Table of Contents

- Utilization and accounting
  - Comparison of sreport, sacct, and sshare
  - Resource accounting uniformization
    - Conversion Rules extract (see below for details)
    - Example Calculation
  - Resources available for research group
  - Job accounting
    - OpenXDMoD
    - sacct
    - Report and statistics with sreport
      - Usage details of a given PI
      - Usage details of all PIs associated with a private group
      - Aggregate usage by all users of a given PI
      - sreport examples

Utilization and accounting

When you submit jobs, they are using physical resources such as CPUs, Memory, Network, GPUs, Energy etc. We keep track of the usage of some of those resource. On this page we'll let you know how to consult your usage of the resource. We have several tools that you can use to consult your utilization: sacct, sreport, openxdmod

Comparison of sreport, sacct, and sshare

We use **sreport** as our primary accounting reference. However, you may find other tools useful for specific purposes. Here's a comparison:

- **sacct**: Displays only account jobs, excluding time allocated via reservations. If duplicate jobs exist, only one is shown.
- **sreport**: By default, jobs with wall times overlapping the report's time range are truncated. For reservation-based jobs, the requested idle time is distributed among all users with access to the reservation.
- **sshare**: Not recommended for accounting purposes; displayed values are adjusted based on fairshare calculations.

Resource accounting uniformization

We apply uniform resource accounting by converting GPU hours and memory usage into CPU-hour equivalents, using the TRESBillingWeights feature provided by SLURM. A CPU hour represents one hour of processing time on a single CPU core.

We use this model because our cluster is heterogeneous, and both the computational power and the cost of GPUs vary significantly depending on the model. To ensure fairness and transparency, each GPU type is assigned a weight that reflects its relative performance compared to a CPU core. Similarly, memory usage is converted into CPU-hour equivalents based on predefined weights.

We also **account for memory usage** because some jobs consume very little CPU but require large amounts of memory, which means an entire compute node is occupied. This ensures that jobs using significant memory resources are accounted for fairly.

Conversion Rules extract (see below for details)

- **1 CPU core = 1 CPUh per hour**
- **1 GB RAM = 0.25 CPUh per hour**
- **1 GPU A100 (40 GB) = 5 CPUh per hour**

Example Calculation

Suppose you request:

- **2 CPUs**
- **20 GB RAM**
- **1 GPU A100**

The cost per hour is calculated as:

- CPU: 2 × 1 CPUh = **2 CPUh**
- RAM: 20 GB × 0.25 CPUh = **5 CPUh**
- GPU: 1 × 5 CPUh = **5 CPUh**

Total per hour = 2 + 5 + 5 = 12 CPUh

This approach guarantees consistent, transparent, and fair resource accounting across all heterogeneous components of the cluster.

You can check the up to date conversion details by inspecting the parameters of any partition on the clusters. The same conversion table is applied on all our clusters and partitions.

```
(bamboo)-[root@slurm1 ~]$ scontrol show partition debug-cpu | grep
TRESBillingWeights | tr "," "\n"
   TRESBillingWeights=CPU=1.0
Mem=0.25G
GRES/gpu=1
GRES/gpu:nvidia_a100-pcie-40gb=5
GRES/gpu:nvidia_a100_80gb_pcie=8
GRES/gpu:nvidia_geforce_rtx_2080_ti=2
GRES/gpu:nvidia_geforce_rtx_3080=3
GRES/gpu:nvidia_geforce_rtx_3090=5
GRES/gpu:nvidia_geforce_rtx_4090=8
GRES/gpu:nvidia_rtx_a5000=5
GRES/gpu:nvidia_rtx_a5500=5
GRES/gpu:nvidia_rtx_a6000=8
GRES/gpu:nvidia_titan_x=1
GRES/gpu:tesla_p100-pcie-12gb=1
```

Here you can see for example that using a gpu nvidia_a100-pcie-40gb for 1 hour is equivalent in term of cost to use 5 CPUhour.

