HPC@UAntwerp introduction

Stefan Becuwe, Franky Backeljauw, Bert Tijskens, Kurt Lust, Carl Mensch, Robin Verschoren, Michele Pugno

Version Spring 2024

HPC@UAntwerp introduction

Table of contents – Part 1

VLAAMS
SUPERCOMPUTER
CENTRUM

HPC@UAntwerp introduction

Table of contents – Part 2

VLAAMS SUPERCOMPUTER CENTRUM

HPC@UAntwerp introduction

1 – Introduction to the VSC

Vlaanderen
is supercomputing

VLAAMS SUPERCOMPUTER CENTRUM

Innovative Computing for A Smarter Flanders

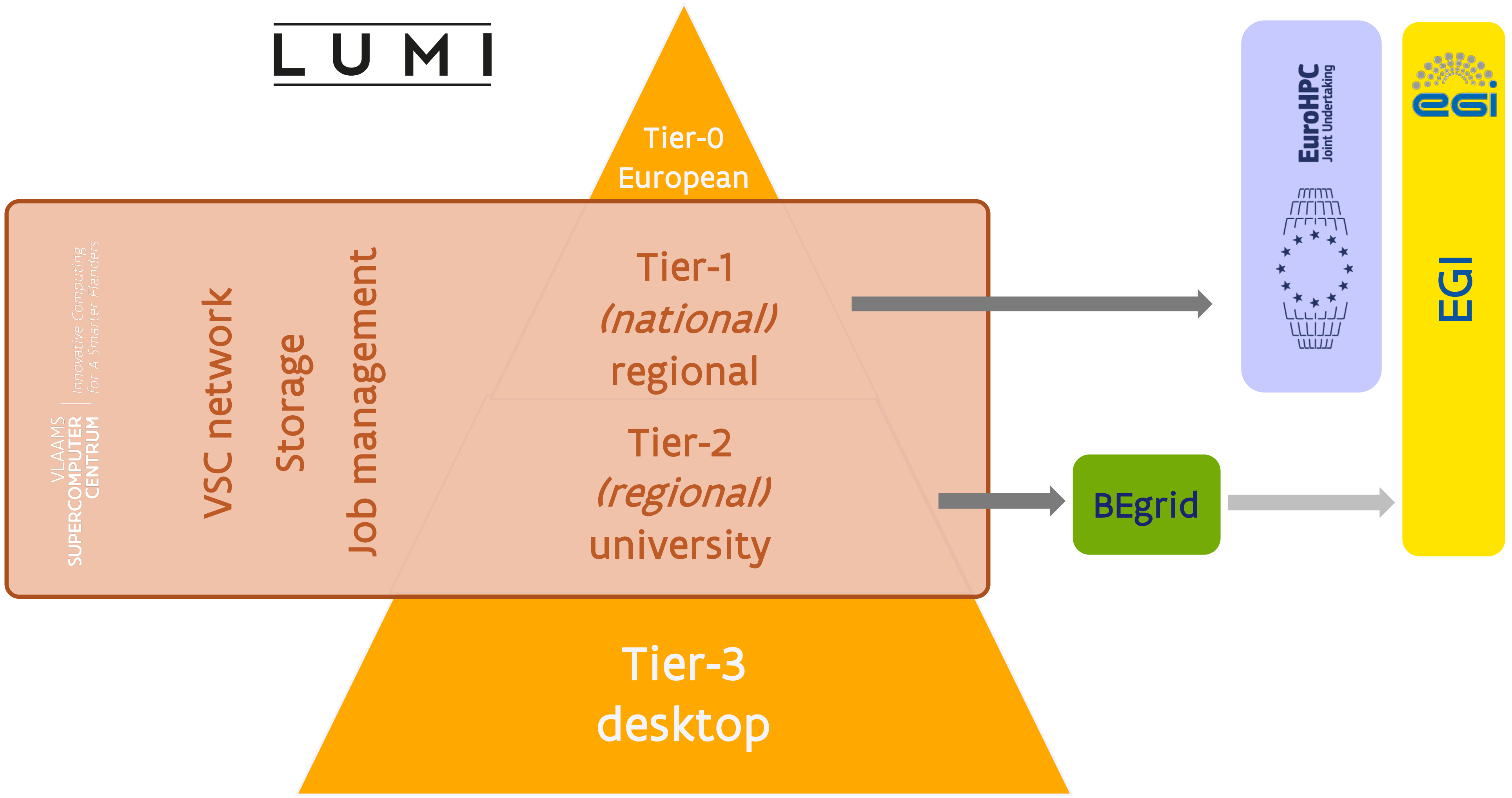
vscentrum.be

VSC – Flemish Supercomputer Center

- Vlaams Supercomputer Centrum **(VSC)**
  - Established in December 2007
  - Partnership between 5 University associations: Antwerp, Brussels, Ghent, Hasselt, Leuven
  - Funded by Flemish Government through the FWO (Research Fund – Flanders)
  - Goal: make high-performance computing (HPC) available to all researchers in Flanders
  - Targets both academic and industrial landscape
  - Provides central **Tier-1** infrastructure with Compute, Cloud and Data components
- Local **Tier-2** infrastructure at UAntwerp: **HPC core facility CalcUA**
  - Provides faculty, staff and students with a HPC infrastructure and software for research purposes
  - Offer training, assistance and support for all research involving HPC projects
- Other local Tier-2 infrastructures at VUB, UGent and KU Leuven / UHasselt
- [VSC docs] Infrastructure

