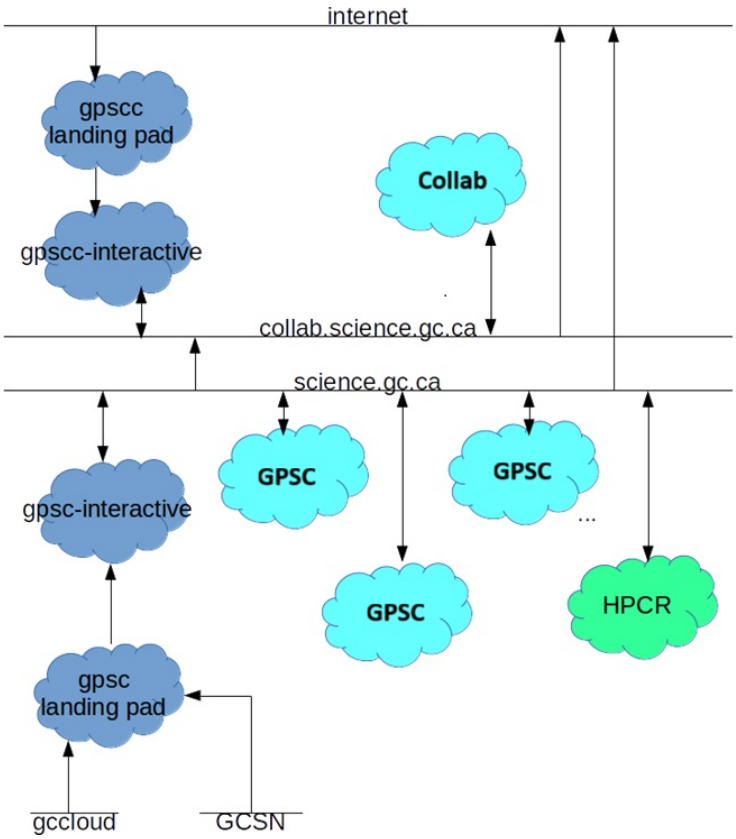
Network

Science

The science network (*.science.gc.ca) is where the majority of the services and contact points for schedulers are. There are two networks from which users can access the science network: GCSN (Canarie) and GC Cloud (the government corporate network).

It is important to understand that the network used to access the science network depends on the building users work in, or the VPN server installed on their computer. Some users only have access to science through either GC Cloud or GCSN, whereas some users will have access through both networks. There is no way for SSC to know where users come from since we are managing the endpoint of that destination.

Here's a simple summary of the network of science.gc.ca:



Only a subset of GPSC clusters are presented here.

Note on network performance

GCSN is composed of an "internal" ring wired with a 20Go bandwidth connection. The external link is 10Go. The partner's VPN support should be able to provide where the connection is set up.

In comparison, GC Cloud is 1Go.

Based on that, it is recommended to use GCSN.

Where are you and what can you see?

To figure out through which route can be used to reach science.gc.ca, there are some easy tests that can be done:

```
# If this address is reachable, a route exist via GCSN
ssh inter-<department>-<os>.science.gc.ca

# If this does not work, try
ssh inter-<department>-lp.science.gc.ca

# If the previous ones were unreachable but this one is, a route exist via GC Cloud
ssh inter-<department>-lp-gccloud.science.gc.ca
```

A more advanced test is to figure out which IP realm is reachable. Know that:

- Hosts that have IPs starting with 142.98 as a destination are on GCSN
- Hosts that starts with 205.195 or 205.188 are for GC Cloud.

```
# Linux
host hpc-stats.science.gc.ca
```

If a service is supposed to be accessible (“gitlab.science.gc.ca”, “hpc-stats.science.gc.ca”, etc.), a test like this would give a good starting point to troubleshoot the situation.

At any point the VPN service provider would be able to answer questions related to accesses. Take note that access through GC Cloud always goes through a firewall that will NAT (Network Address Translation) to science.gc.ca whereas GCSN will simply route to it.

Do not assume all network configurations are the same for all the groups belonging to a given partner. Those can vary greatly.

On science

Once the science.gc.ca network is entered via the command line or a visualisation tool, the hosts inside know each other, and the VPN points are far behind. From science.gc.ca, there are many accesses to both internal (interactive nodes, calls to HPC APIs) and external (Compute Canada is reachable through GCSN) services depending on the needs.

Only connections from the partner’s network are allowed to the science.gc.ca network.

Collab

The main difference with the collab.science.gc.ca network is that it is accessible directly from the Internet (and from the science.gc.ca domain). It gives the opportunity for partners to share their resources with external parties.

Connections from collab.science.gc.ca are not allowed.

```
$ ssh inter-c-<department>-<os>.collab.science.gc.ca
```

Storage

Home ($HOME)

This is the space where users end up when logging into science: their home directory. It is used to store lightweight files like configuration or job files. This space is accessible by jobs that run on the back-end nodes. Also, frequent back-ups are made for this particular storage so that past versions of lost or corrupted files can be recovered.

“Project/work/scratch” space

These storage spaces are managed by the GPFS filesystem (“General Parallel Filesystem”), which is designed to handle large files efficiently by reading and writing to storage devices in parallel. GPFS is coherent, meaning that it has a protocol that handles parallel access to a single storage location. Another important characteristic of this filesystem is that it is divided into multiple layers where the top one is composed of SSD which provides fast input/output speeds.

This storage should be used for most of the work.

HPNLS

This is the archival system which uses a storage hierarchy (multiple layers like GPFS) but with layers composed of SSDs and tapes to for archiving data for long periods of time. This is not to be confused with “back-ups”: archives are to be seen as big files that do not need to be retrieved quickly in case of an emergency.

A tool developed by our HPC team called `hpcarchive` is to be used for a better optimization of that storage.

tmpfs

Also known as `/tmp`, tmpfs is a “storage” built on memory (i.e. RAM). This type of storage intended to be used to store data on a short-term basis that is quickly accessed by running jobs. Data stored here is not persisted, so once the job is finished, the data created under tmpfs will be deleted. This storage is good for quick IO transactions.

Quotas

Most of the storages listed above have quotas associated with them. Users need to be aware of those limits before writing their data.

A tool developed by our HPC team called `sqm` provides information associated to all storages.

Container Images

Users never directly engage with the computing hardware; instead, all interactive sessions and batch jobs are run inside containers.

What is a container?

A container image is an executable program compiled from an operating system along with selected system packages, tools and configurations. A container is a running instance of an image that runs on top of the underlying infrastructure.

Containerization provides several benefits such as:

- access to a variety computing environments
- resource and process isolation between containers (and partners)
- customization of system packages and software applications

Docker

Not all clusters use Docker as a container technology. At the moment only Slurm clusters are doing so.

Since Docker is dependent on *dockerd* which runs as root, *docker* as a command to spin up containers is not available on a shared environment.

We provide a Kubernetes cluster if there is a need for this kind of resource.

Apptainer

> **Note**
>
> This is the recommended way to manage your team dependencies.

Apptainer is the container flavor for HPC. It enables user to bundle their necessary environment and expose it throughout the container as themselves, thus having their privileges. It’s portable and also adapts well with parallel filesystems and GPUs.

SSC offers the opportunity for partners to create their own Apptainer images. These can be imported and executed on clusters as users see fit.

Note that SSC does not manage Apptainer containers and that users are responsible for their maintenance.

Partner images

Multiple container images are made specifically for each partner. These images come installed with a set of system packages and software that are tailored to each partner's needs. Each partner image is classified by Linux platform and version (e.g. Ubuntu24.04 and CentOS9) as well as cpu architecture.

A partner image must be selected when submitting a batch job.

Creation of new images

New versions of existing partner images are created when a new long-term support (LTS) version of a platform is available. Images of different Linux platforms can be created if requested.

Interactive containers

Interactive containers are persistently running partner containers that are intended for interactive use with the clusters. When logging into the interactive nodes, users are automatically redirected to a partner interactive container. As a result, users have access to a computing environment that is configured with their software and package requirements.

Interactive container names follow this structure: `inter-<department>-<os>.science.gc.ca`

Container design

Filesystem access

Filesystem access is separated by partner: filesystems assigned to a partner are only reachable from the containers of the same partner.

Restrictions

Users cannot login to another partner's interactive container.

Visualisation

Visualisation nodes are available for running a Linux desktop environment or using graphical applications.

These nodes should only be used for graphical purposes since they have the necessary GPUs. GPUs are located close to the data in order to do remote visualisation (over WAN) in a more efficient manner). Use other nodes (interactive nodes or compute nodes) when they are a better fit.

Desktop Environment

Thinlinc is a remote desktop tool used by users outside of the science network to access a fully operational Linux Desktop Environment on the science network from their workstation.

Interactive access

For complete information go to our portal page.

Configure ssh

Enable graphical forwarding by adding this setting to the `~/.ssh/config` file on your workstation:

```
user@workstation:~$ cat ~/.ssh/config
Host \*.science.gc.ca
```

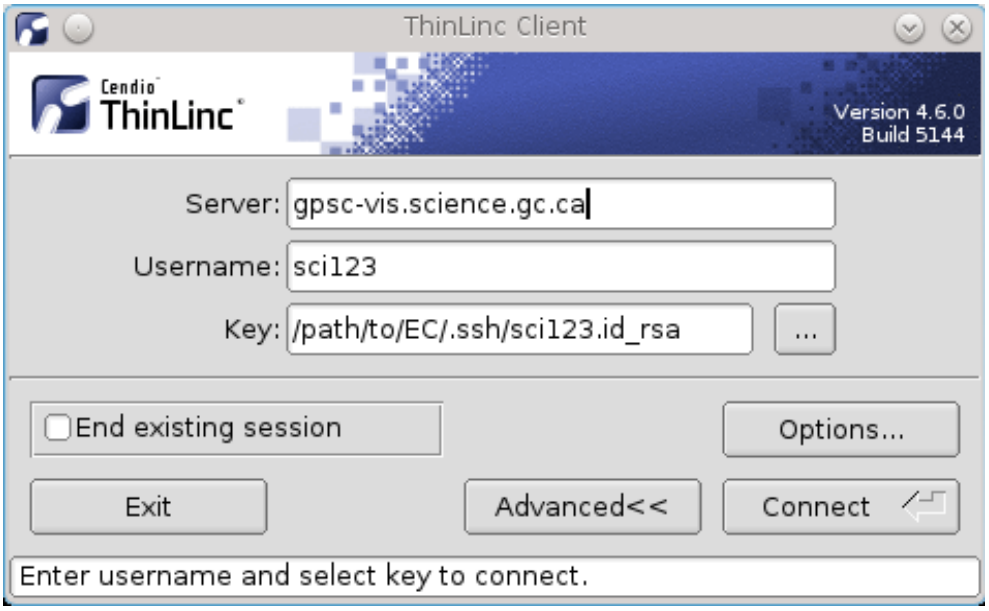
```
ForwardX11 yes
ForwardX11Trusted yes
```

