High-Performance Mathematics

Beware, we are doing science!

Progetto Speciale per la Didattica 2023/24

Fabio Durastante (L5)

May 29, 2024

Dipartimento
di Matematica
Università di Pisa

Table of Contents

Queue Management on Clusters

1 Queue manager

- **Resource Allocation**:
  - Efficiently allocate computational resources (CPUs, memory, etc.) among **multiple users** and **jobs**.
- **Job Scheduling**:
  - **Prioritize** and **schedule jobs** to **optimize cluster utilization** and **minimize wait times**.
- **User Fairness**:
  - Ensure *fair distribution of resources* among users, preventing monopolization.
- **Job Management**:
  - Handle job submission, monitoring, and termination.
- **Scalability**:
  - **Manage** a **large number of jobs** and resources in a scalable manner.

Queue Management on Clusters

1 Queue manager

- **Resource Allocation**:
  - Efficiently allocate computational resources (CPUs, memory, etc.) among **multiple users** and **jobs**.
- **Job Scheduling**:
  - **Prioritize** and **schedule jobs** to **optimize cluster utilization** and **minimize wait times**.
- **User Fairness**:
  - Ensure *fair distribution of resources* among users, preventing monopolization.
- **Job Management**:
  - Handle job submission, monitoring, and termination.
- **Scalability**:
  - **Manage** a **large number of jobs** and resources in a scalable manner.

Challenges Addressed by Queue Managers

A good **queue manager** must be able to *solve a certain number of problems*:

- **Resource Contention**:
  - Multiple users competing for limited resources.
- **Job Dependencies**:
  - Managing jobs that depend on the completion of others.
- **Load Balancing**:
  - Distributing jobs evenly to prevent some nodes from being overburdened while others are idle.
- **Fault Tolerance**:
  - Ensuring job completion despite hardware failures or interruptions.
- **Policy Enforcement**:
  - Implementing organizational policies regarding resource usage and job priorities.

Models based on **Markov chains** are usually used.

SLURM

1 Queue manager

Of the various managers available we—and many super computing centers around the world—make use of SLURM.

- **What is SLURM?**
  - **S**imple **L**inux **U**tility for **R**esource **M**anagement (SLURM)
  - An open-source job scheduler designed for Linux clusters.
- **Components**:
  - **Slurmctld**: Central management daemon.
  - **Slurmd**: Daemon running on each compute node.
  - **Slurmdbd**: Optional database daemon for job accounting.

SLURM

1 Queue manager

Of the various managers available we—and many super computing centers around the world—make use of SLURM.

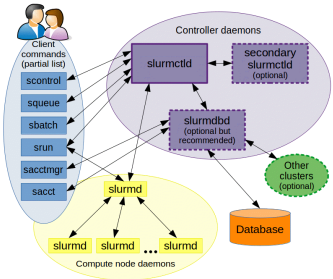
- **What is SLURM?**
  - **S**imple **L**inux **U**tility for **R**esource **M**anagement (SLURM)
  - An open-source job scheduler designed for Linux clusters.
- **Components**:
  - **Slurmctld**: Central management daemon.
  - **Slurmd**: Daemon running on each compute node.
  - **Slurmdbd**: Optional database daemon for job accounting.

🔧 The **basic principles** regarding the use of a queue manager **are always the same**, so the information we will see is portable—as long as you change the name of some commands—even on other systems.

How SLURM Works: the architecture

1 Queue manager

- **Resource Allocation**:
  - Nodes are divided into partitions, each potentially with different characteristics and purposes.
- **Job Submission**:
  - Users submit jobs using `sbatch`, `srun`, or `salloc` commands.
- **Scheduling**:
  - SLURM schedules jobs based on priorities, resource requirements, and availability.
- **Job Execution**:
  - Allocates resources and starts jobs on the appropriate compute nodes.



- **Monitoring and Management**:
  - Provides tools to monitor job status (`squeue`, `scontrol`) and manage jobs (`scancel`).

SLURM Interactive Jobs

1 Queue manager

What are Interactive Jobs?

- Interactive jobs allow users to interact with the job in real-time.
- Useful for debugging, development, and interactive data analysis.

Submitting an Interactive Job

- Use the `srun` command to start an interactive session:
  - `srun --pty bash -i`
- Specify resource requirements as needed:
  - `srun --pty -N1 -n4 --mem=4G --time=01:00:00 bash -i`

Benefits of Interactive Jobs

- Immediate feedback and interaction with the job.
- Easier to troubleshoot and test code on the cluster.
- Allows running exploratory data analysis interactively.