VLAAMS
SUPERCOMPUTER
CENTRUM

VSC – An integrated infrastructure according to the European model



VLAAMS
SUPERCOMPUTER
CENTRUM

UAntwerp Tier-2 infrastructure

Leibniz (2017)	**Vaughan** (2020-2021)
• 152 regular compute nodes - dual 2.4GHz 14-core Intel **Broadwell** processors (Xeon E5-2680v4) - 144 nodes with 128 GB memory - 8 nodes with 256 GB memory	• 152 regular compute nodes - dual 2.35GHz 32-core AMD **Zen 2** (Rome) processors (Epyc 7452) - all nodes with 256 GB memory
• 2 NVIDIA GPU compute nodes - dual Tesla **Pascal** P100	• 1 NVIDIA GPU node - quad Tesla **Ampere** A100 (with 40 GB HBM2e) • 2 AMD GPU nodes - dual Radeon Instinct **Arcturus** MI100
• 1 vector engine node - NEC SX-Aurora TSUBASA A300-2	
• 2 login nodes - identical processor as in the compute nodes	• 2 login nodes - dual 2.8GHz 16-core AMD Zen 2 (Rome) processors (Epyc 7282)
• 1 visualisation node - with NVIDIA Quadro Pascal P5000 GPU	

UAntwerp Tier-2 infrastructure

Breniac (2023)

- 23 regular compute nodes
  - dual 2.6GHz 14-core Intel **Skylake** processors (Xeon Gold 6132)
  - all nodes with 192 GB memory
- 1 login node
  - identical processor as in the compute nodes

Vaughan (2023)

- 40 regular compute nodes
  - dual 2.8GHz 32-core AMD **Zen 3** (Milan) processors (Epyc 7543)
  - 28 nodes with 256 GB memory
  - 12 nodes with 512 GB memory

➢ The Breniac nodes were recovered from old the Tier-1 machine BrENIAC and replace the even older UAntwerp Tier-2 Hopper nodes which were decommissioned in the summer of 2023.

➢ The compute nodes have a small compressed swap space (in memory zram block device) and a small SSD available as a local `/tmp` partition.

➢ The shared storage system is a joint setup with more than 730TB capacity spread over several volumes.

➢ [VSC docs] UAntwerp Tier-2 Infrastructure

UAntwerp Tier-2 infrastructure

CalcUA

VSC

\Orchestrating a brighter world

NEC

VSC Tier-1 infrastructure

Hortense (UGent)

- Inaugurated June 2022
  - 384 compute nodes with dual 64-core AMD Epyc 7H12 (Rome) processors 256 (342 nodes) or 512 (42 nodes) GB memory, HDR100-IB network
  - 20 GPU nodes with dual 24-core AMD Epyc 7402 (Rome) processors and quad NVIDIA Ampere A100-SXM4 (40 GB) NVLink 3
- Phase 2 added in Summer 2023
  - 384 compute nodes with dual 64-core AMD Epyc 7763 (Milan) processors 256 (342 nodes) memory, HDR100-IB network
  - 20 GPU nodes with dual 24-core AMD Epyc 7402 (Rome) processors and quad NVIDIA Ampere A100-SXM4 (80 GB) NVLink 3
- Approximately 5.4 PB shared scratch storage capacity

- [VSC docs] Tier-1 Infrastructure

VSC Tier-1 infrastructure

Hortense (UGent)

VSC Tier-1 infrastructure

Hortense (UGent)

Characteristics of a HPC cluster

- Shared infrastructure, used by multiple users simultaneously
  - You need to request the appropriate resources for the task at hand
  - You may have to wait a while before your computation starts
- Expensive infrastructure
  - **Software efficiency matters!**
- Built for parallel jobs.
  - **No parallelism = no supercomputing**
  - Not meant for running a single single-core job
- Remote computation model
  - For batch computations rather than interactive applications
  - Can't afford wasting time waiting for user input
- Linux-based systems
  - No Windows or macOS software