Login via Thinlinc

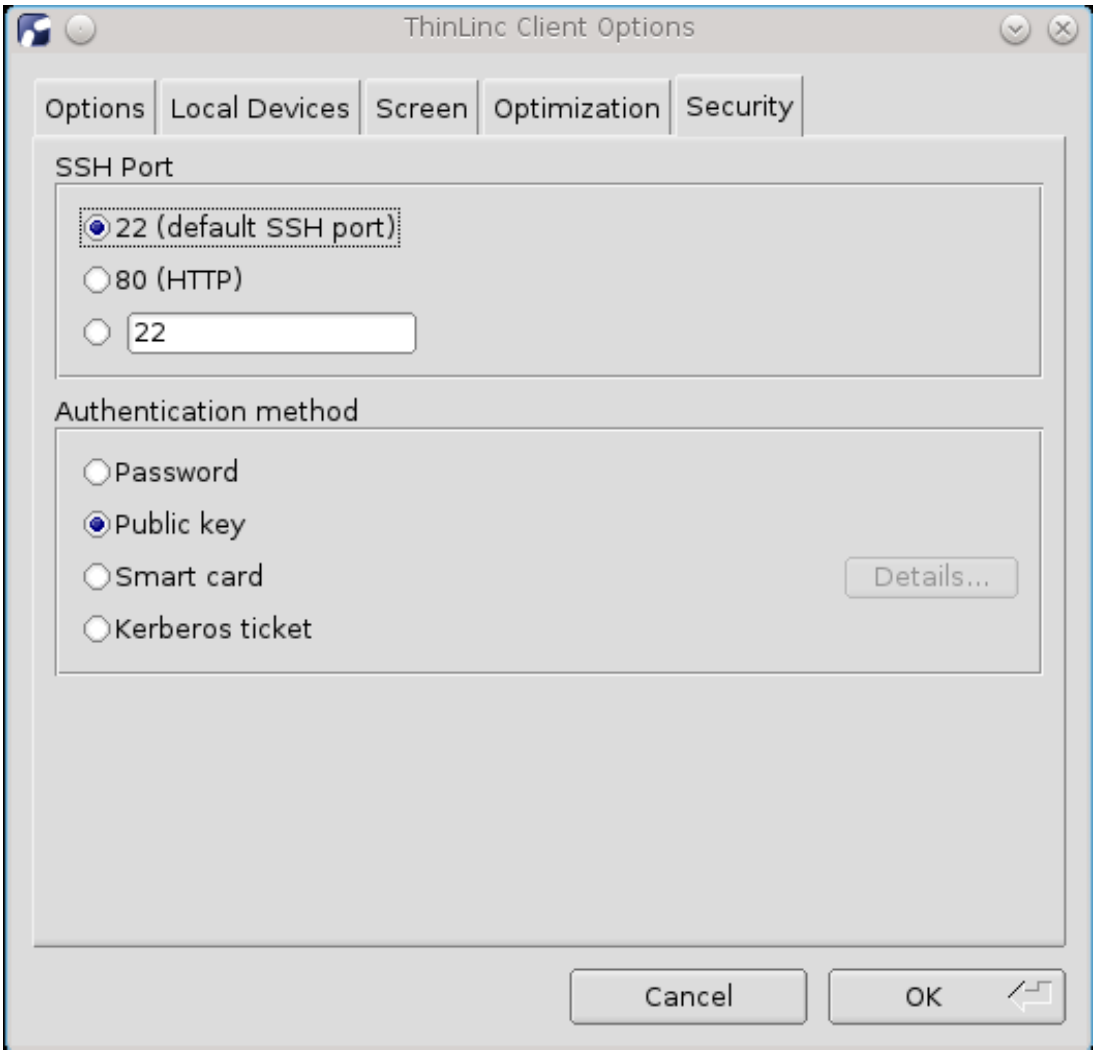
Thinlinc is free for Linux and Windows.

To login to gpsc-vis through Thinlinc, the user's **private** ssh key has to be imported from science (using *scp* for example) to the local machine.

Then specify the server and the user's name like so:



Changes to the connection method is needed in order to use the ssh key:



Login via a navigator

While connected on a VPN that has access to the science network, go to: vis.science.gc.ca.

GPSCC for Collaboration

The GPSCC, or GPSC-C, is a Linux High Performance Computing system accessible from outside of the Government of Canada's internal network.

Purpose

This cluster is made available for non-partner users to collaborate on HPC projects with partners.

Compared to GPSC

Systems

Like the GPSC, the GPSCC has an interactive cluster for interactive access, and a compute cluster for running batch jobs.

Network

GPSC is internal to the Goverment of Canada network while GPSC-C is not. GPSC refers to the network domain ending with `science.gc.ca`. The GPSCC resides in the collab.science network which refers to the externally accessible network with the `collab.science.gc.ca` domain.

Storage

The filesystems accessible from the GPSC are not accessible from the GPSCC. The underlying storage devices mounted on each cluster are unique to each cluster. For instance, the `/home` directories on both clusters are not the same.

Environment configuration

Identical to science network configuration.

Getting Support

Support requests are submitted by email, and they are handled by members of SSC. This is a guide for requesting help with technical issues, submitting service requests, and asking questions about HPC services and systems.

Support Representative

A support representative is a member of the HPC Optimization team who is responsible for overseeing the HPC needs of a department. Your support representative provides trainings and technical help, and assesses your service requests.

Submitting a Service Request

To request help with a technical issue or to submit a service request, write an email following these guidelines:

1. Address the recipient of the email to your department helpdesk (who in turn will submit it to the SSC service desk).
2. Send a carbon copy (CC) to ssc.hpcoptimizationservice-serviceoptimisationchp.spc@canada.ca.
3. In the body of the email, add:

```
Please assign to the SSC/HPCO group with Owner Group DC000134.
Please ensure that any/all attachments are copied.
```

4. For technical issues, describe the issue and include details such as:

```
username:
hostname:
command:
error output:
jobid:
jobfile path:
```

5. For service requests, describe the request and include details which may include:

```
software:
version:
platform:
installation requirements:
```

Your support representative, or another member of HPCO will follow-up with you.

Asking a Question

Submit questions about HPC systems and services to the HPCO Service Inbox email (ssc.hpcoptimizationservice-serviceoptimisationchp.spc@canada.ca).

It’s good measure to CC your support representative as well.

Stay informed

Often the support representive will send information through e-mails concerning upcoming changes.

Environment Management

Apptainer

Apptainer (formerly Singularity) is a container platform designed for High-Performance Computing (HPC) environments. It allows users to package applications, dependencies, and configurations into a single, portable file called a **container**. This ensures consistency and reproducibility across different computing environments, including shared HPC clusters.

Key Features:

- **Portability**: Containers can be moved and executed across different systems without modification.
- **Reproducibility**: Ensures that software runs the same way every time, regardless of the host environment.
- **Security**: Does not require root access, making it suitable for multi-user HPC systems.
- **Compatibility**: Supports integration with MPI, GPU computing, and other HPC tools.

Building Containers

Containers are created using a **definition file**, which specifies the software, libraries, and configurations required. This file is used to build the container image.

```
Bootstrap: docker
From: ubuntu:20.04

%post
    apt-get -y update
    apt-get -y install cowsay lolcat

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    date | cowsay | lolcat

apptainer build lolcow.sif lolcow.def
```

Running Containers

Once built, containers can be executed on any system with Apptainer installed. The container runs in an isolated environment, ensuring that the host system remains unchanged.

```
apptainer exec lolcow.sif
# or interactive
apptainer shell lolcow.sif
```

Sharing Containers

Containers (the *.sif* image) can be shared with collaborators, allowing them to run the same software stack without needing to install or configure anything.

Example: Running RStudio in Apptainer

Step 1: Pull a Pre-Built Container

Apptainer can download pre-built containers from repositories like the SyLabs Cloud Library. For example, to pull an RStudio container:

```
apptainer pull rstudio.sif library://rocker/verse:latest
```

- `rstudio.sif`: The container file.
- `library://rocker/verse:latest`: The source of the pre-built RStudio container.

Step 2: Run the Container

Execute the container to launch RStudio:

```
apptainer run --app rstudio rstudio.sif
```

- `--app rstudio`: Specifies the application to run within the container.
- The container will start a local web server, and you can access RStudio via the provided URL (e.g., `http://localhost:8787`).

Example: Running QGIS

To run QGIS for geospatial analysis:

```
apptainer pull qgis.sif library://gisenv/qgis:latest
apptainer run qgis.sif
```

- This will launch a QGIS desktop environment within the container.

Example: Running a Small Language Model

To run a small language model (LLM) using a Hugging Face container:

```
apptainer pull tiny-llm.sif docker://ghcr.io/huggingface/text-generation-inference:1.0.3
apptainer run tiny-llm.sif --model-id bigscience/bloom-560m
```

- This pulls and runs a container with a pre-configured LLM.

Why Use Apptainer?

- **Consistency**: Eliminates issues related to software dependencies and version conflicts.
- **Portability**: Containers can be executed on any system with Apptainer, including HPC clusters.
- **Ease of Use**: Simplifies the process of setting up and sharing complex software environments.

Getting Started

1. **Install Apptainer**: Follow the instructions at apptainer.org.
2. **Pull a Container**: Use `apptainer pull` to download a pre-built container.
3. **Run the Container**: Use `apptainer run` to execute the container.

Reference

For further details, refer to the official Apptainer documentation.

SSM

SSM is software for packaging and managing software apart from the native packaging system and outside of the standard system areas. As such, SSM is useful for regular users (no administrator privileges required) who don't have access to system areas. It is also useful to administrators that need to manage locally developed or publicly available software outside of the system area.

SSM is provided on all science nodes and is maintain by the SSC HPC teams.

Quick start

To start and load a package from SSM:

1. Find which domain contains the wanted package
2. Load the package

Most of the time SSM will take advantage of a filesystem mounted on all nodes to reach “SSM domains.” In this case it’s most likely `/fs/ssm`. Under this filesystem, three main directories would be worth looking at:

- `/fs/ssm/main`
- `/fs/ssm/[partner]`, replace [partner] with partner abbreviation (ie.: dfo, nrc, nrcan, etc.)
- `/fs/ssm/comm/[partner]`, replace partner by the needed (ie.: dfo, nrc, nrcan, etc.)

Then load the domain with:

```
$ . ssmuse-sh -x [the domain]  # See the dot (.).
# Example
$ . ssmuse-sh -x hpci/opt/hpcarchive-latest
```

Domains can contain one or multiple packages. Once `ssmuse-sh -x` is sourced, the package is loaded, the $PATH is modified and the environment variables associated with it are setup. The package(s) associated with the domain can now be used.

The `.` in `. ssmuse-sh -x [the domain]` is shorthand for the command `source` (source only exists in bash shells; in other shells it’s just .).

See below for package installations.

Overview

Definitions

- **domain**: specially organized directories to contain packages and domain-specific directories and files for domain management
- **package**: a binary package file or an installed instance of it
- **binary package file**: tar-gzipped file (ending in .ssm) containing package-specific directories and files with a specific organization and which is installed in a domain
- **source package file**: tar-gzipped file (ending in .bssm) containing a single mandatory file and optional support files for building a binary package file
- **installed package**: an installed instance of a package file
- **package name**: the name of a package with the full name (e.g., abc_1.0_all) or the short name (abc)
- **platform**: a string used to uniquely identify: an operating system or distribution, a release/version of it, and a hardware architecture
- **repository**: a directory containing package files

The ssm tool

Working with SSM is done using the `ssm` tool. The functions of ssm are available through the various subcommands which are used to manage and inspect domains, and some other helpful operations. The `ssm` tool and subcommands all support the `-h` and `--help` options to show usage/help.

Domain

A domain is organized as:

```
<domain_name>/
  etc/
  lib/
```

```
<platform>/
  bin/
  etc/
  lib/
<package_name>/
  bin/
  etc/
  lib/
...
```

Package

A package consists of files and directories with a single, top-level directory with a name matching the full package name. A package may be found as a package file (ending in .ssm) or installed/upacked to a domain.

The package structure is:

```
<package_name>/
  .ssm.d/
  bin/
  etc/
  lib/
```

There are three files that can be put under `.ssm.d/` - `control.json`: the package metadata (REQUIRED) - `post-install`: script executed after the package contents are unpacked but before it's installed - `pre-uninstall`: script executed before the package is removed

Platforms

A platform is a hypen-separated 4-tuple of the format:

```
<dist>-<version>-<arch>-<objmode>
```

- dist: distribution or operating system (e.g. ubuntu)
- version: version/release
- arch: processor (e.g. i386, amd64)
- objmode: object mode of the binaries and libraries (e.g. 32, 64)

Using SSM

Privileged access is not required for using SSM to deploy domains and packages. Any user can do this. Further, domains may be located anywhere within the filespace.

Areas of responsibility for domain managers are:

- domain management
- domain creation, removal, and updating
- package management
- package installation/uninstallation, publishing/unpublishing

Create a new domain

A domain is a specially structured directory to hold packages.

Create a domain:

```
$ ssm created -d ~/myssm
```

It is created under your `$HOME` directory but as mentioned above, it can also be created under a shareable space like `/fs/ssm` so an organization can use the same set of packages.

