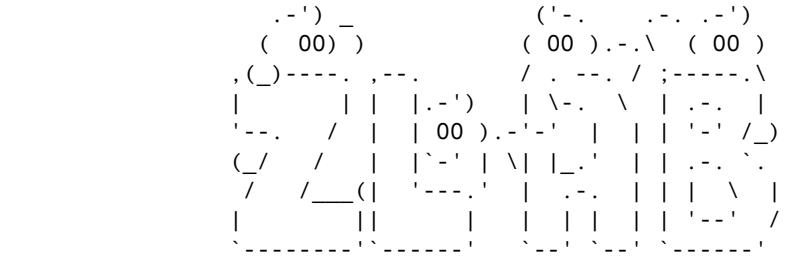
ZLab Server Usage Guidelines

A detailed guide to the usage of the ZLab server resources

Michael Purcaro, Arjan van der Velde, Christian Ramirez



2025-10-11

Welcome to the ZLab!

Our lab maintains shared computing resources for researchers to use at their discretion. This can be running computations that require lots of memory, storage, CPU power, or a simply have a long runtime. To make sure that the capabilities of the machines are sufficient and instill *common sense* as it relates to the usage of these resources, we have put together a few guidelines for the usage of the Zlab machines.

Communication

At the time of writing, ZLab members can `ssh` into the following machines:

- z010
- z011
- z012
- z013
- z014
- z020
- z021
- z022

Acessing the servers is covered in complete depth on the wiki.

Keep in mind that other users will almost always be logged in while you are using a server. When others are logged in, you should be aware of your resource usage and how it will affect others on that server. If it happens to be a resource intensive, but short-lived job, communicate with other users logged in on that server to minimize disruptions. Typically, lab members choose one server as their *primary* machine which they perform day-to-day tasks on. That said, it is entirely acceptable to switch to other servers based on resource availability. Remember that *everything* you run on the servers is your responsibility. Do not hit *enter* without giving a second-thought to the potential resource burden on a server.

In the next sections, we will go over the resources available as well as basic commands for job control and UNIX/Linux usage.

Resources

Server	CPU Model	CPU Cores	CPU Threads	RAM (GB)
z010	Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz	112	112	1510
z011	Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz	112	112	1510
z012	INTEL(R) XEON(R) PLATINUM 8568Y+	192	192	1007
z013	Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz	112	112	1510
z014	Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz	112	112	1510
z020	Intel(R) Xeon(R) Gold 6134 CPU @ 3.20GHz	32	32	376
z021	Intel(R) Xeon(R) Gold 6134 CPU @ 3.20GHz	32	32	376
z022	Intel(R) Xeon(R) Gold 6134 CPU @ 3.20GHz	32	32	376

CPU

Servers can have 32, 112, or 192 cores. For specific details, refer to the table at the beginning of of the resources section.

While a server can run hundreds of jobs (processes) at once, only as many processes as as there are CPU cores can run simultaneously (in parallel). The Linux operating system will try to schedule jobs that are not sleeping (i.e. waiting for I/O or network) among the available cores.

Switching between jobs is a relatively inefficient process and so, higher job:CPU ratios can lead to dramatic loss of computational efficiency. In extreme cases a server may end up spending most of its time trying to manage the scheduling of jobs and "context switching" cores. It is generally a bad idea to submit more jobs than there are cores available. The `top`, `htop`, and `btop` programs can be used to monitor CPU and memory usage.

Memory

Servers can have 0.3, 1.0 and 1.5 *terabytes* of **r**andom-**a**ccess **m**emory (RAM). For specific details, refer to the table at the beginning of of the resources section.

RAM is used by the operating system and its associated processes and to hold whatever programs are running and the memory they allocate. The operating system abstracts memory to a so-called "virtual memory", making it possible to over-allocate memory. In the case of memory contention the operating system will swap parts of this memory to disk (swapping). Swapping is extremely slow and once a system is forced to swap out memory it will slow down dramatically. In general, a system in this state will respond slowly or not at all, barring users from logging in and often even from killing their own processes. Our policy is to favor a reboot over extensive swapping.