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

2 – Get a VSC account

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

SSH and public/private key pairs

- Almost all communication with the cluster happens through SSH (Secure SHell)
  - Protocol to log in to a remote computer
  - Protocol to transfer files (SFTP)
  - Protocol to tunnel IP traffic to/from a remote host through firewalls
- We use public/private key pairs rather than passwords to enhance security
  - The public key is put on the computer you want to access
    - Within VSC, you upload the key to the VSC account page
    - A hacker can do nothing with your public key without the private key
  - The private key stays on your local machine
    - It is never passed over the internet if you do things right
    - You need to keep it very secure (make it readable by your account only)
    - **Protect the private key with a passphrase!**
    - A hacker can do nothing with your private key file without the passphrase

Required software – Windows

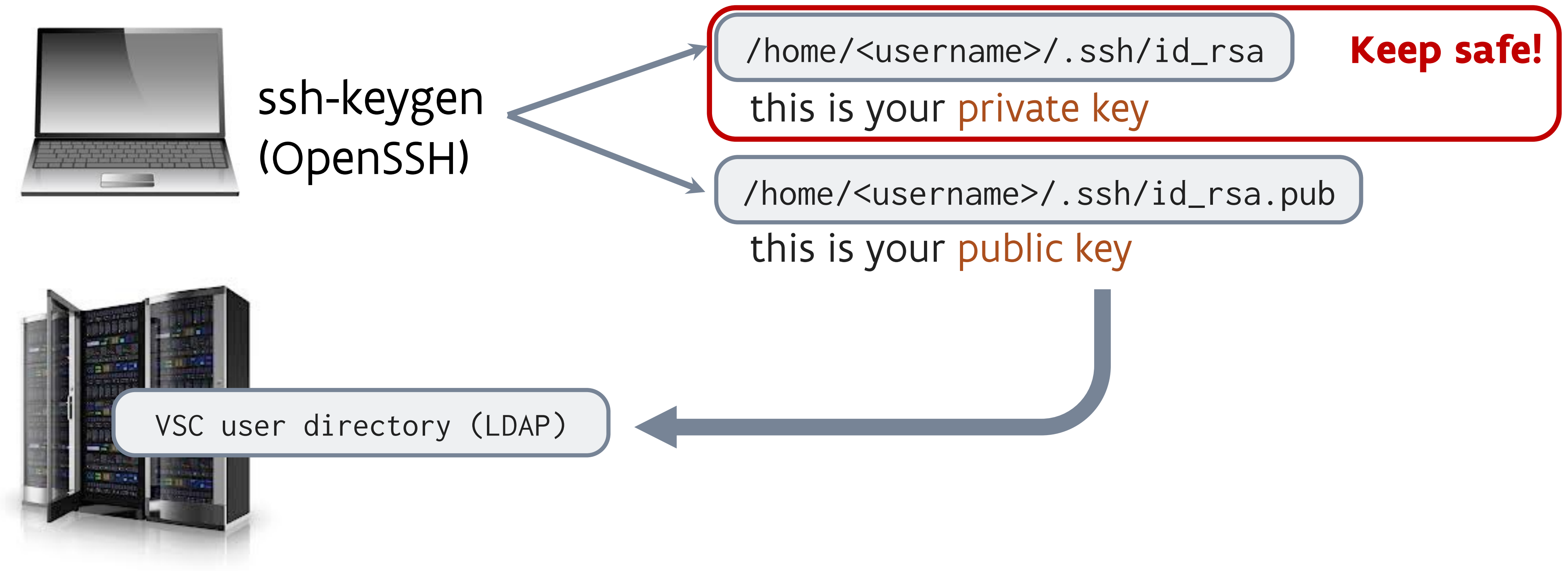
- Windows 10 version 1803 and later come with a Linux-style SSH client in PowerShell or cmd.exe
  - Works as the Linux client
  - In combination with other SSH ports, getting the right permissions on `.ssh` may be a bit tricky
  - Combine with the Windows Terminal (store app)
- PuTTY is a popular GUI SSH client
- MobaXterm combines a SSH/SFTP client, X server and VNC server in one
  - Free edition ("Home Edition") with limited number of simultaneous connections
  - Professional version with unlimited number of connections (yearly subscription)
  - Uses PuTTY keys
- Windows Subsystem for Linux (WSL)
  - Available since Windows 10 version 1709, now WSL 2 starting with Windows 10 version 2004
  - Allows you to install your favourite (often free) Linux distribution from the Windows store

Required software – macOS and Linux

- macOS
  - SSH client included with the OS
  - Terminal package also included with the OS
  - iTerm2 is a free and more sophisticated terminal emulator
  - XQuartz with local xterm is also an option (can be used for running remote graphical applications)
- Linux
  - SSH client included with the OS (though you may need to install it as a separate package)
  - Choice of terminal emulation packages

Creating SSH keys

- Follow the instructions: [VSC docs] Generating keys
- Upload the public key to the VSC account page


ssh-keygen
(OpenSSH)
/home/<username>/.ssh/id_rsa
Keep safe!
this is your private key
/home/<username>/.ssh/id_rsa.pub
this is your public key
VSC user directory (LDAP)