The domain will look like:

```
myssm/
  etc/
    ssm.d/
      broken/
      installed/
      published/
      meta.json
```

Notes:

- the domain name/directory name is `myssm`
- `etc/ssm.d/` is special for SSM
- `installed/` holds references to all installed packages
- `published/` holds references to all published packages
- `meta.json` holds special information for the domain such as the label; it should never be modified directly

Create a package

Create some directories for the package:

```
$ mkdir hello_1.0_all
$ mkdir hello_1.0_all/.ssm.d
$ mkdir hello_1.0_all/bin
```

The release section (“1.0”) can be anything like “latest”.

Create the `hello_1.0_all/.ssm.d/control.json`:

```
{
    "name": "hello",
    "version": "1.0",
    "platform": "all",
    "summary": "Hello World",
    "description": "A hello world package."
}
```

Create the `hello_1.0_all/bin/hello.sh` script:

```
#! /bin/bash
echo "hello world!"
```

Change the file mode:

```
$ chmod 0755 hello_1.0_all/bin/hello.sh
```

Create the package file (it has to be tar):

```
$ tar cvfz hello_1.0_all.ssm hello_1.0_all
```

Deploy package

Install the package:

```
$ ssm install -d myssm -f hello_1.0_all.ssm
```

Publish the package for a given platform (a package can be published to many platforms):

```
$ ssm publish -d myssm -p hello_1.0_all -pp ubuntu-20.04-amd64-64
```

List domain

List the domain:

```
$ ssm listd -d myssm -pp ubuntu*
----- platform (ubuntu-20.04-amd64-64) -----
p     hello_1.0_all
```

Use the package

Then use the package:

```
$ . ssmuse-sh -x ~/myssm/hello_1.0_all
$ hello.sh
hello world!
```

Note that `ssmuse-sh` is sourced, that the domain path is targeted (`~/myssm`) and that the package could have had contain multiple binaries and specific configurations like adding environment variables.

Module

Module is a tool for loading non-standard applications or packages into the shell environment. In order to properly load a package, instructions for configuring environment variables such as `PATH` are described in a modulefile.

Compared to SSM

Similarities

At a high-level, SSM and module offer similar functionalities:

- loading of third-party software without admin privileges
- loading self-made applications or libraries
- manual modification of the shell environment at any time

Disadvantages

SSM has some advantages which are attributed to differences in how each tool structures package management.

SSM optimizes the lookup of executables. With SSM, a single entry is added to the `PATH` variable for each domain that is loaded. This is accomplished by utilizing symbolic links to the packages in the domains /bin directory. In contrast, because module loads packages individually, a `PATH` entry is created for each package. By limiting the number entries, the shell is able to locate executables faster. Although it is possible to attain the same optimization with module, SSM has it built-in.

Additionally, SSM has a mechanism for switching package platforms. SSM domains have the option of supporting various system architectures, and can automatically detect which package installation to load. In contrast, package platform selection using module is handled by the user: either using platform-specific modulefile diretories, or by adding checks in the modulefiles for determining correct arch version of package to load

> **Note**
>
> For the reasons described above, SSM is recommended over module.

Using module

In this example, module will be used to load the following script into the shell environment:

```
$ cat ~/packages/hello/1.0/bin/hello.sh
#! /bin/bash
echo "hello world!"
```

Create a modulefile

1. Create a directory for storing modulefiles

```
mkdir -p modulefiles/hello
```

1. Add directives to the modulefile

Create the modulefile with the filename `1.0`:

```
$ cd modulefiles/hello
$ touch 1.0
```

Add the following directives to the modulefile:

```
#%Module1.0

# variable declarations
set pkg_dir        /home/sdum001/packages/hello
set pkg_version    1.0
set load_dir       $pkg_dir/$pkg_version

# environment variable settings
prepend-path  PATH             $load_dir/bin

# information about the module
module-whatis "Command that returns 'Hello world!' version $pkg_version"

proc ModulesHelp { } {
    puts stderr " HelloWorld module example."
    puts stderr " This module loads a basic hello.sh to the environment."
    puts stderr "\n"
}
```

Use the module

1. Enable access to the modulefile

Update the `MODULEPATH` environment variable so that module can find the modulefile:

```
$ module use ~/modulefiles
```

This subcommand lists the available system and user modules that can be loaded:

```
$ module avail
------------------------------------ /home/sdum001/modulefiles -------------------------
hello/1.0

---------------------------------- /usr/share/modules/modulefiles ----------------------
dot  module-git  module-info  modules  null  use.own
```

1. Load the module

```
$ module load hello/1.0
$ hello.sh
hello world!
```

Now the script is available system-wide.

List module information

```
$ module list
Currently Loaded Modulefiles:
 1) hello/1.0
```

Output module details

```
$ module whatis hello/1.0
------------------------------------ /home/sdum001/modulefiles -------------------------
           hello/1.0: Command that returns 'Hello world!' version 1.0
```

Deactivate the module

Once a module is no longer needed, it can be unloaded from the environment.

```
$ module unload hello/1.0
$ hello.sh
bash: hello.sh: command not found
```

> **Tip**
>
> Improve command lookup by removing unused environment variable configurations.

Profile Configuration

A profile is a bash script that defines a set of environment configurations. It encapsulates an environment that fulfills the requirements of either a host architecture, a partner, a group or session type.

ordenv

ordenv is a system for defining, sharing, and loading shell profiles. ordenv profiles typically load SSM domains and set environment variables needed for some HPC tools.

There are different types of ordenv profiles; they are organized in layers that define increasingly specific environments: by site, community, group, shell session type, and host criteria.

Commands

Two programs are provided for using ordenv profiles:

- `ordenv-load` - loads profiles to update the environment (must be sourced)
- `ordenv-list` - lists profiles

Site profiles

Site profiles provide a way for the site/system administrator to set up a base environment common to all users. The environment settings are based on the platform of the host the user is logged into.

Site profiles are selected by date <YYYYMMDD>.

```
$ ordenv-list site
20190814
20191220
20200724
20210129
```

Community profiles

Community profiles provide a way for community-specific setup to be done without affecting others. These are the responsibility of a user-representative (or an administrator).

Community profiles follow this naming convention: <partner>/<YYYYMMDD>.

```
$ ordenv-list comm
eccc/20210309
```

```
nrcan/20190708
nrcan/20200218
nrcan/20200311
```

Group profiles

Group profiles provide a way for group-specific setup to be done without affecting others. These are the responsibility of a user-representative (or an administrator)

Group profiles commonly follow this naming convention: <partner>/<group>/<date>.

```
$ ordenv-list group
nrcan/geobase/gpsc2
nrcan/geode/20190131-exp
nrcan/gia_models/20190625
nrcan/wildfire/gpsc2
```

User profiles

User profiles provide a mechanism for users to set up the environment based on session type, hostname, architecture, and other criteria.

These profiles are placed inside the `~/.profile.d` directory. To specify the session type in which a user profile will be loaded, the profile must be placed in a sub-directory that describes the session type. Three kinds of sessions are supported: batch, interactive, and default. The profile filename describes which host criteria will load the profile.

.profile setup

Although ordenv profiles can be loaded interactively, it is standard for them to be loaded automatically in shell sessions. This is accomplished by defining the ordenv setup inside `~/.profile`: this file contains user-defined shell environment configurations which are applied at login time.

The following lines must be added to `~/.profile`:

```
export ORDENV_SITE_PROFILE=<site-specific profile>
export ORDENV_COMM_PROFILE=<community-specific profile>
export ORDENV_GROUP_PROFILE=<group-specific profile>
. ordenv-load
```

where ordenv-boot.sh (or a similarly named file, e.g., ordenv-boot-20170103.sh) is a helper script which loads required components and then calls ordenv-load.

Note

See the Announcements page at portal.science.gc.ca for updates to the ordenv profiles.

Python Virtual Environment

```
venv/
■■■ bin
■■■ include
■■■ lib
■■■ share
```

A Python virtual environment provides a means for managing dependencies of Python applications. It is a directory in user-space that contains a selected Python interpreter, and a customizable set of third-party modules (also called site packages). Additionally, it contains scripts for instructing the application to target the Python dependencies in the virtual environment.

How it works

By convention, a virtual environment is created for a particular Python application and contains all of the dependencies for that application. The virtual environment is placed at the root directory of the Python project; it is created and managed using Python command-line tools.

In order to load the virtual environment into the executing environment of the application, the `bin/activate` script is sourced. The application will then source the dependencies in that virtual environment, instead of the system-wide modules.

Advantages

Python virtual environments offer significant benefits for developpers:

- dependency isolation between Python apps: virtual environments do not share libraries with others
- possible to install non-standard Python libraries: third-party libraries can be installed without admin privileges since the libraries are installed in user-space
- appplications are self-contained and can be deployed on any host with the same Python version

Good practices

1. **List the third-party module dependencies in a requirements.txt file.**

   This way, it is clear which modules are required. Furthermore, users can easily setup the same virtual environment when collaborating on the development of an application.

1. **Specify an exact version for each module.**

   By default, if no version is specified, pip will install the latest version. If a new version of the module has changes to the module API or behaviour, then the application that depends on it can unexpectably break. Stating a specific version will prevent this.

Tools

There are two main tools for creating and managing Python virtual environments. Both tools provide the same basic functionality and interface.

`virtualenv`

This tool supports Python 2.7+ and Python 3.3+ virtual environments. It is installed on the partner Ubuntu containers (e.g. <partner>-inter-ubuntu-18.04-amd64).

`python3 -m venv`

The Python3 venv library only available for Python3.3+. So, it is not possible to create Python virtual environments for Python2. The venv library is only installed on Centos partner containers.

Limitations

These tools are not available on gpsc-vis cluster. This is because the gpsc-vis cluster is directly accessed by logging onto the host machines, and these tools are only installed on partner containers.

Another limitation is that only third-party libraries that are compatible with the system version of Python2 or Python3 can be installed. If the library or particular version of the library depends on a more recent version of Python, then the library will fail to install.

To find out which versions of Python2 and Python3 are installed on the system, these commands can be used:

```
$ python2 --version
Python 2.7.17
$ python3 --version
Python 3.6.9
```

`virtualenv` example

Here is an example for initializing a Python3 virtual environment on a Ubuntu container.

After logging into a Ubuntu interactive container, change the working directory to the root directory of a Python3 project.

1. Create the Python3 virtual environment directory called, in this example, `venv`:

```
$ virtualenv -p python3 venv
```

This command creates the directory `/venv` which contains sub-directories for holding the virtual environment Python packages as well as scripts for managing the virtual environment.

1. Load the virtual environment into the working environment:

```
$ source venv/bin/activate
```

The `activate` script prepends the `/venv/bin` directory to the `$PATH` environment variable and updates the bash prompt with the virtual environment name. Now all Python3 calls will target the Python3 executable installed in `/venv`. Libraries will also be imported from the virtual environment.

1. Declare the third-party dependencies and their versions in a file named `requirements.txt`:

```
(venv) $ touch requirements.txt
(venv) $ echo "numpy==1.19.0" >> requirements.txt
```

1. Install the dependencies using the pip Python package installer:

```
(venv) $ pip install -r requirements.txt
```

1. In order to unload (i.e. exit) the virtual environment, use the following command:

```
$ deactivate
```

Anaconda

Conda is another Python virtual environment tool that is available to load in partner containers. It offers more options for configuring virtual environments than the previously described tools. However, it is considered experimental, and may fail in some instances.

Loading the tool

```
. ssmuse-sh -x hpco/exp/mib002/anaconda2/anaconda2-5.0.1-hpcobeta2
```

How to Create, Publish, Install, and Load LAMMPS SSM Package (Concrete Example)

LAMMPS is a classical molecular dynamics code with a focus on materials modeling. It's an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator. To use it, one way is to create an SSM package and load it in our environment whenever we need to.

Create SSM Package using Build Helper

Build Helper or BH is a tool to help create SSM packages. In this document we will be using BH version 1.3.

