Getting Started Guide to HPMoon Cluster

Contents

1 About *HPMoon*

HPMoon is one of the clusters that belongs to the *Department of Computer Architecture and Technology of University of Granada*. The cluster is located in the CITIC-UGR building, and has been funded with ERDF fund and some national research projects. Currently, it is made up of a total of 10 heterogeneous NUMA nodes, which are interconnected via Gigabit Ethernet and managed with *Rocky Linux 8.4*. Table 1 shows the main components of the cluster, while Tables 2 and 3 detail the features of its CPU and GPU devices, respectively:

Table 1: CPU-GPU devices and wattmeters available in the *HPMoon* cluster.

<table>
<tr><th rowspan="2">Node</th><th colspan="2">CPU</th><th>GPU</th><th colspan="2">Wattmeter</th></tr>
<tr><th>Device</th><th>Total Cores/Threads</th><th>Device</th><th>openZmeter</th><th>Arduino</th></tr>
<tr><td>hpmoon.ugr.es</td><td rowspan="2">2x Intel Xeon E5-2620 v2</td><td rowspan="2">12/24</td><td>1x Nvidia Quadro K2000</td><td rowspan="4">Yes</td><td rowspan="5">Yes</td></tr>
<tr><td>compute-0-0</td><td>2x Nvidia TITAN RTX</td></tr>
<tr><td>compute-0-1</td><td>1x Intel Xeon E5-2620 v4</td><td>8/16</td><td rowspan="2">1x Nvidia Tesla K40m
1x Nvidia TITAN Xp</td></tr>
<tr><td>compute-0-2</td><td>2x Intel Xeon E5-2620 v4</td><td>16/32</td></tr>
<tr><td>compute-0-3</td><td rowspan="6">2x Intel Xeon Silver 4214</td><td rowspan="6">24/48</td><td rowspan="6">1x Nvidia Quadro RTX 6000</td><td>No</td></tr>
<tr><td>compute-0-4</td><td>Yes</td><td rowspan="5">No</td></tr>
<tr><td>compute-0-5</td><td rowspan="4">No</td></tr>
<tr><td>compute-0-6</td></tr>
<tr><td>compute-0-7</td></tr>
<tr><td>compute-0-8</td></tr>
</table>

Table 2: Characteristics of the CPU devices found in the cluster.

Model	ALU		RAM Memory			TDP
	# Cores/Threads	Freq. (MHz)	Type	Size (GB)	Freq. (MHz)	(W)
Intel Xeon E5-2620 v2	6/12	2,100	DDR3	32	1,600	80
Intel Xeon E5-2620 v4	8/16		DDR4		2,133	85
Intel Xeon Silver 4214	12/24	2,200		64	2,933	

Table 3: Characteristics of the GPU devices found in the cluster.

Model	ALU		RAM Memory			TDP
	# SMs/Cores	Freq. (MHz)	Type	Size (GB)	Freq. (MHz)	(W)
Nvidia Quadro K2000	2/384	954	GDDR5	2	4,000	51
Nvidia Tesla K40m	15/2,880	745		12	6,008	250
Nvidia TITAN Xp	30/3,840	1,582	GDDR5X		11,408	
Nvidia TITAN RTX	72/4,608	1,770	GDDR6	24	14,000	280
Nvidia Quadro RTX 6000						295

1.1 How to Access and Use the Cluster

HPMoon can only be accessed: (i) from the internal network of the University of Granada or (ii) through a VPN connection (Spanish link). In both cases, a University account is required. From a Linux shell, `ssh` or `sftp` commands can be used to connect the cluster (specifically to the front-end node, called *hpmoon.ugr.es*). The way to establish the connection with `ssh` is as follows:

```
ssh -o ServerAliveInterval=200 username@hpmoon.ugr.es
```

where the parameter `ServerAliveInterval=200` allows to keep alive the connection by sending null packets every 200 seconds, and *username* is the name of the user account created by the cluster administrator. **Do not use the `-X` option because the graphical interface is not enabled**. After entering the password and establishing the connection, the current working directory will be the `$HOME` folder. Note that the `sudo` command is not available in *HPMoon*. Also, **it is not allowed to run commands directly on the front-end node except those necessary to operate the system** (`ls`, `cd`, `rm`...). Instead, **use the queue manager *Slurm* to send and run jobs on the computing nodes** following the instructions outlined in Section 1.2. Following this line, the `ssh` connections to the computing nodes have also been disabled, so the `htop` command cannot be used to monitor the workload of the nodes. Instead, use either the *Slurm* tools or the *Ganglia Monitoring System*, the latter being accessible from a web browser (VPN required).