Creating SSH keys

A technical note

- We currently support two cryptography technologies for keys at the VSC
  - RSA, but it should use at least 4096 bits

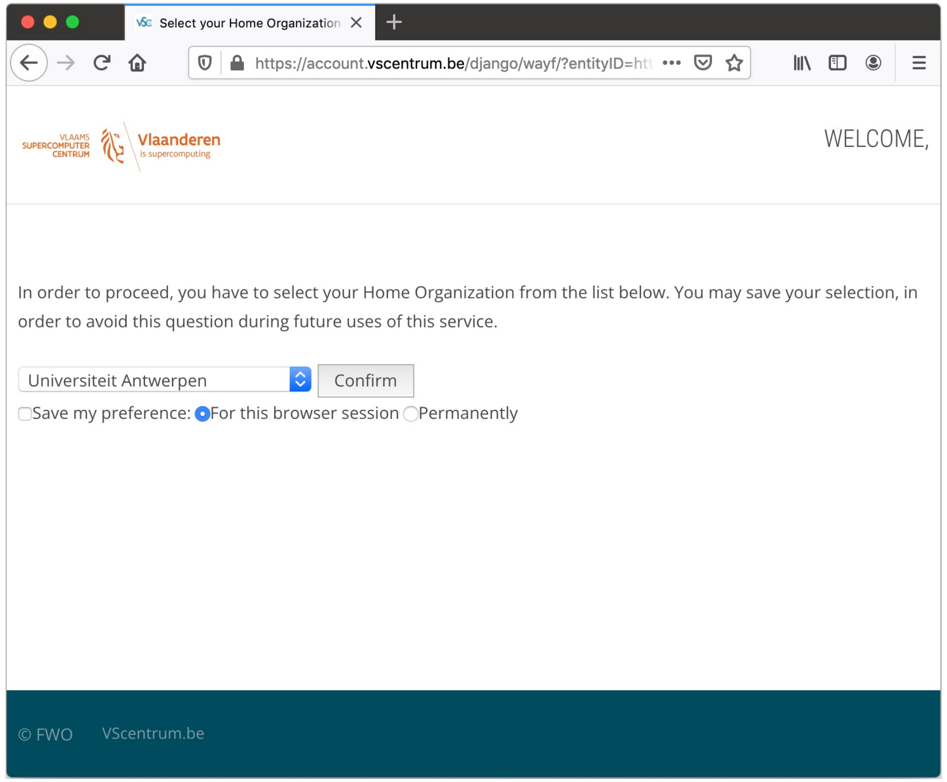
    ```
    $ ssh-keygen -t rsa -b 4096
    ```

  - ed25519 keys
- Our advice is to stick to RSA 4096 bit keys, since some client software does not support ed25519 keys and even longer RSA keys are also often not supported.
- Some SSH implementations use a different file format (PuTTY, SSH2), and those keys may need conversion.
- On Windows with PuTTY and PuTTYgen key generator
  - use PuTTY key format 2 in latest version
  - Convert the public key to OpenSSH format