Resources available for research group

Research groups that have invested in the HPC cluster by purchasing private CPU or GPU nodes benefit from **high-priority access** to these resources.

Although these nodes remain available to all users, owners receive **priority scheduling** and a predefined annual allocation of compute hours, referred to as billings. The advantage of this approach is flexibility: you are free to use any resource on any cluster, rather than being restricted to your own nodes. When doing so, your billings will be consumed.

To view details of owned resources, users can run the script: `ug_getNodeCharacteristicsSummary.py` This script provides a summary of the node characteristics within the cluster.

Note: This model ensures **fairness** across all users. Even if some groups own nodes, resources remain shared. Usage beyond the included billings will be **charged according to the standard accounting model**, ensuring equitable access for everyone.

Output example of the script:

```
ug_getNodeCharacteristicsSummary.py --partitions private-<group>-gpu
private-<group>-cpu --cluster <cluster> --summary
host    sn              cpu    mem    gpunumber    gpudeleted  gpumodel
gpumemory  purchasedate      months remaining in prod. (Jan 2025)    billing
------  -----------  -----  -----  -----------  ------------  -------------
------------  -----------  --------------  ---------------------------------
-----  ---------
cpu084  N-20.02.151     36    187           0             0
0  2020-02-01                                            1         79
[...]
cpu088  N-20.02.155     36    187           0             0
0  2020-02-01                                            1         79
[...]
cpu226  N-19.01.161     20     94           0             0
0  2019-01-01                                            0         41
[...]
cpu229  N-19.01.164     20     94           0             0
0  2019-01-01                                            0         41
cpu277  N-20.11.131    128    503           0             0
0  2020-11-01                                           10        251
gpu002  S-16.12.215     12    251           5             0  NVIDIA TITAN X
(Pascal)           12288  2016-12-01
0         84
gpu012  S-16.12.216     24    251           8             0  NVIDIA GeForce
RTX 2080 Ti        11264  2016-12-01
0        108
gpu017  S-20.11.146    128    503           8             0  NVIDIA GeForce
RTX 3090           24576  2020-11-01
```

```
10        299
gpu023  S-21.09.121    128    503          8            0  NVIDIA GeForce
RTX 3080           10240  2021-09-01
20        283
gpu024  S-21.09.122    128    503          8            0  NVIDIA GeForce
RTX 3080           10240  2021-09-01
20        283
gpu044  S-23.01.148    128    503          8            0  NVIDIA RTX
A5000                  24564  2023-01-01
36        299
gpu047  S-23.12.113    128    503          8            0  NVIDIA RTX
A5000                  24564  2023-12-01
47        299
gpu049  S-24.10.140    128    384          8            0  NVIDIA GeForce
RTX 4090           24564  2024-10-01
57        291

============================================================ Summary
============================================================
Total CPUs: 1364 Total CPUs memory[GB]: 6059 Total GPUs: 61 Total GPUs
memory[MB]: 142300 Billing: 1959 CPUhours per year: 10.30M
```

How to read the output:

- **host**: the hostname of the compute node
- **sn**: the serial number of the node
- **cpu**: the number of CPUs available in the node
- **mem**: the quantity of memory on the node in GB
- **gpunumber**: the number of GPU cards on the node
- **gpudeleted**: the number of GPU cards out of order
- **gpumodel**: the GPU model
- **gpumemory**: the GPU memory in MB per GPU card
- **purchasedate**: the purchase date of the node
- **months remaining in prod. (Jan 2025)**: the number of months the node remains the property of the research group, the reference date is indicated in parenthesis. In this example it is January 2025.
- **billing**: the billing value of the compute node

You can modify the reference year if you want to “simulate” the hardware you'll have in your private partition in a given year. To do so, use the argument `--reference-year` of the script.