There are two groups of users in the cluster: *researchers* and *guests*. The first includes members of the research project, while the second houses students and external collaborating researchers. Users in the *guest* group have three limitations: a disk quota of 5 GB, lower priority and timeout when using computing resources, and access expiration after one year from the moment the account is created. To request the creation of a new user, fill out the form you will find at this link with the required information.

1.2 Slurm Workload Manager: A Queue System

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. As a cluster workload manager, *Slurm* has three key functions:

1. Allocate exclusive and/or non-exclusive access to resources (computing nodes) to users for some duration of time so they can perform work.
2. Provide a framework for starting, executing, and monitoring work on the set of allocated nodes.
3. Arbitrate contention for resources by managing a queue of pending work.

Anyone who wants to use the *HPMoon* cluster for computing must use *Slurm*. This is because most of the experimentation carried out during research tasks involves measurements of time and energy, so the resources of the nodes associated with a user's job must be exclusively available for that job. In this way, the consistency of the data obtained is guaranteed.

There are two commands in *Slurm* for submitting jobs: `srun` and `sbatch`. The first allows interactive executions, so the job can be canceled at any time by pressing `Ctrl-C`. The standard `STDOUT` and `STDERR` outputs are redirected to the user screen, which can be useful for validating test commands or running light tasks. On the contrary, `sbatch` sends a shell script (`.sh`) to execution and ends immediately after the script is successfully transferred to the *Slurm* controller. In this case, by default both `STDOUT` and `STDERR` outputs are directed to a file *slurm-%j.out*, where *%j* represents the ID that *Slurm* has assigned to the job. The scripts specify not only the user's commands to execute but also the requested resources and other considerations through statements of the form `#SBATCH <option>`. Although many of these options are optional, those in charge of requesting resources (CPU, memory, and time) are mandatory. If they are not specified, *Slurm* will assign the default value. See the srun and sbatch documentation for the options supported by each command. Below is an example that uses `srun` to run an application called *Hello_world* with 16 threads/cores:

```
srun -J jobname -p hetero -n 1 -c 16 -o jobname_%j.out -e jobname_%j.err ./Hello_world
```

In contrast, `sbatch` can be used as follows to launch a `.sh` script:

```
sbatch my_script.sh
```

which will run *Hello_world* 10 times and will also notify the user by email when the job ends or fail:

```
#!/bin/bash

#------- SLURM OPTIONS
#SBATCH -J jobname
#SBATCH -p hetero
#SBATCH -n 1
#SBATCH -c 16
#SBATCH -o jobname_%j.out
#SBATCH -e jobname_%j.err
#SBATCH --mail-user=user@example.com
#SBATCH --mail-type=END,FAIL

#------- USER COMMANDS
for i in {1..10}
do
    echo "Iteration %i"
    srun ./Hello_world
done
```

Notice how within the script `srun` is used to launch the *Hello_world* application. Although it is not mandatory to do so, its use is recommended as it provides some advantages: (i) a better reporting of the resource usage: the `sstat` command will provide real-time resource usage for processes that are started with `srun`, and each step (each call to `srun`) will be reported individually in the accounting; (ii) it can be used to setup several instances of a sequential program into a single job, and micro-schedule those programs inside the job allocation. Finally, (iii) for parallel jobs, `srun` will also play the important role of setup the parallel environment by starting as many instances of the program as were requested with the `--ntasks` option (`-n`). In the case of a MPI program, it will also handle the communication between the MPI library and *Slurm*.

On the other hand, special mention must be made of the `-p` option (partition), also referred to as *queue*. In *HPMoon*, four different queues are available: ***hetero***, ***homo***, ***full***, and ***guest***. As each of them covers a different subset of computing nodes, use the one that meets the needs of your job but involving less powerful nodes if possible. Remember that the resources of a node will not be available for other jobs until the current one ends. Moreover, keep in mind that guest users will only be able to submit jobs to the *guest* queue (see Figure 1).