In order to use BH 1.3 the following command must be executed to load the program:

```
$. ssmuse-sh -x main/opt/bh/bh-1.3
```

For more information BH 1.3 refer to this documentation:

https://portal.science.gc.ca/confluence/display/SCIDOCS/bh+-+Build+Helper+User+Guide+1.3

The following is a BH script that creates a LAMMPS SSM package:

bh script (${BH_HERE_DIR}/bh-lammps.py; must be executable

```
#! /usr/bin/env python2
# -*- coding: utf-8 -*-
#
# bh-lammps.py

from os import environ
import sys
from bh import bhlib
from bh.actions import package as actions

def _init(b):
    environ["BH_PROJECT_NAME"] = "lammps"
    environ["BH_PACKAGE_NAMES"] = "lammps"
    environ["BH_PACKAGE_VERSION"] = "29Oct2020"
    environ["BH_PULL_SOURCE"] = "<path to tar file>/lammps-master.tar.gz"

    # override: build in name-version dir
    environ["BH_BUILD_DIR"] =  "%(BH_TOP_BUILD_DIR)s/lammps-master" % environ


def _make(b):

    b.shell(". ssmuse-sh -x main/opt/openmpi/openmpi-3.1.2--hpcx-2.4.0-mofed-4.6--gcc; mkdi
          -D PKG_MPIIO=yes -D PKG_PERI=yes -D PKG_REPLICA=yes -D PKG_RIGID=yes -D PKG_USER-
          -D PKG_USER-OMP=yes -D PKG_USER-REAXC=yes ${BH_BUILD_DIR}/cmake ")
    b.shell("cd ${BH_INSTALL_DIR}/build; cmake --build .")

def _install(b):
    b.shell("cd ${BH_INSTALL_DIR}/build; cmake --install .")

def _package(b):
    actions.package.to_ssm(b)

if __name__ == "__main__":
    dr, b = bhlib.init(sys.argv, bhlib.PackageBuilder)
    b.actions.set("init", _init)
    b.actions.set("pull", actions.pull.unpack_tgz)
    b.actions.set("make", _make)
    b.actions.set("install", _install)
    b.actions.set("package", _package)

    b.supported_platforms = [
        "ubuntu-18.04-amd64-64",
    ]
    dr.run(b)
```

The example above uses cmake which might not be installed. If this is the case, contact your user representative.

Steps

1. init - Initialize: sets up the build environment. This step is always implicitly executed before any others and should not be specified in the step range
2. pull - Pull the source contents from a location and unpack into the ${BH_BUILD_DIR} directory. In the case of LAMMPS you can download the latest version from their website https://www.lammps.org/download.html
3. **make - Make/build the software. In this example we create our own _make method and use the instructions from the LAMMPS website https://docs.lammps.org/Build_cmake.html using cmake.**

1. The first step is to load the GCC openmpi package. This is because, in this example, we are installing MPIIO sub package which requires access to the MPI compiler. Second, we need to create a build folder in our install directory which for BH is specified by ${BH_INSTALL_DIR}.
2. We then run the cmake -D PKG_NAME =yes to specify the sub packages we wish to install for LAMMPS. This page has information on sub-packages available for LAMMPS: https://docs.lammps.org/Packages.html.
3. The last step is to run the cmake –build . command from the ${BH_INSTALL_DIR}/build directory we created initially that will launch the compilation, which, if successful, will ultimately produce a library liblammps.a and the LAMMPS executable lmp inside the build folder.

4. install - Install the built software to a (temporary) destination. We run cmake –install inside the ${BH_INSTALL_DIR}/build directory which installs the LAMMPS executable into this directory.
5. package - Create a package from the installed software and put it in the drop directory (default is $BH_HERE_DIR). This produces an ssm package.

Create control.json and post-install files

control.json

For BH to successfully create the SSM package a control.json must be created in the following directory ${BH_HERE_DIR}/control/lammps/.ssm.d/control.json:

```
{
    "name": "lammps",
    "version": "stable_29Oct2020",
    "platform": "Ubuntu 18.04",
    "maintainer": "behrooz.hedayatil@ssc-spc.gc.ca",
    "summary": "LAMMPS Molecular Dynamics Simulator",
    "description": "See https://www.lammps.org/"
}
```

post-install

The post-install script is also located at: ${BH_HERE_DIR}/control/lammps/.ssm.d/post-install and contains code that adds the location of the lmp executable to the PATH environment variable and also adds the location of the .a files to the LIBRARY_PATH. Make sure this file is executable.

```
#! /bin/bash
#
# post-install <domainHome> <packageHome>

domainHome=$1
packageHome=$2

# create profiles
packageName=`basename ${packageHome}`
profileDirPath=${packageHome}/etc/profile.d
profilePath=${profileDirPath}/${packageName}.sh
loginPath=${profileDirPath}/${packageName}.csh

rm -f ${profilePath} ${loginPath}
mkdir -p ${profileDirPath}

cat > "${profilePath}" <<EOF
export PATH=\$PATH:${packageHome}/build
export LIBRARY_PATH=${packageHome}/build
EOF
```

Run bh-lammps.py

run the command:

```
./bh-lammps.py -p ubuntu-18.04-amd64-64 --local
```

This will run each step mentioned earlier sequentially until the final zap stage.

To receive email reports for troubleshooting specify your email in the ~/.bh/bh.conf file.

Publish and Install LAMMPS SSM Package

In order to use the lammps SSM package we created with BH, it needs to published and installed inside a specific domain.

Create SSM Domain

For the sake of example we will create the domain in ~/lammps but the same procedure applies for any domain location.

Create a domain with a label “LAMMPS”:

```
$ ssm created -d ~/lammps -L LAMMPS
```

Deploy Package

Install the package:

```
$ ssm install -d ~/lammps -f <path to ssm file>/lammps_29Oct2020_ubuntu-18.04-amd64-64.ssm
```

Publish the package:

```
$ ssm publish -d ~/lammps -p lammps_29Oct2020_ubuntu-18.04-amd64-64 -pp ubuntu-18.04-amd64-6
```

List Domain

List the domain:

```
$ ssm listd -d ~/lammps -pp '*'
```

If all was successful IP should appear next to lammps_29Oct2020_ubuntu-18.04-amd64-64 after the above command.

Run LAMMPS Command

To ensure that the lmp executable runs correctly, from the ubuntu 18.04 container, execute the following commands:

```
$ . ssmuse-sh -x main/opt/openmpi/openmpi-3.1.2--hpcx-2.4.0-mofed-4.6--gcc
$ . ssmuse-sh -x main/opt/openmpi-setup/openmpi-setup-0.3-hcoll
$ . ssmuse-sh -d  ~/lammps
$ lmp
LAMMPS (14 May 2021)
OMP_NUM_THREADS environment is not set. Defaulting to 1 thread. (src/comm.cpp:98)
using 1 OpenMP thread(s) per MPI task
```

Before you can use LAMMPS package you must first load GCC openMPI and openMPI runtime configurations:

```
. ssmuse-sh -x main/opt/openmpi/openmpi-3.1.2--hpcx-2.4.0-mofed-4.6--gcc
. ssmuse-sh -x main/opt/openmpi-setup/openmpi-setup-0.3-hcoll
```

These are loaded because LAMMPS uses openMPI to run parrallel processes.

Launch LAMMPS SSM inside SLURM BATCH File

To use LAMMPS inside a SLURM BATCH file the following code can be used as an example

```
#! /bin/bash -l
#SBATCH --job-name=hello
#SBATCH --output=$HOME/hello.out
#SBATCH --partition=standard
```

```
#SBATCH --account=<designated project>
#SBATCH --time=1:00:00
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=2000M
#SBATCH --comment="image=nrcan/nrcan_all_default_ubuntu-18.04-amd64_latest"

. ssmuse-sh -x main/opt/openmpi/openmpi-4.1.1a1--hpcx-2.8.0-mofed-5.1--gcc
. ssmuse-sh -x main/opt/openmpi-setup/openmpi-setup-0.3-hcoll
. ssmuse-sh -d ~/lammps

#lamps lmp executable can be used here along with srun or mpirun
```

Load SSM Package in .profile

An SSM package, whether LAMMPS or any other, can be loaded in the ~/.profile in order to avoid loading it every time the bash session is started.

```
#
# .profile
#
# see https://portal.science.gc.ca/confluence/display/SCIDOCS/Setting+Up+the+Environment


export ORDENV_SITE_PROFILE=20210129
export ORDENV_COMM_PROFILE=
export ORDENV_GROUP_PROFILE=
. /fs/ssm/main/env/ordenv-boot-20201118.sh

. ssmuse-sh -d ~/lammps
```

The OpenMPI packages mentioned earlier must be still be loaded every time so that LAMMPS can work properly.

Load SSM Package in ORDENV Group Profile

ORDENV site profile locations can be found using the $ORDENV_PROFILES_DIR environment variable. There are three such locations:

- comm -> /fs/ssm/main/env/profiles/comm
- group -> /fs/ssm/main/env/profiles/group
- site -> /fs/ssm/main/env/profiles/site

site and comm (community) are managed by HPCO.

For this example we will add the LAMMPS package to a bash script in the following group directory: /fs/ssm/main/env/profiles/group/nrcan/geobase.

Any user that belongs to this group can add packages to the bash scipt.

Create a bash scipt with the most recent date, for example 20210615.sh, and add the following code:

```
#! /bin/bash
#
# group/nrcan/geobase/20201615.sh
#
#

. ssmuse-sh -d ~/lammps
```

If a recent bash script already exists for this group you can add the LAMMPS ssmuse command, or any other ssmuse command, to that file.

Next in the ~/.profile add the bash scipt like so:

```
export ORDENV_SITE_PROFILE=20210129
export ORDENV_COMM_PROFILE=
export ORDENV_GROUP_PROFILE=nrcan/geobase/20200615
. /fs/ssm/main/env/ordenv-boot-20201118.sh
```

This way, anyone in the geobase group can add packages that can be loaded automatically at bash startup.

HPC Utilities

HPC Environment Tools

There exists several tools for engaging with the HPC environment and managing computing resources.

SQM

SQM (Simple Quota Manager) is a tool for listing and modifying disk storage limits.

What are storage quotas?

A storage quota is a restriction on the amount of disk space that is used on a filesystem sub-tree – which is called a volume. These limitations are applied to individual users and groups.

The quota is made up of a soft limit and hard limit. Once the soft limit has been reached, users are generally warned that they are close to reaching the hard limit. Once a user or group reaches the hard limit, it will be no longer possible to write to the volume.

User permissions

There are two types of users: regular and user representatives. Regular users can inspect their personal and group disk usages and allocations. User representatives can inspect and modify disk allocations of groups of users that they oversee.

Supported filesystems

SQM provides a single interface for interacting with multiple types of filesystems, where each type manages quotas differently.

Currently, these filesystems are supported by SQM:

On gpsc:

- /gpfs/fs5
- /gpfs/fs7
- /fs/vnas_*

On gpscc:

- /gpfs/fs3c

Operations

SQM has two main sub-commands: list and modify.

list

```
$ sqm list -u sdum001 -d ~
-------------------------------------------------------------
Quota Path: /fs/vnas_Hnrcan/amspm
Last Updated: 2021-04-20 01:10:09 UTC
Update Frequency: 5 minutes
-------------------------------------------------------------
User Name  Used Space (GB)  Soft Limit (GB)  Hard Limit (GB)
sdum001              0.13                -             10.0
-------------------------------------------------------------
```

modify

```
$ sqm modify -u dam002 -l 10 10 -m G -d /gpfs/fs1/ssc/hpco/test
```

hpcarchive

hpcarchive is a high performance tool for archiving files on the HPNLS archiving system.

This tool allows users to store large amounts of data meant to be preserved for long durations (e.g. historical records), and that are accessed rarely. Each archive is assigned to a user as well as a project.

hpcarchive has optimized operations for managing archives:

- listing files within archives
- searching files/archives using these parameters: tag, regex pattern and date interval
- retrieve
- delete

Getting started

If not already available, load the tool into the working environment:

```
$ . ssmuse-sh -x main/opt/hpcarchive/hpcarchive
```

List projects that can be written to:

```
$ hpcarchive -w
Projects available to archive to :
- ssc_hpco
```

> **Note**
>
> User representatives manage projects that users can write to.

Command examples

Search for files in an archive:

```
$ hpcarchive -L -F 2019-01-01 -T 2021-01-01 -x -c command
@dam002 ssc_hpco     2710 Oct 08 12:38 commands
@dam002 ssc_hpco      458 Oct 04 13:29 command-timer.sh
total 2 - 3.094KB
```

hcron

Hcron is a cron-like tool for scheduling command executions on various hosts from one centralized location.

Hosts

Users must login to a host where the hcron service is available.

