HPC for dummies

This document present in general what are computing clusters and what is High Performance Computing (HPC). It can be read out of curiosity, if you want to know what are those infrastructures for or if you are a potential future user with limited technical background. For more practical information on using unige HPC infrastructure, refer to the rest of the documentation.

Documentation : root of the unige's HPC documentation.

When doing their job, researchers often needs to use non interactive computer programs that runs for various amount of time and use various amount of memory. Those programs are like functions, you give them an input (data, parameters), they run for some times and gives you an output (data, text output). They can range from small scripts and programs running in a fraction of second to very long tasks requiring days of computation.

When running heavy computations, you can face multiple challenges. If one of your tasks takes several days to complete, you can let it run on your lab's computer by taking care of not turning it off during that time. And now what if you have several of those tasks to run at the same time ? You can do it if the number of tasks is small, while modern computers comes with multiple computing cores (typically 4 to 8) allowing to run several tasks at the same time.

But what if you have to run hundreds or thousands of those tasks ? Then the idea is simple : use a lot of computers at the same time. But having lots of computers available is not enough, you need a way to manage them in a centralized way. Otherwise, you would have to connect individually to each computer, run some tasks, wait for completion, and manually gather the results. And still, what happens if you want to share the resources with other people ? You would have to establish some kind of usage schedule. If you want to use hundreds of computers or more, manual management of tasks is simply not an option.

That's where computing clusters comes into play. They are quite literally clusters of computers, interconnected by a network, with a centralized storage and a central tasks (or jobs) management software.



Other technologies exists to run tasks on demand on multiple computers, such as cloud computing or grid/distributed computing, but they each comes with their own advantages and limitations. We can cite a lack of very high performance network needed for large scale parallel computation (we'll come to that later) and lack of high availability and security concerns for distributed computing.

First clusters where made with commodity hardware and named “Beowulf” clusters and compound of standard computers on which computations runs (the computing nodes), a head node (or login node or front end) from which users interact with the system and a file server accessible by all nodes (computing nodes and login node). Modern High Performance Computing (HPC) clusters relies essentially on the same architecture and software stack with more high end and specialized hardware. Nodes runs under some Linux distribution and a task scheduling tool (slurm in unige's case) is available.

Documentation : see documentation on how to use slurm on unige's clusters for more informations

on the queuing system.

Clusters hardware and characteristics

A computer is mainly compound of CPU(s), RAM, coprocessor(s) and a network interface. A CPU (Central Processing Unit) is the main processor of a computer. It is used to run the operating system and users programs. Each CPU can embed multiple processing cores (usually 2 to 4 in standard laptops and computers, up to tens of cores in server CPUs) and each computer can embed one or more CPUs (one in commodity computers, up to two in servers). This allows running multiple programs at the same time, or run programs using multiple cores in parallel (see below).

RAM (Random Access Memory) is a fast memory where program's data are stored during computation. Coprocessors are specific hardware specialized in some types of computation. The best known types of coprocessors are currently GPU (Graphics Processing Units). They where originally designed to perform graphic rendering tasks, but they prove to be very efficient for other types of problems, such as machine learning, and are now commonly used for general purpose computing. The network interface is used to exchange data with other computers.

Documentation : the glossary gives the meaning of some terms.

Each node in a cluster is a computer, embedding one or more multicore CPUs, a certain amount of RAM, one or more network interfaces and possibly one or more coprocessors (usually GPUs). Thus, a cluster can be characterized by its number of nodes, quantity of RAM, type and number of CPU cores and GPUs and network bandwith. Some cluster are homogeneous (every node has the same configuration) while others are heterogeneous (there are nodes with different configurations).

Another very important part of a cluster is its storage. Indeed, software running on a cluster needs to access data to process. In case of clusters, data are stored as close as possible to the compute nodes in storage servers or local storage rather than in some distant server or service, such as cloud storage.

Storage can mainly be subdivided in two categories : local and shared storage. Local storage is the storage of each individual compute node. Its fast, but rather small and not shared. Each node can only access its own local storage. It's thus generally used for temporary file useful only during computation rather than to store input or output data. Shared storage on the other hand is slower, larger and shared among all compute nodes and login node. This is where the home directory of users is stored, and where input and output data (results) can be stored. It is accessible from the outside of the cluster through the login node. Shared storage is materialized by storage nodes (or storage servers) physically located in the same server room as the compute nodes, that are accessible through the network by compute and login nodes.

Most powerful machines in the world are ranked in a list called the top500 and can have up to several millions of computing cores. Current unige's clusters Baobab and Yggdrasil have respectively ~4200 cores, a 40Gigabit/s network and 95 GPUs and ~4300 cores, a 100 Gigabit/s network and 52 GPUs.

External link : the top500 list.

Documentation : see here for more detailed informations on unige's clusters specifications.