VLAAMS
SUPERCOMPUTER
CENTRUM

Register your account

- Web-based procedure via the VSC account page



Select your Home Organization

https://account.vscentrum.be/django/wayf/?entityID=htt

VLAAMS SUPERCOMPUTER CENTRUM Vlaanderen is supercomputing

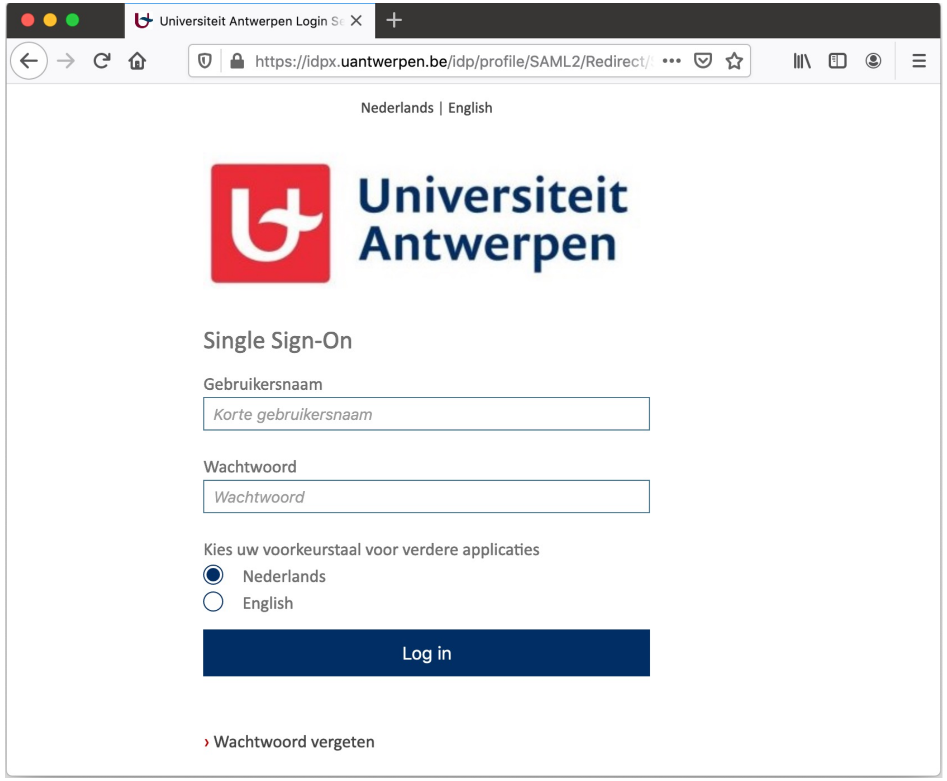
WELCOME,

In order to proceed, you have to select your Home Organization from the list below. You may save your selection, in order to avoid this question during future uses of this service.

Universiteit Antwerpen | Confirm

☐ Save my preference: ● For this browser session ○ Permanently

© FWO VScentrum.be



Universiteit Antwerpen Login Se

https://idpx.uantwerpen.be/idp/profile/SAML2/Redirect/

Nederlands | English

Universiteit Antwerpen

Single Sign-On

Gebruikersnaam: Korte gebruikersnaam

Wachtwoord: Wachtwoord

Kies uw voorkeurstaal voor verdere applicaties

● Nederlands

○ English

Log in

› Wachtwoord vergeten

your VSC username is **vsc20xxx**

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

3 – Connect to the cluster

VLAAMS
SUPERCOMPUTER
CENTRUM

*Innovative Computing
for A Smarter Flanders*

vscentrum.be

A typical workflow

1. **Connect to the cluster**
2. Transfer your files to the cluster
3. Select the software and build your environment
4. Define and submit your job
5. Wait while
   - ➢ your job gets scheduled
   - ➢ your job gets executed
   - ➢ your job finishes
6. Move your results

Types of cluster nodes

- A HPC cluster typically has
  - login nodes : the part of the cluster to which you connect and where you launch your jobs
  - visualisation nodes : if you need to run software that requires a graphical user interface
  - compute nodes : the part of the cluster where the actual computations are done
  - storage and management nodes : as a regular user, you don’t have access to them
- Use the login nodes for
  - accessing the clusters
  - editing and submitting jobs, also via some GUI applications
  - small compilations only – submit large builds as a regular job
  - visualisation (but we also have a dedicated visualisation node for that purpose)

Do not use the login nodes for running your jobs

The login nodes

- UAntwerp login nodes: **restricted public access**
  - Only directly accessible (using SSH) from within Belgium
  - When connecting from abroad, you need a VPN connection to the UAntwerp network
    - Service provided by UAntwerp ICT department, not by CalcUA/VSC
    - Preferably, use the Cisco AnyConnect VPN Client, downloadable via vpn.uantwerpen.be or vpnmfa.uantwerpen.be (split tunnel alternative)
    - OS-builtin clients sometimes fail to connect, and are not supported
    - Instructions can be found in the “ICT Guide” on Pintra (section “Remote working - VPN”)
- In some cases, you need to use individual login names rather than the generic ones (see next slide)
  - When using a graphical desktop (via VNC) for GUI applications
  - When using the screen or tmux commands and you want to reconnect
  - When using Eclipse or VS Code remote projects

The login nodes

Cluster	External name	Internal name
Vaughan (generic name)	**login-vaughan.hpc.uantwerpen.be**	login.vaughan.antwerpen.vsc
Vaughan (individual login nodes)	login1-vaughan.hpc.uantwerpen.be login2-vaughan.hpc.uantwerpen.be	login1.vaughan.antwerpen.vsc login2.vaughan.antwerpen.vsc
Leibniz (generic name)	**login-leibniz.hpc.uantwerpen.be** **login.hpc.uantwerpen.be**	login.leibniz.antwerpen.vsc
Leibniz (individual login nodes)	login1-leibniz.hpc.uantwerpen.be login2-leibniz.hpc.uantwerpen.be	login1.leibniz.antwerpen.vsc login2.leibniz.antwerpen.vsc
Leibniz (visualisation node)	viz1-leibniz.hpc.uantwerpen.be	viz1.leibniz.antwerpen.vsc
Breniac	**login-breniac.hpc.uantwerpen.be**	login.breniac.antwerpen.vsc

- Once on the cluster, you can also use the internal names to go to another node
  - These names can also be used to connect from other VSC clusters via the internal VSC network

Connecting to the cluster
Using SSH

- Using PuTTY (Windows only)
  - Follow the instructions: [VSC docs] Text-mode access using PuTTY
- OpenSSH-derived clients
  - Open a terminal
    - Windows 10: PowerShell or a bash command prompt in a terminal emulator
    - macOS: Terminal which comes with macOS, or iTerm2
    - Ubuntu: Applications > Accessories > Terminal or Ctrl+Alt+T
  - Login via secure shell

```
$ ssh vsc2xxxx@login.hpc.uantwerpen.be
```

