Server, Workscheduler and other unknown creatures

Server vs. Laptop

Laptop:

- \+ Fast clock speed, but
- − few cores
- − Limited RAM
- \+ Graphical interface and programs like IDEs
- − Not shared, thus root access and private environment which allows installing software globally

Server:

- \+ Many cores
- \+ Large RAM
- \+ GPU available with large VRAM
- \+ Runs 24/7
- \+ Fast connection to NAS
- − Slow clock speed
- − No Graphical interface
- − Shared with other people

Our servers:

TABLE 1 Hardware configurations and benchmark setups.

CPU			GPU			
No. of cores	Clock speed [GHz]	Memory [GB]	Architecture	No. of CUDA-cores	Memory [GB]	Performance (single) [TFLOPS]
Server 1 (S1) Ubuntu 16.04 LTS						
Dual AMD Opteron 6380			GeForce GTX 970			
2 × 16	2.5	128	Maxwell	1,664	4	3.5
Server 2 (S2) Ubuntu 16.04 LTS						
Dual Intel Xeon E5-2630 v4			-			
2 × 10	2.2	192	-	-	-	-
Server 3 (S3) Ubuntu 20.04 LTS						
Intel Xeon Gold 6248R			Quadro RTX A6000			
24	3.0	128	Ampere	10,752	48	38.7

Server 4 is equivalent to Server 3.

Server vs. Laptop: When to use?

Laptop:

- Development and debugging of code
- Code / Programs which require a fast GUI (otherwise ssh -X can be used on server)
- Jobs which can not be parallelized and are dependent on the speed of a single core (if they run reasonable time)

Server:

- Execution of code
- Jobs which require large amounts of RAM
- Jobs which require a CUDA-GPU
- Jobs which can be parallelized
- Execution of many single core jobs

How to use our servers? - without scheduler

1. Ensure you have an account on our servers. (Otherwise ask for it)
2. Connect to server 1-4: ssh can be extended by options like -X (graphical output or -L to forward ports to access, e.g. a Jupyter server)

   ssh *username*@agmn-srv-3.zoologie.uni-koeln.de
3. Install your environment locally. You don't have root access, thus software can not be installed via apt-get install …, but has to be compiled and installed in your folder on the NAS (or on the server, but I would not recommend it).
4. Transfer files to NAS (or server). You can find our NAS via the network discovery of your laptop. Otherwise, you can mount one of the servers (sftp://username@agmn-srv-3.zoologie.uni-koeln.de/) in the file explorer and find the NAS under /agnawrot-nas/
5. Use a terminal multiplexer like tmux or screens to start a permanent session which does not stop when you disconnect from the ssh connection
6. Load your environment and start your job.

Multiprocessing basics

- Programs can be parallelized via Multi-threading or multiple processes
- Multiple-threads share the same memory space, while multiple processes have different ones → Shared Variables in Threads and different ones in processes
- Global interpreter lock of Python does only allow execution on one core in a serial manner → Multiple threads share the same CPU and are thus scheduled in an intermittent manner
- We can use different tools of interprocess communication to orchestrate multiple processes / threads.

Work scheduler

- **The Role of Schedulers in HPC Clusters**
  - Without a scheduler, an HPC Cluster is a chaotic mix of server jobs.
  - Users in large clusters face confusion about resource availability.
- **Importance of Cluster Batch Control Systems**
  - Cluster batch control systems resolve these issues.
  - They utilize HPC Schedulers for efficient job management.
- **Key Functions of Cluster Batch Control Systems**
  - Sequential job queuing.
  - Priority assignment for jobs.
  - Distributing and parallelizing tasks.
  - Suspending and terminating jobs cluster-wide.
- **Why Schedulers are Essential**
  - Schedulers enable orderly and efficient cluster operation.
  - They optimize resource utilization and user experience.
  - Essential for achieving high-performance computing goals.

Work scheduler - How to use them

Prepare your code:

- It is better to submit multiple small jobs then one big
  - Small jobs can be better distributed across multiple servers and run together with jobs from other users
  - Try to avoid parallelization schemes which generates long periods of only one core used -e.g. numerical methods with convergence problems -> better submit multiple jobs with only a few cores or even only single cores
- Use command line arguments and/or environment variables to control the behavior of your code
  - Both are available in all schedulers - most schedulers also supply some environment variables which are set based on the allocated resources
  - command line arguments are good to supply simulation parameters …
- Write results of intermediate steps to disk and collect them with another job afterwards
- Write a shell script which loads the right environment -> e.g. loads conda and activates the right env.
- Write a job submission script which request the right amount of resources, sets up the logs, states the shell script as executable and sets the right env. variables and command line arguments. Here you can use loops to transmit multiple jobs with different arguments to e.g. simulate a network multiple times with different parameters (grid search). The job submission script can be either a shell script or a python script which uses an API for the scheduler.
- Submit the jobs, check the logs, job status etc..

- Submission of interactive job to debug:
  You can acquire an interactive job which is basically a terminal session with allocated resources via …

Containerization:

- Containers are closed environments for applications
- The overhead of containers is small compared to virtual machines as they use the kernel functionality of the host
- Environments can be freely configured with full root access
- Images of containers can be shared between multiple devices and allow for reproducibility independent of the host OS and software stack
- Containers might be slower, if the chosen software stack does not fit the host system. E.g. Intel compilers which use the full Intel instruction set might be faster.

Further tricks

- cluster ssh to run code on multiple hosts
- ssh-copy to share keys and don’t need to login via password
- ssh -L port1:port2 to share a port e.g. jupyter lab
- ssh -X to direct xserver output to you (might need to install additional software on MAC)