SLURM flags and options

1 Queue manager

`-J`	`--job-name=<jobname>`	**Specify a name** for the job allocation. The specified name will appear along with the job id number when querying running jobs on the system.
`-N`	`--nodes=<minnodes>`	Request that a minimum of minnodes nodes be allocated to this job. A maximum node count may also be specified with maxnodes. If only one number is specified, this is used as both the minimum and maximum node count.
`-n`	`--ntasks=<number>`	Number of **MPI tasks** to be **allocated**.
`-t`	`--time=<time>`	Set a minimum time limit on the job allocation, format is: `d-hh:mm:ss`.
	`--mem=<size>[units]`	Specify the real memory required per node. Default units are megabytes. Different units can be specified using the suffix [K|M|G|T].

An example on steffe

1 Queue manager

Let’s take an example where we **use an interactive job to compile some code**.

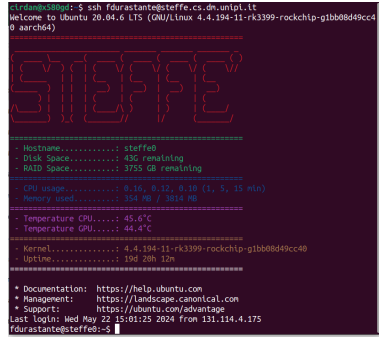
An example on steffe

1 Queue manager

Let’s take an example where we **use an interactive job to compile some code**.

- Connect via `ssh` to `steffe`:

  ```
  ssh fdurastante@steffe.cs.dm.unipi.it
  ```



An example on steffe

1 Queue manager

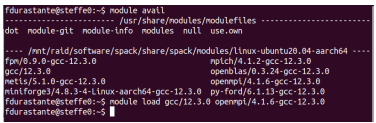
Let's take an example where we **use an interactive job to compile some code**.

- Connect via `ssh` to `steffe`:

```
ssh fdurastante@steffe.cs.dm.unipi.it
```

- Load the relevant `modules`:

```
module avail
module load gcc/12.3.0 openmpi/4.1.6-gcc-12.3.0
```



An example on steffe

1 Queue manager

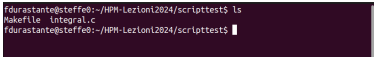
Let's take an example where we **use an interactive job to compile some code**.

</> Connect via `ssh` to `steffe`:

```
ssh fdurastante@steffe.cs.dm.unipi.it
```

</> Load the relevant `modules`:

```
module avail
module load gcc/12.3.0 openmpi/4.1.6-gcc-12.3.0
```



git Clone the example

```
git clone git@git.phc.dm.unipi.it:HighPerfor⌋
↪  manceMathematics/HPM-Lezioni2024.git
cd HPM-Lezioni2024/scripttest
ls
```

An example on steffe

1 Queue manager

We can now **take a node** to compile our code

```
srun --job-name=compile -N1 --time=00:10:00 --pty bash -i
```

by doing so we requested a node, `-N1` , for 10 minutes with an interactive shell.

An example on steffe

1 Queue manager

- We can now **take a node** to compile our code

```
srun --job-name=compile -N1 --time=00:10:00 --pty bash -i
```

by doing so we requested a node, `-N1` , for 10 minutes with an interactive shell.

- We can then use the `Makefile` to compile our example code (which computes an integral using the **trapezoid formula**):

```
make
```

- If everything went well, we should read:

```
mpicc integral.c -o integral -lm
```

that has created the executable file `integral`

```
ls
integral  integral.c  Makefile
```

Batched jobs

1 Queue manager

🕓 In general a simulation that requires the use of a cluster will be something that will run for quite some time and **we don't want to stay connected and watch the execution**, especially running the risk of our connection dropping causing the job to fail…

☰ To this end we want to have a way to **queue a job** and have any outputs saved to a file.

Batched jobs

1 Queue manager

🕓 In general a simulation that requires the use of a cluster will be something that will run for quite some time and **we don't want to stay connected and watch the execution**, especially running the risk of our connection dropping causing the job to fail…

☰ To this end we want to have a way to **queue a job** and have any outputs saved to a file.

This is achieved in **2 steps**:

1. Create a `sh` script containing the instruction to be executed, e.g., `run.sh`,
2. Pass the script to slurm with the `sbatch` command:

   ```
   sbatch run.sh
   ```

An example

1 Queue manager

File `run.sh`

```
#!/bin/bash
#SBATCH --partition=production
#SBATCH -N 1
#SBATCH -n 6
#SBATCH -o integral.out

module load gcc/12.3.0
↪  openmpi/4.1.6-gcc-12.3.0

srun ./integral 6000
```

That can then be put into the queue with the command

```
sbatch run.sh
```

- lines beginning with with `#SBATCH` are commands that we need to pass to slurm,
- we always take care to reload the environment with the command `module`,
- the `srun` command is used to launch the executable and takes as task number all those made available by the `-n` option.