    - if your private key file has the standard filename, otherwise point to it with `-i`

Connecting to the cluster
Using a SSH config file

- Using PuTTY (Windows only)
  - Follow the instructions: [VSC docs] Text-mode access using PuTTY
- OpenSSH-derived clients
  - Open a terminal
    - Windows 10: PowerShell or a bash command prompt in a terminal emulator
    - macOS: Terminal which comes with macOS, or iTerm2
    - Ubuntu: Applications > Accessories > Terminal or Ctrl+Alt+T
  - Login via secure shell

```
$ ssh vsc2xxxx@login.hpc.uantwerpen.be
```

    - if your private key file has the standard filename, otherwise point to it with `-i`

Connecting to the cluster

Using an SSH config file

```
Host *
  ServerAliveInterval 60

Match User vsc2xxxx
  IdentityFile ~/.ssh/id_rsa_vsc

Host calcua
  User vsc2xxxx
  HostName login.hpc.uantwerpen.be
  ForwardAgent yes
  ForwardX11 yes
```

Use these options for all the hosts you connect to
- (Try to) Keep the connection with the server alive

These options apply when connecting as user vsc2xxxx
- The location of the private key

Use the name “calcua” as an alias for the following
- Connect as user vsc2xxxx
- To the login node login.hpc.uantwerpen.be
- Use agent forwarding (for subsequent ssh calls (`-A`))
- Use X11 forwarding (for visualisation purposes (`-X`))

- Put this file in ~/.ssh/config and then you can connect using: `ssh calcua`
- [VSC docs] SSH config

VLAAMS
SUPERCOMPUTER
CENTRUM

HPC@UAntwerp introduction

4 – Transfer your files to the cluster

Vlaanderen
is supercomputing

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

A typical workflow

1. Connect to the cluster
2. **Transfer your files to the cluster**
3. Select the software and build your environment
4. Define and submit your job
5. Wait while
   - your job gets scheduled
   - your job gets executed
   - your job finishes
6. Move your results

VLAAMS SUPERCOMPUTER CENTRUM

Transferring your files

- **sftp** protocol – ftp-like protocol based on ssh, using the same keys
- Graphical ssh/sftp file managers for Windows, macOS, Linux
  - Windows: PuTTY, WinSCP, MobaXterm (all using PuTTY keys)
  - Multiplatform: FileZilla (also supports server-to-server transfers (FXP))
  - macOS: CyberDuck
- Command line: **scp** - secure copy
  - copy from your local computer to the cluster

```
$ scp file.ext vsc2xxxx@login.hpc.uantwerpen.be:
```

  - copy from the cluster to your local computer

```
$ scp vsc2xxxx@login.hpc.uantwerpen.be:file.ext .
```

- Consider using **Globus** for big transfers
  - Requires software on the computer from/to where you want to transfer files
  - Supports data and scratch file systems on UAntwerp clusters

User directories

- `/scratch/antwerpen/2xx/vsc2xxyy`
  - fast scratch for temporary files, per cluster/site
  - highest performance, especially for bigger files
  - highest capacity (0.6 PB)
  - does not support hard links

`$VSC_SCRATCH`

- `/data/antwerpen/2xx/vsc2xxyy`
  - to store long-time data
  - exported to other VSC sites
  - slower and much smaller (60 TB) than /scratch
  - prefer no jobs running in this directory unless explicitly told to do so

`$VSC_DATA`

- `/user/antwerpen/2xx/vsc2xxyy`
  - only for account configuration files
  - exported to other VSC sites
  - never run your jobs in this directory!
  - very small but good performance for its purpose

`$VSC_HOME`

File systems

- **/scratch** uses a parallel filesystem (BeeGFS), hence it is more optimized for large files.
- **/data** and **/user** use a standard Linux filesystem (xfs) and have better performance for small files (a few kB) which is why we suggest to install packages for Python, Perl, R, etc. on /data.
  - The underlying storage technology of /user is very expensive so it should only be used for what it is meant to be used.

- Some fancy software installers still think it is a good idea to use a lot of hard links. These don't work on /scratch.
  - Conda-software should be installed on /data as it will not work on /scratch unless you force conda to use symbolic links instead.

- Currently, we do take a backup of /user and /data, but in the future we may exclude certain directories if the volume and number of files becomes too large.
  - Conda-installed Pythons are an example of directories that may be excluded as every Conda installation tries to install half an OS.

Block and file quota

- Block quota: limits the amount of data you can store on a file system
- File quota: limits the number of files you can store on a file system

- Current defaults:

File system	Block quota	File quota
/scratch	50 GB	100,000
/data	25 GB	100,000
/home	3 GB	20,000

- Quotas are shown when you log in or by running the **`myquota`** command.