Clusters can range from small lab infrastructure to the most powerful supercomputers in the world,

but they all use the same fundamental principle of interconnected computing nodes and the same type of software technologies.





Back views of the Yggdrasil cluster of the University of Geneva during assembly. You can see the nodes stacked onto each other.

Parallel computing and HPC

For now, we have talked about programs running on one computing core, and running a lot of them at the same time. While this is a common use case for clusters, this is not what they where designed for. Where they really shine is for parallel computing, and this is when dealing with large problems that we can really start talking about High Performance Computing.

Parallel computing is when we use multiple processing units at the same time to work on one given

problem. Let's say you want to simulate the weather of Switzerland in the following 24 hours to determine if you can have a picnic tomorrow. Atmosphere will be split in chunks of a certain size (the spatial resolution, the smaller the better). Each of those chunks will have it's own characteristics (temperature, humidity, wind speed and direction, ...). Knowing the state of a chunk and it's neighboring chunks, it is possible to compute the state of the chunk a small time step in the future (the time resolution, the smaller the better). By calculating a lot of small time steps for all chunks, one can determine the state of the atmosphere in the future. Suppose you have a program (or model) that does that job for you. You can give it an initial condition (the current weather over all Switzerland), and ask it to advance a lot of small steps in time to get the state of the atmosphere tomorrow to get your answer (can we have this picnic or not ?).

First challenge, if the spatial resolution is small, all of the atmosphere chunks will not fit in the memory of one computer. To address this problem, chunks of atmosphere will be scattered over multiple computers (nodes of a cluster). But remember, to see how one chunk state evolve over time, you need informations about neighboring chunks. When a chunk needs informations about a chunk located in another computer, a network communication is necessary. Actually, network communications will be necessary at each time step, and while we are considering a lot of time steps and chunks, a lot of communications are necessary.

This is the first goal of parallelism : dealing with larger problems. Now your problem fits in the memory of the cluster by spreading the problem over multiple nodes. You start your simulation to know the weather tomorrow at 12 and … get your result after tomorrow at 12. Useful if you want to validate your simulation accuracy, not much if you want to know if you can have your damn picnic ! This is now the next goal of parallelism : going faster.

Computing the state of an atmosphere chunk for the next time step takes time. Each node has to deal with a certain amount of chunks. So if you lower the number of chunks per node (and thus increase the total number of used nodes), the time per iteration will be lower. But there is a caveat. The more you scatter your problem over many nodes, the more network communications will be needed. And here you really need an HPC cluster to deal with this situation. Without very high bandwidth and a good network architecture, you could even end with a longer overall computing time because network communications takes too much time. Finding the balance between efficiency and computing time, writing efficient software and designing efficient hardware are actually difficult tasks that are at the core of HPC.

Now that you have scattered your problem over more nodes of an HPC cluster, you get the result of your simulation at 6pm after starting it at 12, and you are able to tell if, according to your simulation, your can have your picnic tomorrow. And the answer is yes ! And tomorrow at 12… it rains ! Well… your need a better model…

Then, you decide to run simulations with five different weather models you have access to. You decide to simulate every day of the past week for each five models and compare the results with observed weather to see which model best reproduce the real weather. You can do all those computation in only one afternoon because each model is parallelized, and you have enough compute nodes to run all simulations at the same time. This is the last goal of parallelism : dealing with larger problems in a given amount of time. Now you are able to choose the best simulation tool for your situation and have the best possible weather estimation for your next picnic.

A very widely used standard in HPC to exchange data between separate parts of a program (or processes) is MPI (Message Passing Interface). It is beyond the scope of this document to present MPI more thoroughly.

External link : mpitutorial is a nice reference if you are curious and want more informations about MPI and the technics involved in designing large scale parallel programs.

Using a cluster

Now you understand what those big machines are and why they are useful. But what does it actually looks like to use one of those ?

Let's recall the main bricks we are dealing with :

- The head node, or login node.
- Some compute nodes.
- A shared storage, accessible from all nodes.
- A management system that decides when and on which nodes programs are run (the queuing system).