On the gpsc,

- hcron1.science.gc.ca.

On the gpscc,

- hcron1.collab.science.gc.ca

Note

Users must request access by submitting a service request to their partner service desk.

Structure

An hcron event is defined in a text file using `field=value` settings. The event files are organized in a tree structure in `~/.hcron/<hcron-host>/events`.

Using hcron

Hcron events are scheduled and managed using the hcron tool.

Create an event file and fill out the directives:

```
$ cd ~/.hcron/hcron1.science.gc.ca/events
$ hcron event myevent
```

To schedule the hcron events:

```
$ hcron reload
```

To list the currently scheduled events:

```
$ hcron list
```

sscp

SSCP (Simple Smart Copy) is an optimized file transfer tool. Its performance is notably better than that of scp.

It improves the speed of file transfers by

1. identifying optimized network routes between source and destination paths; 1. and determining which of these file transfer tools is optimal to use: `mcp`, `bbcp` or `scp`.

The syntax is similar to known tools such as `scp`.

Warning

Currently not available on the GPSC and GPSCC.

fmgr

The russfmgr (File Manager) server allows user representatives to manage files/directories for which they are responsible: those of a user group, or those on a storage device. It permits user representatives to manipulate the files/directories they oversee even though they do not have the Unix file permissions.

Note

User representatives must submit a formal request to be granted manager rights.

The file and directory operations are available:

- change ownership
- get status (file/directory meta-data)
- move
- rename
- delete

Science Web Services

There are three web services available on the Science network: the HPC Science documentation portal, HPC statistics, and the GitLab web application.

These websites are accessible from the GC Cloud and GCSN networks.

HPC Science Portal

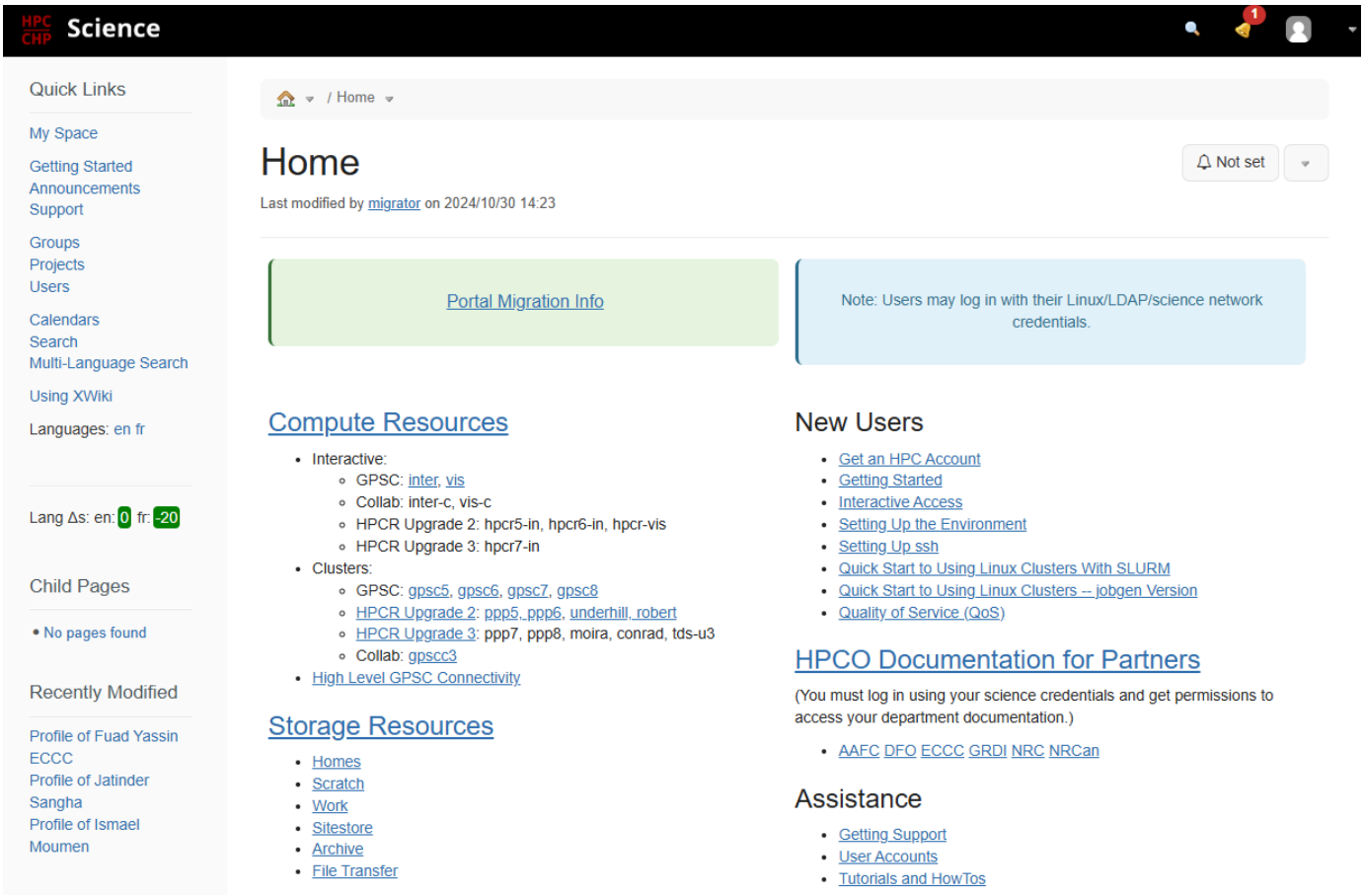
The portal site provides documentation about Science HPC resources including:

- specifications of HPC systems
- descriptions of HPC software
- guides and tutorials
- announcements such as software updates, new ordenv profiles, and service disruptions

It's also a place where users themselves can create their own documentation.

The landing page is accessible at this URL: https://portal.science.gc.ca

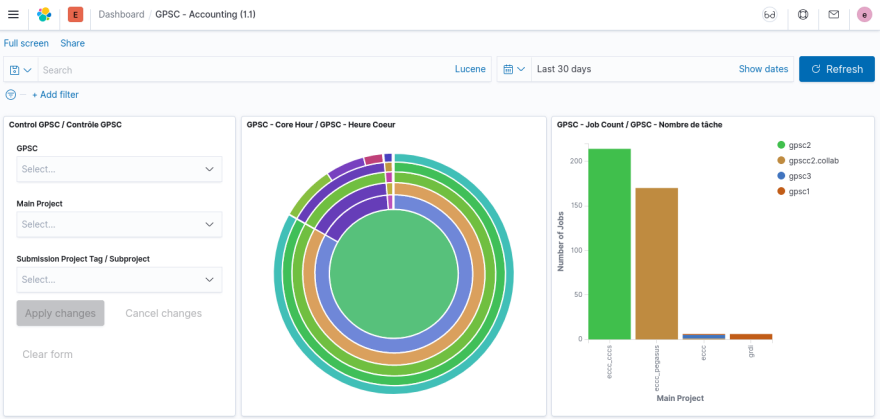
If this URL is not accessible, this alternative URL should work: https://portal-gccloud.science.gc.ca



This site is the first place to look when seeking answers to HPC Science questions.

hpc-stats

The hpc-stats website provides visualisations of HPC resource statistics. For instance, users can view their personal and group HPC resource usages and allocations including CPU hours and storage space.



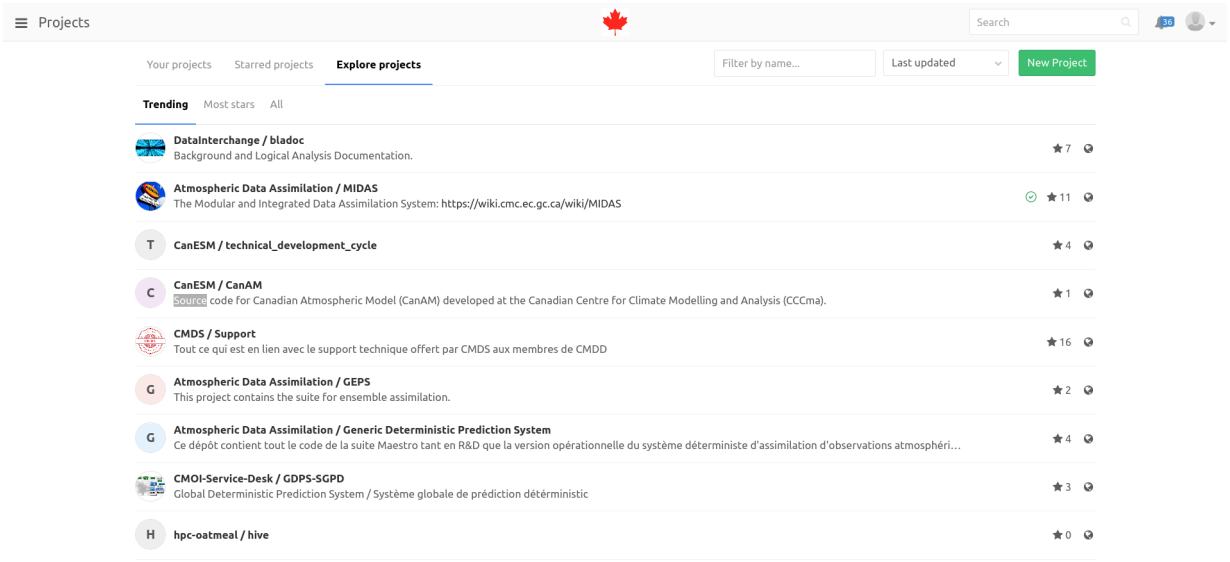
Users login using their Science account credentials at https://hpc-stats.science.gc.ca

If this URL is not accessible, this alternative URL should work: https://hpc-stats-gccloud.science.gc.ca

GitLab

GitLab is a web application for managing remote git repositories. It also includes several features for applying software development practices such as issue-tracking, branch management, a code reviewing tool, version controlling, and much more.

GitLab provides a way for users to collaborate on software projects.



Users login with their Science account credentials at https://gitlab.science.gc.ca

If this URL is not accessible, this alternative URL should work: https://gitlab-gcsn.science.gc.ca

Parallelism and Slurm

Scheduling Overview

The Slurm Workload Manager is software for managing and scheduling jobs on Linux clusters. It has many functions including:

- allocates cluster resources to jobs;
- provides utilities for creating and managing jobs;
- arbitrates the scheduling of jobs competing for compute resources;
- and provides a means for monitoring and limiting compute resource usages.

Access

Users must login to the GPSC using their Science account credentials in order to submit and manage Slurm jobs on the GPSC clusters.

Command-line tools

Slurm provides many command-line utilites for managing jobs or for obtaining scheduling information.

These are the most commonly used commands:

- `sacct`: view accounting information about jobs
- `sacctmgr`: view accounting information about projects
- `salloc`: request compute resource allocations
- `sbatch`: submit a batch job
- `scancel`: cancel a job
- `sinfo`: list information about the state of cluster nodes and partitions
- `squeue`: list the state of all jobs on the cluster
- `srun`: launch job steps or start an interactive job

Slurm Components

Job Scheduler

The job scheduler is the program that oversees jobs submitted to the cluster. Once a job is submitted, the scheduler is responsible for allocating the requested compute resources and dispatching the job on the cluster. It is also determines the order in which jobs, that are contending for resources, are started on the cluster.

Partitions

A partition is a sub-set of cluster compute nodes that has assigned resource and scheduling limits. Nodes can belong to more than one partition.

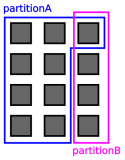


Diagram of a cluster with two partitions.

For each partition, the job scheduler maintains an ordered list of jobs to be run. Because of this, a partition can also be considered as a job queue.