And we can see what is queued with the command

```
squeue
```

Options

1 Queue manager

We can use **all the options** we have seen for the **interactive jobs**.

`-o`	`--output`	Specifies the file where standard output is directed.
`-e`	`--error`	Specifies the file where standard error is directed.
`-N`	`--nodes`	Requests a specific number of nodes.
`-n`	`--ntasks`	Specifies the number of tasks to run.
`-c`	`--cpus-per-task`	Defines the number of CPUs to allocate per task.
`-p`	`--partition`	Specifies the partition or queue where the job will be submitted.
	`--ntasks-per-node`	Specifies the number of MPI tasks to run on each node.

Further information are available at: slurm.schedmd.com/sbatch.html.

Exercise

1 Queue manager

Do a **weak** and **strong scalability** test of the code for computing the integral.

Weak scaling how the solution time varies with the number of processors for a fixed problem size per processor.

Strong scaling how the solution time varies with the number of processors for a fixed total problem size.

Some *useful tools*:

</> Syntax for `for` loops in `bash`:

```
for i in {1..5}
do
   echo "Welcome $i times"
done
```

</> Output redirect in `bash`:

```
./command 2>&1 >> output.txt
```

</> Input from *fictitious* file `bash`:

```
./command << EOF
file line 1
file line 2
EOF
```

Table of Contents

2 An example with PSCToolkit

Scalable sparse matrix-vector products

2 An example with PSCToolkit

The next example I want to show you is a test of the sparse matrix-vector product using `PSCToolkit`.

Scalable sparse matrix-vector products

2 An example with PSCToolkit

The next example I want to show you is a test of the sparse matrix-vector product using `PSCToolkit`.

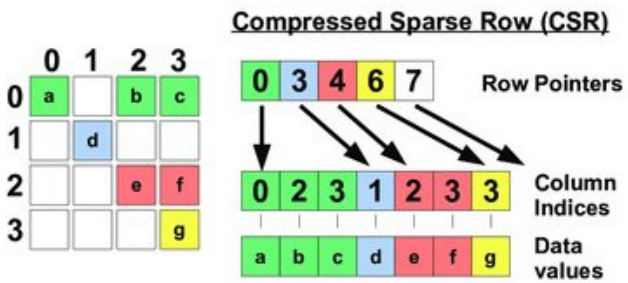
The **C**ompressed **S**parse **R**ow (CSR) format stores a sparse $m \times n$ matrix M in **row form** using three (one-dimensional) arrays (`V`, `COL_INDEX`, `ROW_INDEX`). If we let `NNZ` denote the number of nonzero entries in M.

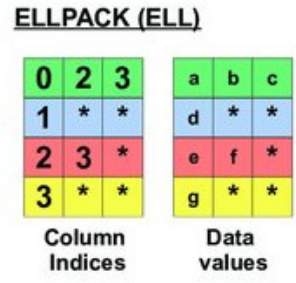
- The arrays `V` and `COL_INDEX` are of length `NNZ`, and contain the non-zero values and the column indices of those values respectively
- `COL_INDEX` contains the column in which the corresponding entry `V` is located.
- The array `ROW_INDEX` is of length $m + 1$ and encodes the index in `V` and `COL_INDEX` where the given row starts. This is equivalent to `ROW_INDEX[j]` encoding the total number of nonzeros above row `j`. The last element is `NNZ`,i.e., the fictitious index in `V` immediately after the last valid index `NNZ - 1`.

Scalable sparse matrix-vector products

2 An example with PSCToolkit

The next example I want to show you is a test of the sparse matrix-vector product using `PSCToolkit`.





Scalable sparse matrix-vector products

2 An example with PSCToolkit

In the `HPM-Lezioni2024` folder the folder `psctoolkitest` contains an example from the `PSBLAS` library doing repeated matrix-vector products to measure performances.

Scalable sparse matrix-vector products

2 An example with PSCToolkit

In the `HPM-Lezioni2024` folder the folder `psctoolkitest` contains an example from the `PSBLAS` library doing repeated matrix-vector products to measure performances.

First thing we have to load the right modules:

```
module load gcc/12.3.0 openmpi/4.1.6-gcc-12.3.0
↪  openblas/0.3.24-gcc-12.3.0 metis/5.1.0-gcc-12.3.0
```

Scalable sparse matrix-vector products

2 An example with PSCToolkit

In the `HPM-Lezioni2024` folder the folder `psctoolkitest` contains an example from the `PSBLAS` library doing repeated matrix-vector products to measure performances.

- First thing we have to load the right modules:

```
module load gcc/12.3.0 openmpi/4.1.6-gcc-12.3.0
↪  openblas/0.3.24-gcc-12.3.0 metis/5.1.0-gcc-12.3.0
```

- Then we can compile the example by typing `make`.