- Note: on /scratch, the number of counted files corresponds to the number of data chunk files (created on the BeeGFS storage targets) and does not include the number of directories.
  - Currently, an end-user created file can be spread over at most 8 data chunk files.

VLAAMS SUPERCOMPUTER CENTRUM

Globus data sharing platform

- Service to transfer large amounts of data between local computers and/or remote servers
  - Transfer data from your local computer (laptop/desktop) to a remote server (or vice versa)
    - Globus Connect Personal software needed on your local computer
    - after disconnecting, the transfer will be resumed automatically
  - Using a direct transfer between two remote servers, initiated from your local computer
    - no software needed on your laptop except for a recent web browser
  - It also offers data sharing features (guest collections)
- **Globus web app**
  - It is possible to sign in with your UAntwerp account
  - You can also create a Globus account and link that to your UAntwerp account
  - You do need an active VSC account to access data on our servers
- HPC@UAntwerp collection: **VSC UAntwerpen Tier2**
  - From there you have access to /data and /scratch
- [VSC docs] Globus data sharing platform

Best practices for file storage

- **The cluster is not for long-term file storage.**
  - You need to move back your results to your laptop or server in your department
  - There is a backup for the /user and /data, but we can only continue to offer this if these file systems are used in the proper way, i.e., not for very volatile data
  - Old data on scratch can be deleted if scratch fills up (so far this has never happened at UAntwerp, but it is done regularly on some other VSC clusters)
- Remember that the cluster is optimised for parallel access to big files, not for using tons of small files (e.g., one per MPI process).
- On scratch (but also on other file systems)
  - If you run out of block quota, you can get more without too much motivation
  - If you run out of file quota, you will have to motivate why you need more files
- Note: text files are good for summary output while your job is running, or data that you'll read into a spreadsheet, but not for storing 1000x1000-matrices – use binary files for that!

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

5 – Select the software and build your environment

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing for A Smarter Flanders

vscentrum.be

A typical workflow

1. Connect to the cluster
2. Transfer your files to the cluster
3. **Select the software and build your environment**
4. Define and submit your job
5. Wait while
   - ➢ your job gets scheduled
   - ➢ your job gets executed
   - ➢ your job finishes
6. Move your results

System software

- Operating system: **Rocky Linux** – currently, version 8.9
  - Red Hat Enterprise Linux (RHEL) 8 clone
  - Installed on all CalcUA clusters: Vaughan, Leibniz and Breniac
    - All clusters are kept in sync as much as possible
- Resource management and job scheduler: **Slurm**
- Software build and installation framework: **EasyBuild**
- Environment modules system: **Lmod**

Rocky Linux™

an open enterprise operating system

slurm

workload manager

EasyBuild.io

building software with ease

Lmod

Development software

- C/C++/Fortran compilers: Intel oneAPI and GCC
  - Intel compiler comes with several tools for performance analysis and assistance in vectorisation/parallelisation of code
  - Fairly up-to-date OpenMP support in both compilers
  - Support for other shared memory parallelisation technologies such as Cilk Plus, other sets of directives, etc., to a varying level in both compilers
  - PGAS Co-Array Fortran support in Intel Compiler (and some in GNU Fortran)
  - Due to some VSC policies, support for the latest versions may be lacking, but you can request them if you need them
- Message passing libraries: Intel MPI (MPICH-based), Open MPI (standard and Mellanox)
- Mathematical libraries: Intel MKL, OpenBLAS, FFTW, MUMPS, GSL, …
- Many other libraries, including HDF5, NetCDF, Metis, …
- Scripting and/or programming languages, such as Python, Perl, …

Application software

- Installed in **/apps/antwerpen**
  - Preferably built and installed using EasyBuild
  - Often multiple versions of the same package
- ABINIT, CP2K, OpenMX, QuantumESPRESSO, *Gaussian*
- *VASP*, Siesta, CPMD
- GROMACS, NAMD2
- TELEMAC
- Bowtie, BLAST
- Gurobi
- *MonolixSuite*
- *COMSOL*, R, MATLAB, …

Additional software can be installed on demand

- [EasyBuild] List of supported software

VLAAMS SUPERCOMPUTER CENTRUM

Application software

- Restrictions on commercial (licensed) software
  - 5 concurrent licenses for Intel Parallel Studio XE and Intel oneAPI
  - MATLAB: toolboxes limited to those in the campus agreement
  - Other licenses paid for and provided by the users
    - Owner of the license decides who can use the license (see next slide)
- Additional software can be installed on demand
  - Make sure the system requirements can be met (e.g., no Windows software, …)
  - Provide working building instructions (rpm/deb packages rarely work on a cluster)
  - For commercial software, have a license that is valid for cluster usage
  - Since we cannot have domain knowledge in all science domains,
    users must do the testing with us
- Compilation support is limited: best effort, no code fixing
  - An issue nowadays as too many packages are only tested with a single compiler