To list all partitions on a cluster:

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard*    up   infinite     44   idle ib10-fx01f42-be[01-04],ib10-fx01h42-be[01-04],ib10-
xfer         up      30:00      3   idle ib10-fe[01-03]
testing      up   infinite      4   idle ib10-fx01e42-be[01-04]
amd          up   infinite      1   unk* ib10-demo1
gpu-v100     up   infinite      1   idle ib9-gpu02
```

The asterix (*) denotes the default partition that will be assigned to a job if no partition is specified in the job submission request.

Accounting

All users are members of one or more usage accounts which are equivalent to projects (and GPSC Linux groups). Each user's default account has the same name as their primary Linux group.

Usage accounts are used to manage many per project administration tasks:

- assign compute resource or scheduler limits;
- assign cluster shares which affects priority weight;
- track compute resource usages of projects and users;
- and restrict access to clusters or partitions.

Each job is attributed to an account, and the resources that have been used by the job are charged against this account. Users have the option of selecting whichever usage account they belong to; if an account is not specified in the job submission, then the user's default account is used.

Allocations

There are no limitations on compute resources, so all users are able to request as much resources on a partition as possible; however, there are project-specific restrictions on job wallclock time and on the number of jobs a user can submit at any given time.

For projects that have not been allocated cluster shares, the jobs run under these projects are limited to a wallclock time of 6 hours, and each user is limited to 100 jobs at a given time.

In contrast, projects that have been allocated shares have no wallclock or job count limits.

To list a user's project allocations:

```
$ sacctmgr -s show user name=$USER
```

Once a user's project limits have been reached, the user is no longer permitted to submit jobs. The user may however submit the job under a different account.

Job prioritisation

Each pending job is assigned a priority weight which is a positive integer that the scheduler uses to determine the placement of the job in the queue. Jobs are ordered from lowest to highest priority.

Job priority is calculated as a function of several factors including:

Project shares

An integer that describes the portion of cluster resources that a project has priority to use. Jobs that are attributed to a project that has more shares will be prioritized over jobs that belong to project with fewer shares.

Job age

Job priority increases with respect to how long the job waits in the queue.

Fair-share

This factor is calculated by the difference in a project's allocated shares and the amount of shares that have already been consumed. The fair-share factor increases the more a project is under-utilizing its allocated shares.

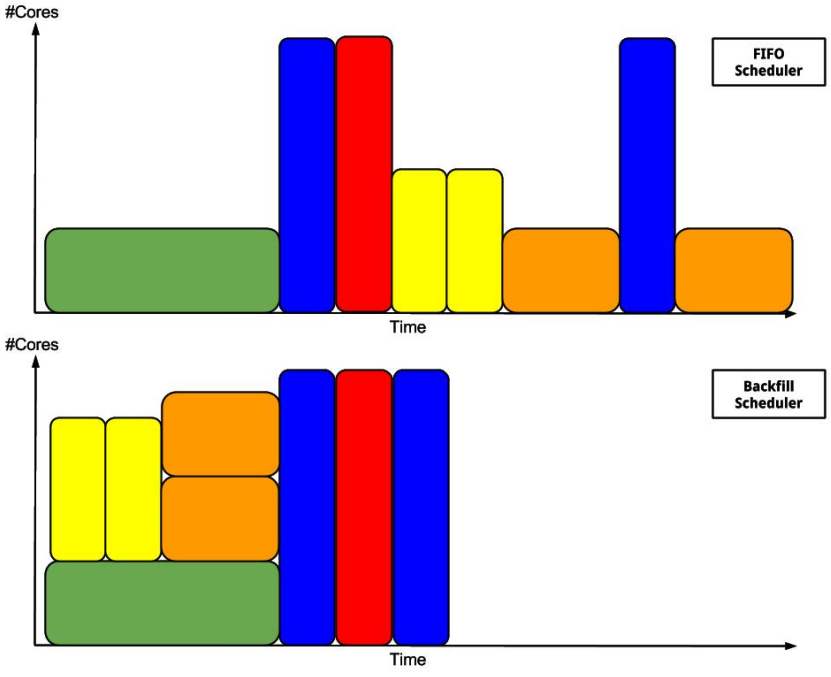
Job size

Jobs that request fewer compute resources than larger-sized jobs are prioritized.

Scheduling Algorithm

The scheduler does not strictly dispatch pending jobs in order of priority. The scheduler uses an algorithm called backfill scheduling which has rules for improving job throughput. Using this strategy, lower priority jobs can be scheduled to run before higher priority jobs in the queue. This occurs if the lower priority job requests resources such that it can finish before the scheduled start time of any higher priority jobs.

As a result, this scheduling algorithm yields greater cluster utilization than the standard First in, first out (FIFO) algorithm.



Comparison of FIFO and backfill scheduling algorithms. In this example, the backfill scheduling algorithm is able to schedule the jobs so that they finish in nearly half the time compared to the FIFO method.

Submitting a Job

The job submission process has a number of steps:

1. Create the jobscript
2. Choose the operations the job will perform (i.e. commands to run)
3. Choose the compute resources that will satisfy the operations
4. Examine how to configure the job environment to satisfy the operations
5. Submit the jobscript

Defining a job file

A job file (or job script) is a shell script that defines the parameters that will be used to run a job.

Below is an example a simple job file called *myjobfile.slurm*:

```
#!/bin/bash -l
#SBATCH --job-name=base
#SBATCH --time=00:10:00
#SBATCH --comment="image=nrcan/ps_all_default_ubuntu-24.04-amd64_latest"
#SBATCH --account=ps_dset

# job environment set-up

# commands to run
srun myprogram
```

The lines that begin with *#SBATCH* are Slurm directives: parameters that the scheduler uses configuring the job. There exists numerous configuration options, but the directives listed in this example are essential for every job submission. A complete list of options is available here: https://slurm.schedmd.com/sbatch.html

Select resource amounts

1. Assign a project to the job

Select the project from the list `sacctmgr -s show user name=$USER`.

For example,

```
#SBATCH --account=nrcan_asmpm
```

2. Choose a wallclock limit

Wallclock time is the amount of time allocated for running the job; it does not include the job's time in the queue.

For instance, to select a wallclock time of 2 hours:

```
#SBATCH --time=00:02:00
```

3. Choose the number of nodes, tasks, and CPUs per task

Here are some suggestions for appropriate resource allocation for different types of jobs.

For serial code, which can only run on 1 CPU:

```
#SBATCH --nodes=1
#SBATCH --ntasks=1
```

```
#SBATCH --cpus-per-task=1
```

For multithreaded code that will be run on a full node on GPSC:

```
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=44
```

For parallel code:

```
#SBATCH --nodes=<M>
#SBATCH --ntasks=<N>
#SBATCH --cpus-per-task=1
```

Configure the environment

Jobs are run within GPSC containers. Select the partner image with the desired platform:

```
#SBATCH --comment="image=nrcan/nrcan_all_default_ubuntu-24.04-amd64_latest"
```

To configure the job environment to be identical to user's login environment as defined in the .profile, use this bash shebang:

```
#!/bin/bash -l
```

If the job operations require particular environment settings or non-standard tool, commands must be added to load these requirements using `. ssmuse-sh` for example.

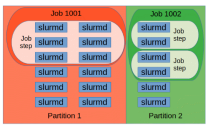
Submit the job file

Once the job file is fully defined, it can be submitted in batch mode using `sbatch`:

```
$ sbatch --cluster=gpsc5 myjobfile.slurm
```

Anatomy of a job

A job has allocated compute resources for doing work, and the work that is carried out is structured with job steps and tasks.



A job has one or more steps, and each step has one or more tasks that use one or more CPUs.

Job

A job is an allocation a defined amount of compute resources assigned to a user for a specified amount of time. The `sbatch` and `salloc` commands are used for requesting the resources.

Steps

A step, usually created with `srun`, is a set of tasks that reserves all or parts of the job resource allocations. A job can be separated into one or more steps.

Tasks

A task is a unit of execution or work; it can be considered as a process or a thread. Tasks are assigned to requested compute resources (CPU and memory). Task resource allocations can be defined at job level or at step level.

Defining steps and tasks

Slurm is loose in its usage of the terms core and CPU. They should be read as the same since Slurm has the ability to make cores of a single processor “independent” to users. In other words, requesting 1 CPU would not reserve all the cores of that processor but instead use 1 core of it.

One would thus submit a job using `sbatch` and use `srun` to create steps with scripts.

From the Slurm documentation: `sbatch`: sbatch is used to submit a job script for later execution `srun`: srun is used to submit a job for execution in real time (interactive/link to the current process)

Parameters `--ntasks`, `--nodes`, `--cpus-per-task`, `--ntasks-per-node` are the same for both `srun` and `sbatch`

Note that `srun` will trigger an allocation if there are no jobs that exist on the partition.

Understanding job states

A job transitions through many standard states during its lifetime following this order: pending, running, completing, and completed.

Here is a description of each job state:

State	Code	Description
PENDING	PD	Job is waiting in the queue for resources to become available, or is waiting behind higher-priority jobs
RUNNING	R	Job is executing
COMPLETING	CG	Job has finished executing, but there remains some processes to clean up
COMPLETED	CD	Job has successfully completed

Current job states are obtainable using the `squeue` command.

Job arrays

A job array is a mechanism for submitting a job that needs to be run multiple times with very few differences between the jobs.

```
#SBATCH --array=0-1

# If 0, do this, if 1 do that, etc.
# The id is used to work on a subset of the work.
echo "$SLURM_ARRAY_TASK_ID"
```

Interactive jobs

There are three ways of thinking about an interactive job.

1. Using `srun` would connect stdout/stderr to the current shell like so `srun --time=00:00:10 echo "hello"`.
2. By only allocating the resources with `salloc`. This logs the user in the node itself, `salloc --time=00:00:10`
3. It’s possible to connect to a given job by creating an additional step like so `srun --jobid=<id> --pty bash -i`

Good practices

Consider putting related work into a single Slurm job with multiple job steps both for performance reasons and ease of management. Each Slurm job can contain a multitude of job steps and the overhead in Slurm for managing job steps is much lower than that of individual jobs.

Job arrays are an efficient mechanism of managing a collection of batch jobs with identical resource requirements. Most Slurm commands can manage job arrays either as individual elements (tasks) or as a single entity (e.g. delete an entire job array in a single command).

Example

```
#SBATCH --nodes 8
#SBATCH --tasks-per-node 8
# The job requests 64 CPUs, on 8 nodes.

# First step, with a sub-allocation of 8 tasks (one per node) to create a tmp dir.
# No need for more than one task per node, but it has to run on every node
srun --nodes 8 --ntasks 8 mkdir -p /tmp/$USER/$SLURM_JOBID

# Second step with the full allocation (64 tasks) to run an MPI
# program on some data to produce some output.
srun process.mpi <input.dat >output.txt

# Third step with a sub allocation of 48 tasks (because for instance
# that program does not scale as well) to post-process the output and
# extract meaningful information
srun --ntasks 48 --nodes 6 --exclusive postprocess.mpi <output.txt >result.txt &