Scalable sparse matrix-vector products

2 An example with PSCToolkit

In the `HPM-Lezioni2024` folder the folder `psctoolkitest` contains an example from the `PSBLAS` library doing repeated matrix-vector products to measure performances.

- First thing we have to load the right modules:

```
module load gcc/12.3.0 openmpi/4.1.6-gcc-12.3.0
↪  openblas/0.3.24-gcc-12.3.0 metis/5.1.0-gcc-12.3.0
```

- Then we can compile the example by typing `make`.
- Now the folder `runs` contains the executable `pdgenspmv`.

Scalable sparse matrix-vector products

2 An example with PSCToolkit

In the `HPM-Lezioni2024` folder the folder `psctoolkitest` contains an example from the `PSBLAS` library doing repeated matrix-vector products to measure performances.

- First thing we have to load the right modules:

```
module load gcc/12.3.0 openmpi/4.1.6-gcc-12.3.0
↪  openblas/0.3.24-gcc-12.3.0 metis/5.1.0-gcc-12.3.0
```

- Then we can compile the example by typing `make`.
- Now the folder `runs` contains the executable `pdgenspmv`.
- The folder contains a batch file called `run.sh` that can be used to test the program.

Scalable sparse matrix-vector products

2 An example with PSCToolkit

In the `HPM-Lezioni2024` folder the folder `psctoolkitest` contains an example from the `PSBLAS` library doing repeated matrix-vector products to measure performances.

```
#!/bin/bash
#SBATCH --ntasks=30
#SBATCH --partition=production
#SBATCH --time 10:00:00
#SBATCH --job-name=psct

srun ./pdgenspmv 2>&1 >> logfiles.txt <<EOF
CSR
200
EOF
```

The script takes two inputs, the matrix format `CSR` and the size of the test matrix `200` (size is actually n^3).

Scalable sparse matrix-vector products

2 An example with PSCToolkit

The output looks something like this:

```
Test on                          :                  30 processors
Size of matrix                   :             8000000
Number of nonzeros               :            55760000
Memory occupation                :           701120360
Number of flops (20 prod)        :          2230400000.
Time for 20 products (s)         :                1.531
Time per product    (ms)         :               76.555
MFLOPS                           :             1456.722
Time for 20 products (s) (trans.):                2.035
Time per product    (ms) (trans.):              101.769
MFLOPS                   (trans.):             1095.814

MBYTES/S                         :            10830.318
MBYTES/S                 (trans):             8147.072
Storage type for DESC_A: HASH
Total memory occupation for DESC_A:    190555968
```

Table of Contents

3 Libraries with support for parallelism

Libraries with support for parallelism

		MPI	GPU	OpenMP
PETSc	PETSc, the Portable, Extensible Toolkit for Scientific Computation, pronounced PET-see (/ˈpɛt-siː/), is for the scalable (parallel) solution of scientific applications modeled by partial differential equations (PDEs). It has bindings for C, Fortran, and Python (via petsc4py).	✔	✔	part.
FEniCS	FEniCS is a popular open-source computing platform for solving partial differential equations (PDEs) with the finite element method (FEM). FEniCS enables users to quickly translate scientific models into efficient finite element code.	✔	part.	part.
deal.II	A C++ software library supporting the creation of finite element codes and an open community of users and developers.	✔	✔	✔

Libraries with support for parallelism

		MPI	GPU	OpenMP
MFEM	MFEM is a free, lightweight, scalable C++ library for finite element methods.	✔	✔	✔
GADGET-4	Is a parallel cosmological N-body and SPH code meant for simulations of cosmic structure formation and calculations relevant for galaxy evolution and galactic dynamics.	✔		✔
Quantum ESPRESSO	Quantum ESPRESSO is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.	✔	✔	✔

Libraries with support for parallelism

		MPI	GPU	OpenMP
ParFlow	ParFlow is a numerical model that simulates the hydrologic cycle from the bedrock to the top of the plant canopy. It integrates three-dimensional groundwater flow with overland flow and plant processes using physically-based equations to rigorously simulate fluxes of water and energy in complex real-world systems.	✔	✔	✔
SUNDIALS	SUNDIALS is a SUite of Nonlinear and DIfferential/ALgebraic equation Solvers.	✔	✔	✔
FFTW	FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data).	✔		✔

Libraries with support for parallelism

		MPI	GPU	OpenMP
KRATOS	KRATOS Multiphysics (”Kratos”) is a framework for building parallel, multi-disciplinary simulation software, aiming at modularity, extensibility, and high performance. Kratos is written in C++, and counts with an extensive Python interface.	✔		✔

To find **other interesting projects**:

- The GitHub MPI Topic list and CUDA Topic list.