In the context of processes, the terms "virtual" and "resident" memory usage refer to logically and actually allocated physical memory. Memory not used by any processes is used by the operating system as a pool to cache blocks read from disk, alleviating the slowness of disk access (this shows up as "cache" or "buffer" in `op` and `free`). The previously mentioned top commands as well as `free`, `ps` and `vmstat` can be used to investigate memory usage system-wide and per process.

I/O (input/output)

I/O is

> any transfer of information to or from the CPU/memory combo, for example by reading data from a disk drive…
>
> — Wikipedia

However, for this guide, we will focus on I/O devices and their role as storage devices.

HDDs The servers use HDDs (hard disk drives) also known as *mechanical hard drives* or *spinning disks*.

HDDs are much slower compared to their SSD counterparts. The time to access a disk is basically divided into two pieces, seek time and read/write time. The seek time of a single disk is the most

expensive part and it is generally on the order of milliseconds. Disk operations can be categorized as "burst" or "scattered", the former being reading/writing large contiguous pieces of data and the latter being short read operations scattered across disks. Although the cost of disk access (for frequently accessed data) is somewhat reduced by the operating system (i.e. by caching/speculative reading, etc.), it remains costly and a source of contention.

The commands `vmstat` and `iostat` provide insight into I/O activity and the closely related management of virtual memory and disk cache.

SSDs TODO: Add section about SSDs Michael

Storage Pools TODO: Talk about Ceph and the concept of a storage pool Michael

Networking

TODO: Talk about topology, implications in reading/writing to and from `/data` and `/zata`? Michael TODO: Firewall limitations? TODO: `/zata/public_html`

Since we all use the internet, this is probably the most tangible resource of all.

Useful Commands

`man`: Manual pages. Self-teach yourself on any command and on how to use Linux.

`kill`/`killall`: Handy utilities to (forcefully) kill your jobs. These can be life-savers when things get out of hand.

`who`: Shows *who* is currently logged on to the system. Run `finger` to figure out the person behind an account name if it is not obvious.

`nohup`: Run command in the background and isolate it from the "hangup" (HUP) signal. This means that if you run your jobs using `nohup`, it is safe to disconnect your session without killing the running job. For example,

```
1 nohup ./myscript.sh >myscript.log 2>&1 &
```

would run `myscript.sh` in the background, piping both stdout and stderr to a log file. The `bash` manual page has a section on job control and input/output redirection.

`screen`: GNU screen is a virtual terminal multiplexer. It runs in the background and keeps (multiple) terminal sessions open, even if you disconnect, allowing you to keep processes running inside a shell session and reconnect to the session at a later point.

`ulimit`: This is a bash built-in shell command that can be used to set resource limits in the current session and for all programs started in that session. Limits can be set on the total number of processes/threads, a maximum amount of memory, the maximum priority at which programs can be scheduled, etc. Any process started under certain ulimit restrictions can further decrease but not increase their limits. ulimit can be useful if you find it hard to estimate your resource usage.

`renice`/`nice`: Thse are bash built-in shell commands that can be used to set the "nice level" (scheduling priority) for a process. The scheduling priority for a process is inversely proportional to the nice value. Consider lowering the priority of long running processes if you can, causing it to use resources (at normal speed – low priority does not equal slow processing) when available and at the same time be *nice* to other processes.

`parallel`: GNU parallel is installed on all servers and it consists of a number of tools that make it easy to run large numbers of jobs in the background while ensuring you never actually run more than a set number of jobs at any given time. In order to not overwhelm a server with too many jobs at once, the use of `parallel` and `sem` is highly encouraged.

To keep the most up-to-date on CLI tools for Linux, please refer to the wiki.

A note on *multithreaded* programs

Some programs such as bowtie2, samtools, and pigz are capable of leveraging parallelisms via multithreading. When using such programs, set number of threads equals to the *number of CPU cores* you want to program to use. Be sure to know how many threads are spawned (by checking `top`, for example) and adjust that number as to not exceed the number of cores available on a server.