Job accounting

OpenXDMoD

We track the job usage of our clusters here: https://openxdmod.hpc.unige.ch/

We have a tutorial explaining some of the features: here

Openxdmod is integrated into our SI. When you connect to it, you'll get the profile “user” and the data are filtered by your user by default. If you are a PI, you can ask us to change your profile to be PI.

OpenXDMoD currently supports only CPUh and GPUh metrics, not the billing metrics (yet?). For this reason, you need to use sreport or our script if you want to view the billed metrics.

sacct

You can see your job history using `sacct`:

```
[sagon@master ~] $ sacct -u $USER -S 2021-04-01
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
45517641        jobname  debug-cpu   rossigno          1     FAILED      2:0
45517641.ba+      batch              rossigno          1     FAILED      2:0
45517641.ex+     extern              rossigno          1  COMPLETED      0:0
45517641.0            R              rossigno          1     FAILED      2:0
45518119        jobname  debug-cpu   rossigno          1  COMPLETED      0:0
45518119.ba+      batch              rossigno          1  COMPLETED      0:0
45518119.ex+     extern              rossigno          1  COMPLETED      0:0
```

Report and statistics with sreport

To get reporting about your past jobs, you can use `sreport` (https://slurm.schedmd.com/sreport.html).

We wrote a helper that you can use to get your past resource usage on the cluster. This script can display the resource utilization

- for each user of a given account (PI)
- total usage of a given account (PI)

```
(baobab)-[sagon@login1] $ ug_slurm_usage_per_user.py --help
usage: ug_slurm_usage_per_user.py [-h] [--user USER] [--start START] [--end
END] [--pi PI] [--group GROUP] [--cluster {baobab,yggdrasil,bamboo}] [--all-
users] [--aggregate] [--report-type {user,account}]
                                  [--time-format {Hours,Minutes,Seconds}] [-
-verbose]

Retrieve HPC utilization statistics for a user or group of users.

options:
  -h, --help            show this help message and exit
  --user USER           Username to retrieve usage for.
  --start START         Start date (default: first of month).
  --end END             End date (default: now).
  --pi PI               Specify a PI manually.
  --group GROUP         Specify a group name to get all PIs belonging to it.
  --cluster {baobab,yggdrasil,bamboo}
                        Cluster name (default: all clusters).
  --all-users           Include all users under the PI account.
  --aggregate           Aggregate the usage per user.
  --report-type {user,account}
                        Type of report: user (default) or account.
  --time-format {Hours,Minutes,Seconds}
                        Time format: Hours (default), Minutes, or Seconds.
  --verbose             Verbose output.
```

By default when you run this script, it will print your past usage of the current month, for all the accounts you are member of.

Usage details of a given PI

```
(baobab)-[sagon@login1] $ ug_slurm_usage_per_user.py --pi **** --report-type
account --start 2025-01-01
--------------------------------------------------------------------------
----

Cluster/Account/User Utilization 2025-01-01T00:00:00 - 2025-12-08T13:59:59
(29512800 secs)

Usage reported in TRES Hours

--------------------------------------------------------------------------
----

Cluster    Login    Proper Name    Account    TRES Name      Used
---------  -------  -------------  ---------  -----------  ------
bamboo                             krusek     billing      176681
baobab                             krusek     billing      961209
yggdrasil                          krusek     billing           0
Total usage: 1.14M
```

Usage details of all PIs associated with a private group

Usage example to see the resource usage from the beginning of 2025 for all the PIs and associate users of the group private_xxx. The group private_xxx owns several compute nodes:

```
(baobab)-[sagon@login1 ~]$ ug_slurm_usage_per_user.py --group private_xxx --
start=2025-01-01 --report-type=account
--------------------------------------------------------------------------
----

Cluster/Account/User Utilization 2025-01-01T00:00:00 - 2025-08-21T14:59:59
```

```
(20095200 secs)

Usage reported in TRES Hours

--------------------------------------------------------------------------------
----

Cluster    Login    Proper Name    Account    TRES Name       Used
---------  -------  -------------  ---------  -----------  -------
baobab                             PI1        billing        56134
yggdrasil                          PI1        billing       105817
bamboo                             PI2        billing         5416
baobab                             PI2        billing      1517001
yggdrasil                          PI2        billing        23775
bamboo                             PI3        billing            0
baobab                             PI3        billing      1687963
yggdrasil                          PI3        billing      1344599
[...]
Total usage: 7.36M
```