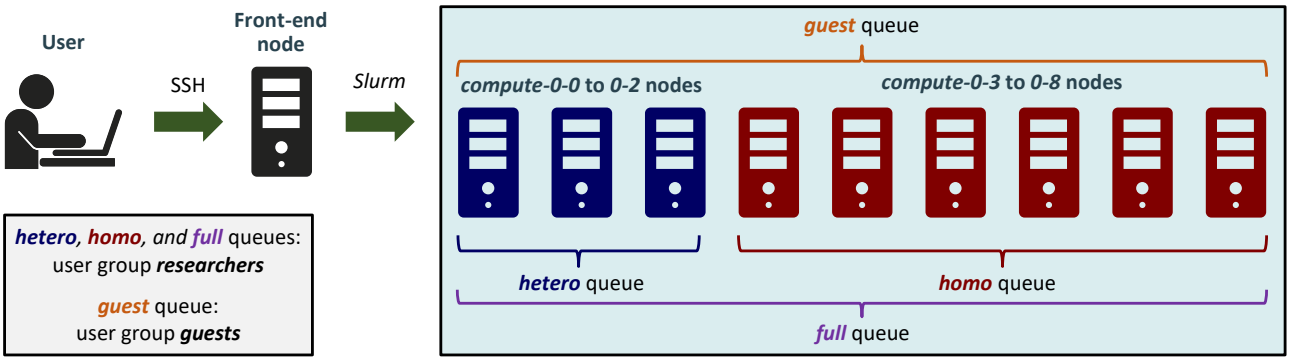


Figure 1: Queues and user groups available in *HPMoon* and what subset of computing nodes they cover.

In addition to `srun` and `sbatch`, *Slurm* provides other commands that are useful for displaying information about jobs, queues, or computing nodes:

- `sinfo`: shows information about partitions (queues) and nodes. An asterisk (*) in the *PARTITION* column indicates the default queue to use if none is specified in `srun` or `sbatch` with the `-p` option.
- `scancel`: removes a job from the queue.
- `squeue`: shows information about the jobs located in the queues. The most frequent states are the following: PD (Pending), R (Running), ST (Stopped), F (Failed), and CO (Completed). The appearance of the TO (Timeout) status is also common, indicating that the job has reached the maximum time allowed for the queue on which it was launched. In *HPMoon*, there are no time restrictions, except for the *guest* queue, which has a timeout of 24 hours. Users launching in this queue can also get the S (Suspended) state, meaning that the execution has been suspended and CPUs have been released for priority user jobs.
- `sstat` and `sacct`: provide users with information on the status of their jobs. This includes information about CPU usage, task information, node information, Resident Set Size (RSS), and Virtual Memory (VM). The difference between the two commands is that `--sstat` displays information for currently running jobs, while `--sacct` displays information for past jobs. The `--format` option allow to filter the information using a comma-separated list of fields. Use `--helpformat` to query the supported fields for each command. Here is an example of how to use them:

  ```
  sstat --jobs=job_ID --format=JobID,AveRSS,MaxRSS,ConsumedEnergy
  sacct --jobs=job_ID --format=JobID,AveRSS,MaxRSS,ElapsedRaw,ConsumedEnergy
  ```

  Special mention to the `ConsumedEnergy` field. From version 2.5 onwards, *Slurm* incorporates a plugin called *acct_gather_energy/rapl* to collect energy data from hardware sensors through the Running Average Power Limit (RAPL) interface. The data provided only includes CPU, DRAM, and cache. The accumulated energy consumption E is reported in Joules (J), while the instantaneous power P is reported in Watts (W). To get the consumption of the GPU devices use the NVIDIA Management Library (NVML). For the entire computing node, take a look at the wattmeters in Section 2.