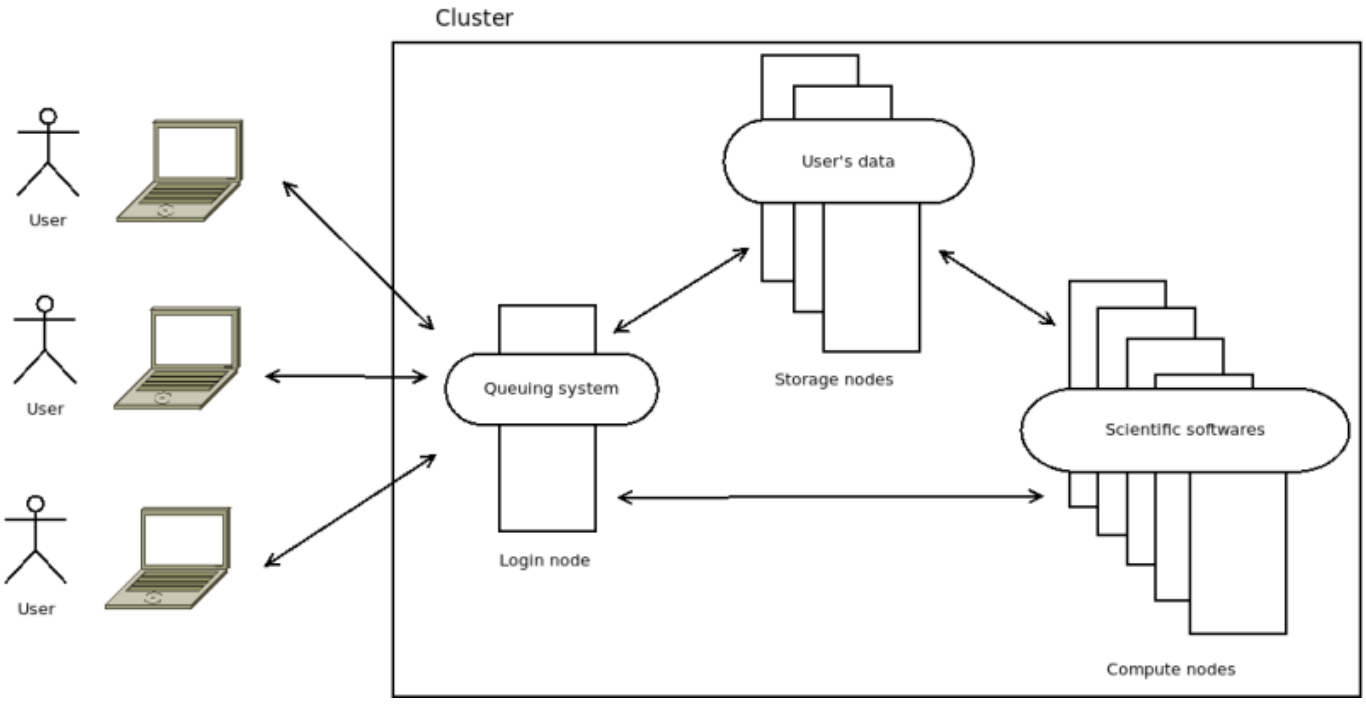
The typical use case consist of :

- Sending data to the cluster storage.
- Asking the queuing system to run a certain program using certain resources on the data.
- Get back the data after the program has been run.

There are three important things to keep in mind here :

- Clusters are running Linux, which heavily relies on a command line interface. You will be able to perform some tasks with a graphical interface, but at some point you will have to use a command line interface.
- There are many users using the cluster at the same time. So you have no guarantee your computations will start immediately.
- You will not directly run your program on the cluster as you do on your personnal computer. You will ask the queueing system to run a program, and once resources are available, the queuing system will start the program.
- HPC clusters where not designed for interactive tasks. While doable, they are much better candidates for asynchronous computing (without user interaction).

As a user, you will interact directly with the login node only. From this computer, you will manage your files, set up your execution configuration and ask the queuing system for computation on the compute nodes. But you will never run your programs directly on the login node.



An example

Even if this document is not a tutorial, let's look at a small concrete example. We will run a python script that double every numbers in a text file and write the result in an output file.

The files we are dealing with are `data.txt`, which contains a list of numbers :

```
1 4 6
```

`multiply.py`, a python script (a program) that reads this list and output the result in a file name `output.txt` :

```
with open('data.txt') as fin:
    data = fin.read().split()

data = [2*int(x) for x in data]

with open('output.txt', 'w') as fout:
    fout.write(' '.join(str(x) for x in data))
    fout.write('\n')
```