Aggregate usage by all users of a given PI

```
$ ug_slurm_usage_per_user.py --pi ***** --report-type account --start
2025-01-01 --all-users --aggregate
--------------------------------------------------------------------------------
----

Cluster/Account/User Utilization 2025-01-01T00:00:00 - 2025-12-08T13:59:59
(29512800 secs)

Usage reported in TRES Hours

--------------------------------------------------------------------------------
----

Login       Used
--------  ------
a***u    547746
d***i    272634
d***on    91178
d***l     86860
e***j     60649
v***d0    37962
w***r     29886
s***o      9120
k***k      1853
m***l         1
Total usage: 1.14M
```

sreport examples

by default, the TRES (tracking resource) shown by sreport is CPUh. If you want to see what will be accounted and billed, you need to use the TRES “billing”.

Here are some examples that can give you a starting point :

To get the number of jobs you ran (you ⇔ $USER) in 2018 (dates in yyyy-mm-dd format) :

```
[brero@login2 ~]$ sreport job sizesbyaccount user=$USER PrintJobCount
start=2018-01-01 end=2019-01-01


----------------------------------------------------------------------------
----
Job Sizes 2018-01-01T00:00:00 - 2018-12-31T23:59:59 (31536000 secs)
Units are in number of jobs ran
----------------------------------------------------------------------------
----
  Cluster   Account     0-49 CPUs   50-249 CPUs  250-499 CPUs  500-999 CPUs
>= 1000 CPUs % of cluster
--------- --------- ------------- ------------- ------------- -------------
------------- ------------
   baobab      root           180            40             4            15
0      100.00%
```

You can see how many jobs were run (grouped by allocated CPU). You can also see we specified an extra day for the *end date* `end=2019-01-01` in order to cover the whole year :

```
Job Sizes 2018-01-01T00:00:00 - 2018-12-31T23:59:59''
```

You can also check how much CPU time (seconds) you have used on the cluster between since 2019-09-01 :

```
[brero@login2 ~]$ sreport cluster AccountUtilizationByUser user=$USER
start=2019-09-01 -t Seconds
----------------------------------------------------------------------------
----
Cluster/Account/User Utilization 2019-09-01T00:00:00 - 2019-09-09T23:59:59
(64800 secs)
Usage reported in CPU Seconds
----------------------------------------------------------------------------
----
  Cluster         Account     Login     Proper Name     Used   Energy
--------- --------------- --------- --------------- -------- --------
   baobab        rossigno     brero   BRERO Massimo     1159        0
```

In this example, we added the time `-t Seconds` parameter to have the output in seconds. *Minutes* or *Hours* are also possible.

Please note :

- By default, the CPU time is in Minutes
- It takes up to an hour for Slurm to upate this information in its database, so be patient
- If you don't specify a start, nor an end date, yesterday's date will be used.
- The CPU time is the time that was allocated to you. It doesn't matter if the CPU was actually used or not. So let's say you ask for 15min allocation, then do nothing for 3 minutes then run 1 CPU at 100% for 4 minutes and exit the allocation, then 7 minutes will be added to your CPU time.

Tip : If you absolutely need a report including your job that ran on the same day, you can override the default end date by forcing tomorrow's date :

```
sreport cluster AccountUtilizationByUser user=$USER start=2019-09-01
end=$(date --date="tomorrow" +%Y-%m-%d) -t seconds
```