2 Wattmeters for Energy Measurements

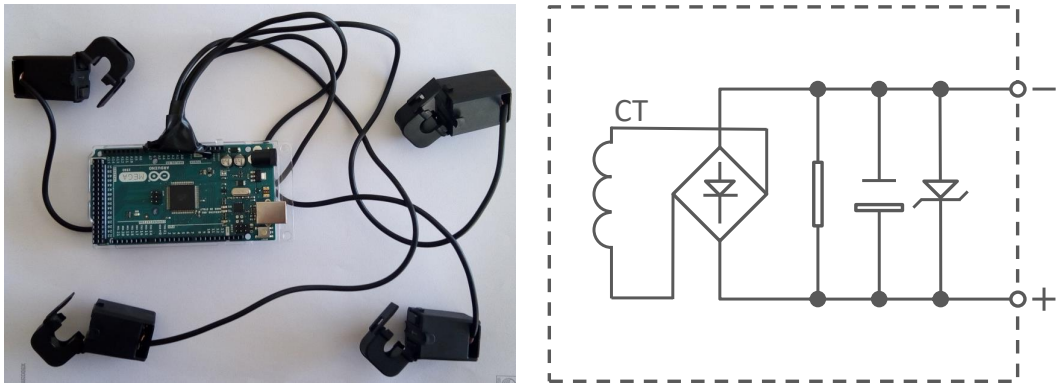
Each node in the cluster has one or more devices to measure energy (see Table 1). The measurements are made by sensors capable of obtaining the electric current flowing through the cable that supply the platform, which allows to determine its real consumption (including all active components as well as the conversion losses of the power supply). Among its different utilities, measuring energy consumption can be used to identify the energy profile of an application and make the necessary improvements according to the observations.

2.1 The *Arduino Mega* Board

It is designed for *Linux* distributions and allows real-time energy measures of multiple nodes. The measurements are made at intervals of one second and determine, for each node, both instantaneous power $P\,(\mathrm{W})$ and accumulated energy consumption $E\,(\mathrm{W}\cdot\mathrm{h})$. Note that unlike RAPL, E is not provided in Joules (J).

2.1.1 Composition

The wattmeter consists of an *Arduino Mega* board and a set of current sensors connected directly to the analog inputs. The power cable of each equipment is placed in a different sensor and the clamp is closed to fix it, detecting the current flowing through the cable. Then, the *Arduino* board is connected to the USB port of the platform, which supplies power and allows data to be transmitted through the serial port created in the */dev/ttyACM0* interface. The *Arduino* obtains four measurements of instantaneous power and accumulated energy consumption per second from each sensor, calculates the average, and transmits the result through the serial port at intervals of one second. Figures 2(a) and 2(b) show the wattmeter and the internal diagram of the current sensors, respectively:



(a) *Arduino Mega* board and current sensors (b) Diagram of the current sensor

Figure 2: Wattmeter based on *Arduino Mega* for energy measurements.

The sensor model used is known as *SCTD010T-5A*, and has been supplied by *YHDC Electronics Co., Ltd.* The sensor is capable of measuring up to 5A and provides an output voltage between 0 and 5V. In addition, its accuracy is around ±2% and the supported working temperature is in the range of −20 to +50° Celsius. The analog signals emitted by the sensors are processed by the *Arduino* through its internal 10-bit analog to digital converter. In addition, to take better advantage of its dynamic range, the internal reference of 2.56V is used, which is only available in the *Mega* model. Although this reduces the measurement to a maximum input of 2.56V, in practice it allows instantaneous power values of up to 588W. Since the platforms present lower values, the reduction of the maximum voltage does not represent any limitation when obtaining the measurements.

2.1.2 Software Description

The software has been developed with *Python* and allows different users to obtain energy measurements independently and simultaneously. As the *Arduino* board sends the information to the serial port through USB,

it is necessary to share the data received between the different users. For this, the *ZeroMQ*[1] message system is used under the *publish-subscribe* pattern. A master process called *master_meter.py* opens the serial port and activates with *ZeroMQ* a data publishing process through TCP port 5214, in which the received data is retransmitted. Each user who wants to receive data executes a process in *Python* named *show_consumption.py* to subscribe to the system, so that data is received from *Arduino* in real-time. Since multiple processes can be connected to the sending system, all receive the information simultaneously. A general scheme of this process can be seen in Figure 3:

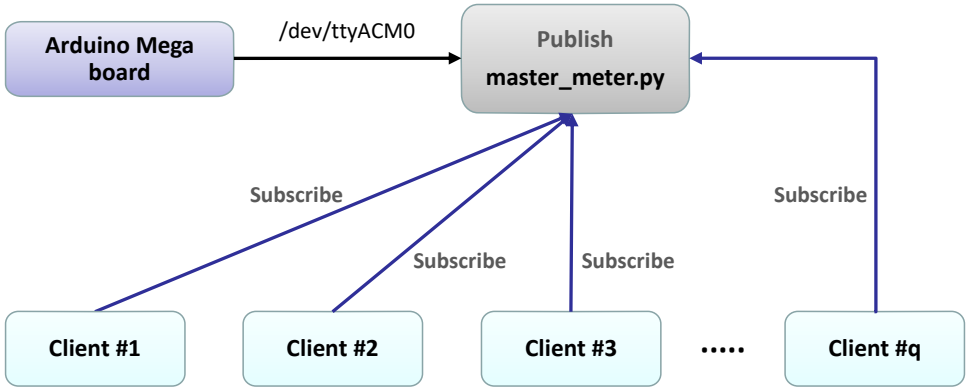


Figure 3: Scheme of the *publish-subscribe* pattern.

In addition, each user has in the `$HOME` directory a file with the accumulated energy consumption after the first call to *show_consumption.py*. In the next call, the user will obtain the difference between the current value and the one stored in the file. However, the values can be set to 0 at any time to re-estimate the accumulated energy for a period of time. To do this, The `R` option of *show_consumption.py* can be used to rewrite the file with the values of that moment. To reset the values and start the measurement with the shortest time interval the following command is used, which produces the output shown in Figure 4:

```
./show_consumption.py R && ./show_consumption.py
```

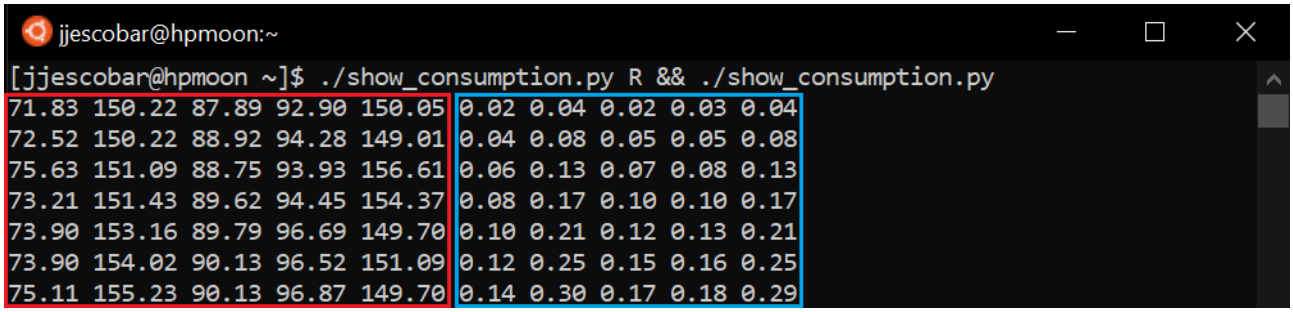


Figure 4: Energy values obtained by the *Arduino Mega* when the cluster nodes are in idle state. The red and blue boxes show the instantaneous power and accumulated energy, respectively. The columns within each color box represent, in this order, the front-end node followed by the computing nodes *compute-0-0* to *compute 0-3*.

2.2 The *openZmeter* (*oZm*)

oZm was born as a collaborative project between the Universities of Granada and Almería to offer an effective tool to obtain accurate electrical measurements and promote awareness in energy-saving. The wattmeter has been designed to comply with international standards IEC 61000-4-30 and EN 50160 and it is able to measure RMS currents with a precision of 1%. Instructions on how to use *oZm* will be added soon.

[1] Asynchronous messaging library used in distributed or concurrent applications.

Acknowledgements

I would like to thank the following people for their contributions and comments, which have been very useful to improve the content of this tutorial:

- Dr. Antonio Francisco Díaz.
- Dr. Mancia Anguita.
- Mr. Francisco Manuel Illeras.
- All *openZmeter* developers from the University of Almería.