# Four step with a sub-allocation on a single node (because maybe
# it is a multithreaded program that cannot use CPUs on distinct nodes)
# to compress the raw output. This step runs at the same time as
# the previous one thanks to the ampersand `&`
OMP_NUM_THREAD=12 srun --ntasks 12 --nodes 1 --exclusive compress output.txt &

wait
```

References:

- https://slurm.schedmd.com/quickstart.html
- https://researchcomputing.princeton.edu/support/knowledge-base/scaling-analysis
- https://researchcomputing.princeton.edu/support/knowledge-base/slurm
- https://stackoverflow.com/questions/46506784/how-do-the-terms-job-task-and-step-relate-to-each-other/46532581#46532581

Parallelism

Conceptual Mapping with Slurm

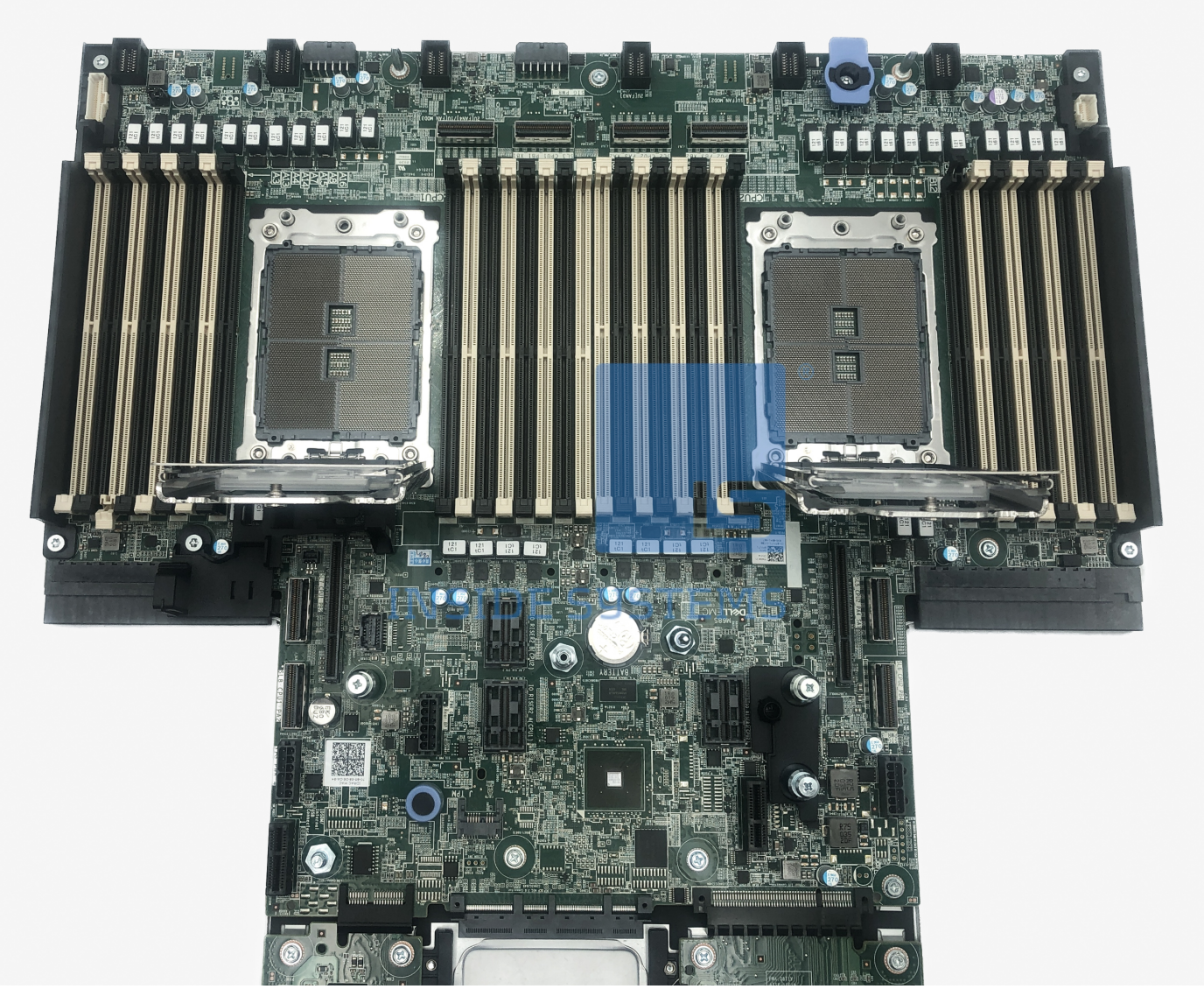
Considering the following specifications of a node on GPSC:

```
# lscpu

Architecture:                    AMD Infinity
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
CPU(s):                          64
On-line CPU(s) list:             0-63
Thread(s) per core:              1
Core(s) per socket:              32
Socket(s):                       2
NUMA node(s):                    2
Vendor ID:                       AMD EPYC
Model:                           7532
Model name:                      AMD EPYC 7532 2.40GHz
```

```
L3 cache:                        256MB
NUMA node0 CPU(s):               0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,4
NUMA node1 CPU(s):               1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,4
```

The motherboard, 2 sockets:



- Sockets are the physical entry of a CPU. The motherboard above has two physical entries, thus 2 CPUs for that node.
- Each socket entry has access directly to half the cores (32).
- Each core has 1 thread (hyperthreading is only active on GPU nodes. CPU nodes are 1 thread per core.).
- All cores have access to memory on that node.

In theory, the HPC nodes will always run programs in parallel (as opposed to concurrent) where the “one task one core” rule is applied since we have 1 thread per core.

Warning

../_static/core.gif

Python can only do concurrent work even though multiple processors exist because of the GIL (Global Lock). Note that the GIL will be dismantled from Python 3.13 . TensorFlow, Numpy and Pytorch, to name a few, are built using C to bypass this limitation.

When launching a job with Slurm, it will by default assume 1 task for 1 node. Remember that a job is the sum of all tasks resources. In practice, the tasks will be the number of *units* of work required to be executed in parallel.

In other words, *–ntasks* means the maximum amount of tasks that can be executed in parallel. Combined with *–ntasks-per-core*, this would limit the number of work for a given core.

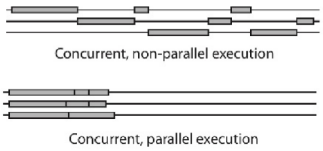
If the number of tasks *–ntasks* exceeds the number of cores, Slurm will create a logical allocation where tasks will be oversubscribed and compete for the physical CPUs. In this scenario, the operating system (OS) will do context switching between each of the tasks. Consequently, the tasks on a single core will start working *concurrently* and performance might be reduced.

Finally, working with multiple nodes is where paralellism on HPC starts to really kick in. However, since the memory is now distributed and inaccessible by all tasks, different patterns need to be considered to fully harness the capability.

Introduction

Definitions

- Concurrent program: A program that contains multiple threads that collaborate via shared variables or message passing. A single process.
- Parallel program: Similar to concurrent programs, the difference is that each thread is using its own processor.
- Distributed program: Also concurrent and/or parallel but the processes communicate through a network and live on different machines with distinct memory.



- Task granularity: Fine-grained parallelism means individual tasks are relatively small in terms of code size and execution time. Data is transferred among processors frequently through lots of communication. **The finer the granularity, the greater the potential for parallelism but the greater the overheads for communication and synchronization.**
- CPU-bound: The program execution performance is limited primarily by the CPU capacities.
- IO-bound: The program execution performance is limited primarily by the writing or reading (input/output) capacities.
- Static repartition: The number of threads is known before executing the program so the hardware needed can be mapped precisely.
- Critical path: The longest directed path in the task dependency graph.
- Embarrassingly Parallel: Processes do not communicate with each other and run completely independently, possibly including a step to merge results when the last process finishes. These types of processes are typically CPU or I/O limited.

Parallel Program Patterns

Fork-Join

A method of programming in which one or more child threads branch out from the root task to do work in parallel. These child threads would then “join” when the work is done. Mostly shown as a tree, this method implies a recursive sequence where the forked steps are done in parallel.

Loop parallelism

Often seen using OpenMP (ex.: `parallel for`), this pattern makes use of the loop mechanism to distribute its work for each iteration to a single thread (fine grained) or through a defined number of thread (coarsed grained).

Data parallelism (Map and Reduce)

The same operation is applied uniformly to all elements of a given set, as the collection is generally homogeneous. Data parallelism is often known by the terms Map or Reduce or a hybrid of the two: MapReduce.

Producer/Consumer (or TasksBag)

In this pattern, there is usually one “master” node providing tasks dynamically to a given set of threads when they become available. This approach is particulary useful when the data to be worked on is heterogeneous or when its size varies unknowningly. The disavantage is the overhead created by the synchronization and the communication.

Stream (filter and pipeline)

The processing of the data is decomposed into a series of independent steps like in a factory. The output of a step is expected to be an input of another step. Each step would work in parallel on its own processor.

PCAM Approach on Designing Parallel Algorithms

With parallel computing, the goal is to minimize synchronization and interprocess communication, as well as to distribute the work equally between processors. The PCAM approach shown below is mainly for MPI (message passing) programs but its logic can apply to other types of parallelism as well.

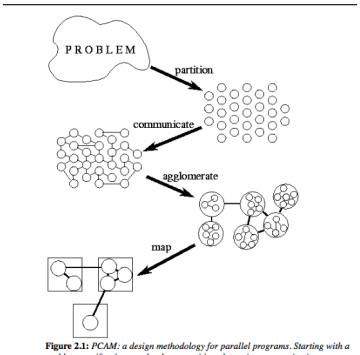


Figure 2.1: *PCAM: a design methodology for parallel programs. Starting with a problem specification, we develop a partition, determine communication requirements, agglomerate tasks, and finally map tasks to processors.*

1. Partitioning

Expose as many parallelism opportunities as possible by making the tasks as granular as possible by reducing whether the calculation or the data.

Two approaches are usually known: domain decomposition and functional decomposition. Domain decomposition (e.g. 3D matrices) would partition the work using the data and then associate the calculation needed to work it. Functional decomposition (e.g. climate model) would do the calculation before and then determine how this data would be associated back together.

The decomposition should be designed to have similar-sized tasks, provide more tasks than processors, and avoid redundancy between data and calculation. Also, the provided number of tasks should increase if the problem at hand gets bigger in size.

2. Communication

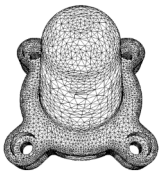
a. Local vs Global

Local means that one task communicates with a small number of other tasks (neighbors). Contrary to global where many tasks would communicate with one (i.e.: MPI_Gather).

b. Structured vs non-Structured

A grid is a structured pattern, whereas a assembly design mesh would be non-structured.

Here’s an example of an unstructured “neighborhood”:



c. Static vs Dynamic

Static means that communication partners are known from the start. A dynamic setup would change partners along the way.

d. Synchronous vs Asynchonous

Either the sender and the receiver mutually block their communication (synchronous) or the sender doesn't (asynchronous).

3. Agglomeration

This is where the parallelism takes a more "applied" step. The objective is to reduce communication by reducing the number of tasks so they would work locally on a node. At this point, the number of processors would still need to be smaller or equal to the number of tasks.

- Reduce the communication cost
- Keep flexibility so it can scale
- Preserve program maintainability

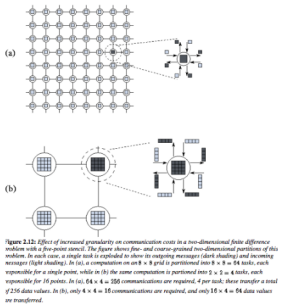


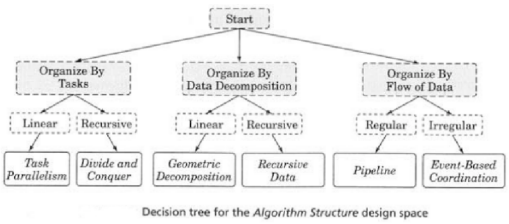
Figure 2.12: Effect of increased granularity on communication costs in a two-dimensional finite difference problem with a five-point stencil. The figure shows fine- and coarse-grained two-dimensional partitions of this problem. In each case, a single task is exploded to show its outgoing messages (dark shading) and incoming messages (light shading). In (a), a computation on an 8×8 grid is partitioned into $8 \times 8 = 64$ tasks, each responsible for a single point, while in (b) the same computation is partioned into $2 \times 2 = 4$ tasks, each responsible for 16 points. In (a), $64 \times 4 = 256$ communications are required, 4 per task; these transfer a total of 256 data values. In (b), only $4 \times 4 = 16$ communications are required, and only $16 \times 4 = 64$ data values are transferred.

4. Mapping

Basically, which processor would do which task(s) based on the requested resources. The important point here is to keep in mind the *load balancing*. In other words, how would the tasks be distributed on the processors? Would there be a script monitoring the global state of the work? Would the work be executed in serial, do jobs have dependencies between each other, etc.

Conclusion

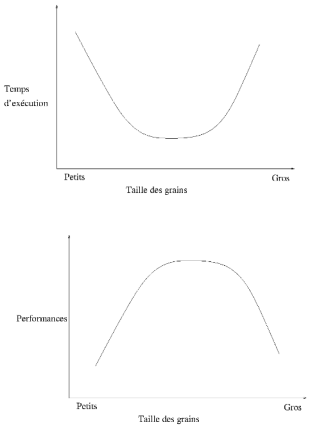
There is a lot more about parallelism than can be covered here, if more details are needed, feel free to explore the tree below to better grasp the available parallelism techniques:



Decision tree for the *Algorithm Structure* design space

Performance

Parallelism is a way to increase performance of a given software. It is not always guaranteed that it will succeed in doing so. Usually performance will be distributed like so depending on the task's granularity and if the job is scalable:



Consequently, it is always a good idea to benchmark the programs to make sure the performance is maximized.

Benchmarks

Using the command `time` is also a good approach like so:

```
#!/bin/bash
#SBATCH --job-name=MyJob
...
time srun mymodel
```

An other simple approach for benchmarking a job is to manually wrap the code with:

```
#!/bin/bash
#SBATCH --job-name=MyJob
...

start=`date +%s.%N`
srun mymodel
end=`date +%s.%N`

runtime=$(echo "$end - $start" | bc -l)
echo "$SLURM_JOB_NAME:$runtime"
```

This wrapping offers the advantage of timing multiple commands.

`time` and the wrapper ensure the runtime is calculated on the job itself without considering the scheduler setup and synchronization. It also provide an easy way to customize the needed benchmarks.

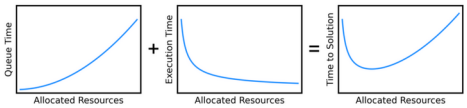
Previous information on jobs can be obtained with `sacct` or by accessing hpc-stats.science.gc.ca .

Serial First

It is a good practice to always start building and benchmarking the program without parallelization using 1 CPU with the appropriate memory. The job's runtime will be at its maximum, and this value can be used as a foundation for evaluating job performance. Increasing the number of CPUs can help determine whether the executed program can effectively use additional resources. Software used for modeling usually knows how to handle multiple resources. Still, it is a good idea to confirm that the model in question is scalable and therefore suited for parallelism.

Time-to-Solution

It is not a good idea to request much more compute resources than what would be used. The scheduler, on a shared and fully used system, would have difficulty reserving a bigger set of resources which would increase the wait time of the job in the queue. Also, asking for more resources than needed would reduce the allocated fairshare portion assigned to the given project which negatively impacts the job's priority score. So, it is advantageous to select resource amounts that reduce the job's wait time, but also allow the job to execute in a timely manner.



References:

- https://www.labunix.uqam.ca/~tremblay_gu/INF7235
- https://www.dais.unive.it/~calpar/New_HPC_course/5_Parallel_Patterns.pdf
- https://pureadmin.qub.ac.uk/ws/portalfiles/portal/16748759/A_case_study_of_OpenMP_Applied_to_MapReduce_style_Computations.pdf
- https://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf
- https://login.scg.stanford.edu/faqs/cores/
- https://researchcomputing.princeton.edu/support/knowledge-base/scaling-analysis

MPI

Introduction

MPI stands for *Message Passing Interface*. When used, MPI creates processes "statically", meaning that there's a fixed number of processes launched at the beginning of its initialization. Each process possess its own private memory space and should be seen as a program in itself.

MPI is based on the SPMD (single program, multiple data) model.

> This is the most common way to organize a parallel program, especially on MIMD computers. The idea is thata single program is written and loaded onto each node of a parallel computer.Each copy of the single program runs independently(aside from coordination events), so the instructionstreams executed on each node can be completely different. The specificpath through the code is in part selected by the node ID.

Communication

MPI has the ability to communicate in two ways:

1. Through a process to process approach (MPI_Send, MPI_Recv, …)
2. Through a collective, a process that communicates with multiple processes, approach (MPI_Gather, MPI_Scatter, …)

How to use it

The steps to follow are: 1. Compile with the proper compiler 2. Choose the correct MPI library 3. Execute the binary with the correct MPI type

OpenMPI (default)

By default, the *gcc* compiler is available, step 1 above can be skipped.

At compile time, this package needs to be loaded:

```
. ssmuse-sh -x main/opt/openmpi/openmpi-4.1.1a1--hpcx-2.8.0-mofed-5.1--gcc
```

and then, the compilation is done like so:

```
mpicc -o mpihello mpihello.c
```

Users will be notified when a new MPI version comes along.

Then at runtime:

```
#!/bin/bash -l
#SBATCH ...

. ssmuse-sh -x main/opt/openmpi/openmpi-4.1.1a1--hpcx-2.8.0-mofed-5.1--gcc

#only for slurm, load hcoll which improves the communication between processes
. ssmuse-sh -x main/opt/openmpi-setup/openmpi-setup-0.3-hcoll

srun myprogram
```

Intel Compilers

Using intel compilers follows the same approach except that the intel compiler and the related MPI library need to be loaded for steps 1 and 2.

```
. ssmuse-sh -x main/opt/intelcomp/intelpsxe-cluster-19.0.3.199
. ssmuse-sh -x main/opt/openmpi/openmpi-3.1.2--hpcx-2.4.0-mofed-4.6--intel-19.0.3.199
```

IMPI

not supported yet

Conclusion

> One aspect of concern, particularly to novices, is the large number of routines comprising the MPI specification. In all, there are 128 MPI routines, and further extensionswill probably increase their number. There are two fundamental reasons for the sizeof MPI. The first reason is that MPI wasdesigned to be rich in functionality.This is reflected in MPI's support for derived datatypes, modular communication viathe communicator abstraction, caching, application topologies,and the fully-featuredset of collective communication routines. The second reason forthe size of MPI reflectsthediversity and complexity of today's high performance computers.This is particularly true with respect to the point-to-point communication routineswhere the different communication modes arise mainly as a means ofproviding a setof the most widely-used communication protocols. […] The availability of a largenumber of calls to deal with more esoteric features of MPI allows one to provide asimpler interface to the more frequently used functions.

References:

- https://www.labunix.uqam.ca/~tremblay_gu/INF7235/
- http://www.netlib.org/utk/papers/mpi-book/node198.html

OpenMP

OpenMP is an interface for parallel programs that are designed to work with shared memory. OpenMP is not a language in itself; it's more like a set of routines that can be adapted to multiple languages.

OpenMP is based on the fork-join model: executing tasks in parallel, and then joining the results.

How to use it

At compilation, the compiler needs to know that OpenMP is used like so:

```
gcc -fopenmp myprogram.c
```

At runtime, the number of threads needs to be specified, like so:

```
#!/bin/bash -l
#SBATCH ...
```

```
export OMP_NUM_THREADS=20
./myprogram
```

The number of threads can exceed the real numbers on a node (i.e.: 44 threads for GPSC3). When threads become “virtual” the main one will have to do more synchronization work between them.

Example

```
# Example 1
# Each thread will do the entire loop in any order
void foo( int n, int  nb_threads ) {
    printf( "foo( %d, %d )\n", n, nb_threads);

    omp_set_num_threads( nb_threads  );
# pragma omp parallel
    for( int i = 0; i < n; i++ ) {
        int id = omp_get_thread_num();
        printf( "i = %d: id = %d\n", i, id );
    }
}

# Example 2
# The first thread will do the loop
void foo( int n, int  nb_threads ) {
    printf( "foo( %d, %d )\n", n, nb_threads);

    omp_set_num_threads( nb_threads  );
# pragma omp for
    for( int i = 0; i < n; i++ ) {
        int id = omp_get_thread_num();
        printf( "i = %d: id = %d\n", i, id );
    }
}

# Example 3
# Each thread will do an increment of the loop in any order
void foo( int n, int  nb_threads ) {
    printf( "foo( %d, %d )\n", n, nb_threads);

    omp_set_num_threads( nb_threads  );
# pragma omp parallel for
    for( int i = 0; i < n; i++ ) {
        int id = omp_get_thread_num();
        printf( "i = %d: id = %d\n", i, id );
    }
}
```

Reference:

- https://www.labunix.uqam.ca/~tremblay_gu/INF7235/

Job Management

List jobs

```
$ squeue  # list all jobs on the cluster
$ squeue -u $USER  # list personal jobs
```

Submit a batch job

```
$ sbatch <jobscript>
```

Hold jobs in queue

In order to prevent a job from running, the following command can be used:

```
$ scontrol hold <jobid>
```

The job will be assigned a priority of 0.

To release the job:

```
$ scontrol release <jobid>  #
```

> **Note**
>
> The hold command has no effect on running jobs.

Delete jobs

To stop and delete a job,

```
$ scancel <jobid>
```

Troubleshooting

Several Slurm tools are available for diagnosing job issues and poor job performance.

The `squeue -u $USER` command lists information about the user's current jobs on the cluster and provides a good starting point for investigating job issues.

The `ST` column shows which state the job is in, and if the job is not running, the `NODELIST(REASON)` column will indicate the reason why.

The example below shows that the user has one job and that it is in the pending state:

```
$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          67148235  standard hurrican  sdum001 PD       0:00      1 (Priority)
```

The following are some typical scenarios a job can encounter as well as some possible solutions to deal with them:

Long wait time

It is normal for a job to be in the pending state for some time. However, if the job is in this state for an unprecedented amount of time, there can be recourses for improving its start time.

These are the common reasons for a job to be in the pending state:

Reason Code	Description
Priority	Higher priority jobs exist in the queue
Dependency	The job depends on another job to finish before it can run
Resources	The job is waiting for its requested compute resources to become available
AssociationMaxJobsLimit	The maximum number of running jobs associated to an account has been reached

Once the described requirement has been satisfied, the job will run. Information about other job reason codes are available here: https://slurm.schedmd.com/squeue.html#lbAF

Some reasons for long pending times include:

Low priority score

A common reason for a job to be in the pending state is because it is waiting for higher-priority jobs to run.

The `sprio` command will provide information about the job's place in the queue:

```
$ sprio -p standard
JOBID PARTITION   PRIORITY       SITE        AGE  FAIRSHARE    JOBSIZE  PARTITION
   67148203 standard       11941          0         20       9953        969       1000
   67148210 standard        1999          0         17          2        981       1000
   67148211 standard        1989          0         16          2        971       1000
   67148213 standard        1998          0         16          2        981       1000
```

The number of jobs that have higher priority along with their *JOBSIZE* will give a sense of how long the job currently needs to wait.

It may be possible to jump the queue by leveraging the backfill scheduling algorithm. This type of scheduler will run a lower priority job if it can finish before higher priority jobs. So, if it's possible to request a shorter wallclock time and fewer resources for the job, the scheduler might be able to run the job before the higher-priority jobs.

Requesting too much resources

It becomes difficult to satisfy large resource requests when the cluster is under heavy use. As a result, large jobs will experience long wait times. So, if possible, it is advantageous to reduce the amount of requested resources.

Job never starts

If a job is waiting for resources to become available, even when there exists no other job on the partition, it is possible that the requested wallclock time is to blame.

Validate if the requested wallclock time is less than the wallclock time the usage account associated to the job is limited to. The usage account's wallclock limit can be listed using this command: `sacctmgr -s show user name=$USER`. If it is larger, then the job will never run. Delete it with `scancel`, and re-submit it with a wallclock time less than maximum allowed for the given usage account.

Job failed

A job will enter the `FAILED` state if it returns a non-zero exit code during execution.

The following command outputs detailed information about a job:

```
$ scontrol show job <jobid>
```

which includes a summary of the requested resources, the amount of used resources, the exit code value, and more.

Some causes include:

- the job ran out of resources (e.g. out of memory)
- invalid output or error filepaths
- a bug was encountered during execution

Other error job states

Jobs can enter a few different error states whereby the job will be terminated.

Here are some error states and how to react:

Error State	Code	Description	Response

TIMEOUT	TO	The job was terminated upon reaching its requested wallclock time	Re-submit it with a longer wallclock time
NODE_FAIL	NF	One or more allocated nodes has had a techinical failure	Contact a support representative
BOOT_FAIL	BF	One or more allocated nodes failed to launch the job	Contact a support representative

FAQ

What is the cluster status and its specification?

```
$ sinfo --Node --long
# or
$ scontrol show nodes
```

`sinfo` provides a summary of each node. S:C:T column means: Allowed Allocating Nodes, Number of CPUs, State of Node.

A partition defines a set of resources (similar to a queue in SGE or PBS).

`scontrol` goes into more detail about each node.

How to log into a running job?

```
$ srun --jobid=<jobid> --pty bash -i
```

How to monitor my job?

```
$ sacct -j <jobid>
$ sacct -u <username> --format Account,UserCPU,ReqMem,AllocTRES%60
# to get the list of format options
$ sacct -e
```

Most of the time, sacct will provide the needed information. More information can be gathered with:

```
$ sstat  # For job step information
$ squeue  # For partition information
```

When will my job start running?

```
$ squeue --start -u $USER
```

Indices and tables

- **genindex**
- **modindex**
- **search**