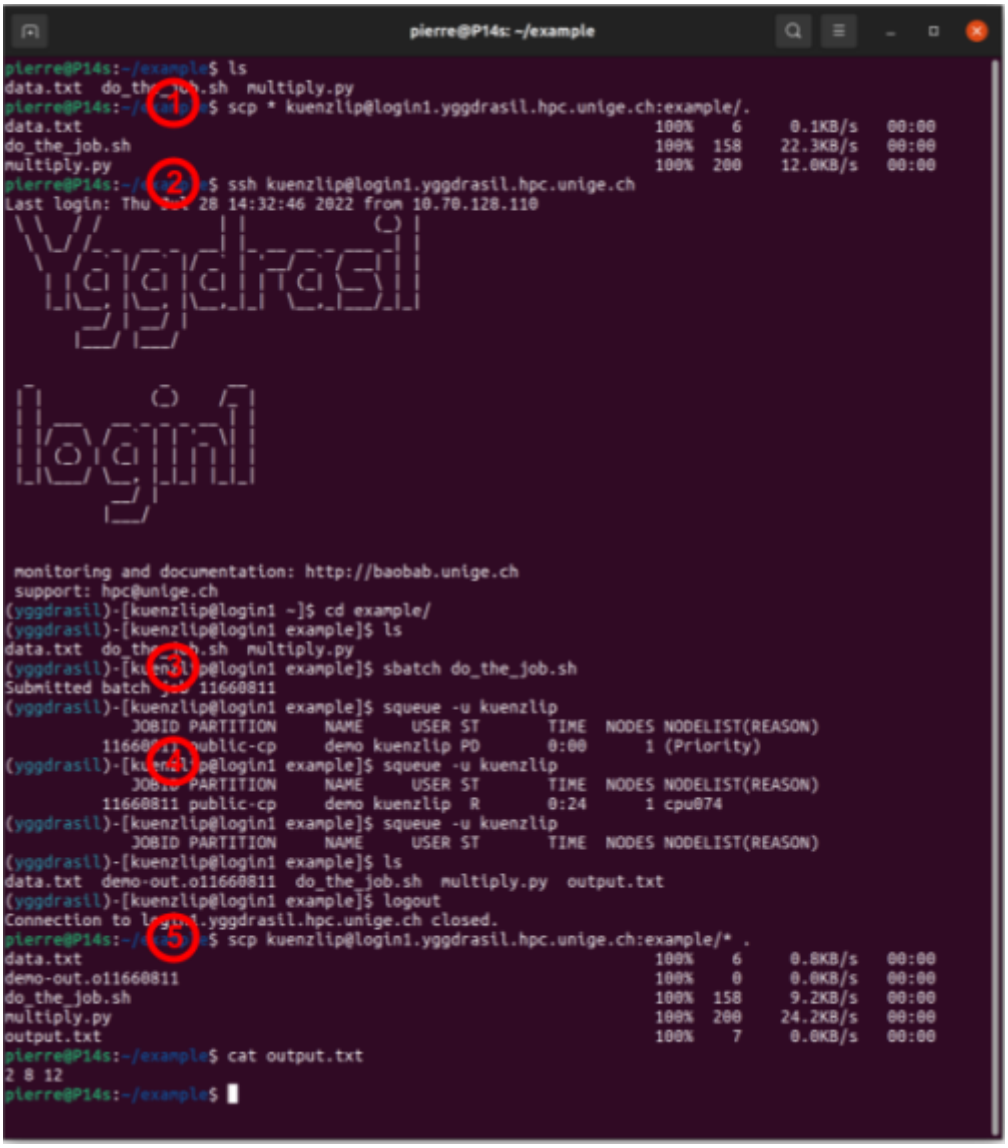
and finally `do_the_job.sh` which is a file in which we describe the job to be done, which will be given to the queuing system :

```
#!/bin/sh
#SBATCH --job-name demo
#SBATCH --output demo-out.o%j
#SBATCH --ntasks 1
#SBATCH --partition public-cpu
#SBATCH --time 00:01:00
```

```
python multiply.py
```

With this file, we are telling the queuing system “I want you to run my multiply program on one processor of one of the computing node, for a maximum time of one minute”. Of course, when working with more complex program, you will increase those values (number of computing resources and computing time) according to your needs. You could replace the input data by a large data file or a set of files and the program with any complex simulation tool, the principle will be the same.

So let's do the job. Send those files to the cluster, submit the job to the queuing system and gather results once done.



In this image, you see exactly the steps described above achieved from a linux terminal.

- 1) Data and script are sent to the cluster (first `scp` command).
- 2) Then we connect to the login node (`ssh command`).
- 3) We ask for the computation to be done (`sbatch` command).
- 4) We monitor the state of the job (`squeue` command) which is finally executed on node 74 of the cluster.
- 5) Finally we gather results by downloading all files, include `output.txt` (last `scp` command).

Going further

Now that you have an idea of what HPC clusters are. If you are willing to actually use them, your next steps are :

- Getting familiar with Linux and command line environment.
- Go through the rest of the HPC unige documentation to get used to the local infrastructure.

More particularly, please read the best practices guide which help avoid more common mistakes.

Documentation : best practices guide.

You can as well find help and advice through our forum, FAQ and direct contact with the HPC admin and support team.

Documentation : support and advice using the cluster.

Documentation : some other important parts of the unige HPC documentation :

- The main page of unige HPC documentation
- Some pointers to get started with Linux
- Description and presentation of the local HPC infrastructure
- Connection and file transfer with the clusters
- The slurm queuing system

From:
https://doc.eresearch.unige.ch/ - **eResearch Doc**

Permanent link:
https://doc.eresearch.unige.ch/hpc/getting_started

Last update: **2023/06/09 09:07**