Using licensed software

- VSC or campus-wide license: you can just use the software
    - Some restrictions may apply if you don’t work at UAntwerp but at some of the institutions that have access (ITG, VITO) or at a company
- Other restricted licenses
    - Typically licenses paid for by one or more research groups (or individual users), sometimes just other license restrictions that must be respected
    - Talk to the owner of the license first
        - The owner of the license decides who can use the license
        - If no one in your research group is using a particular package, your group probably doesn’t contribute to the license cost
- Access controlled via UNIX groups
        - e.g., `avasp` is the group for VASP users from UAntwerp (resp. `avasp6` for VASP6)
    - To get access, request group membership via the VSC account page (“New/Join group”)
    - The group moderator will then grant or refuse access

The module concept

- Dynamic software management through **modules**
  - Allows to select the version that you need and avoids version conflicts
  - Automatically pulls in all the required dependencies
  - Loading a module sets some environment variables:
    - `$PATH, $LD_LIBRARY_PATH, ...`
    - Application-specific environment variables
    - Specific variables that point to the location of a package (typically starting with `$EB`)
- Software on our system is organised in **software stacks**, built with **toolchains**
  - Toolchain: a compiler bundled with a compatible MPI library and core mathematical libraries
  - Software stack: collection of software installed in the same time frame on the cluster

If you're not loading modules, you're probably doing something wrong

Toolchains

- Regular (common) toolchains: bundle of compiler with compatible MPI and math libraries
  - `intel` : Intel and GNU compilers, Intel MPI and MKL libraries
  - `foss` : GNU compilers, Open MPI, OpenBLAS, FFTW, …
- Subtoolchains: not including MPI or mathematical libraries
  - `GCCcore` : works with corresponding specific intel and/or foss toolchain
- System toolchain: software compiled with the system compilers
- Refreshed twice per year with more recent component versions (2020a, 2020b, 2021a, …)
  - New software is preferably installed against the most recent available version of a toolchain
  - It is impossible to follow pace since the quality of software is generally going down
  - At UAntwerp, we sometimes do silent updates within a toolchain
- [VSC docs] Available toolchains at the VSC
  [EasyBuild] Overview of common toolchains (and their component versions)

Software stack modules

- Software stack modules:
  - `calcua/2023a` : enables the 2023a version compiler toolchain modules (both intel and a compatible foss version) and software built with those compilers
    - Also enables modules provided by calcua/system and calcua/x86_64
  - `calcua/system` : enables modules for software built with the system compilers
  - `calcua/x86_64` : enables modules for software installed from binaries (for x86_64 architectures)
  - `calcua/all` : enables all currently installed application modules (toolchain, system and binary)
- Currently available software stacks and toolchains on the CalcUA clusters:
  - 2023a, 2022a, 2021a : mostly foss, but also intel
  - 2020a : intel only
- We try to support a software stack and their corresponding toolchains for 2 years
- It is a good practice to **always load the appropriate software stack module first** before loading any other module!

Module naming scheme

- Full module name is typically of the form

```
<name of software>/<version>[-<toolchain info>][-<additional info>]
```

  - For the additional information, we only add the components that matter to distinguish between different available modules for the same package (and version)

- Some examples:
  - `intel/2022a`
    - The Intel compiler toolchain module, version 2022a (includes the Intel compilers version 2022.1.0)
  - `Boost/1.79.0-GCC-11.3.0`
    - The Boost libraries, version 1.79.0, built with the `GCC/11.3.0` toolchain
  - `SuiteSparse/5.13.0-foss-2022a-METIS-5.1.0`
    - SuiteSparse library, version 5.13.0, built with `foss/2022a` toolchain, and using `METIS/5.1.0`

No two modules with the same name can be loaded simultaneously

Discovering software

`$ module av`	List (currently) **available** modules • depends on the software stack module you have loaded
`$ module av openfoam`	Search for available modules whose name contains “openfoam” • searching is case-insensitive
`$ module spider`	List **all installed** modules and extensions
`$ module spider openfoam`	Search for installed modules and extensions whose name contains “openfoam”
`$ module spider OpenFOAM/v2206-foss-2022a`	Display additional information about the `OpenFOAM/v2206-foss-2022a` module • shows the software stack modules you may need to load (choose one line only)
`$ module keyword cmake`	Search for a module using all the information in the module definition • also searches in the whatis and help information
`$ module whatis baselibs`	Show whatis information about the module – **the module name is case-sensitive!** • usually only a short description, some keywords, a link to the homepage, …
`$ module help baselibs`	Show help information about the module • can include a longer description, usage instructions, documentation links, … • specify a version to get the most specific information
`$ module help`	Display help about the module command

VLAAMS
SUPERCOMPUTER
CENTRUM

Loading and unloading modules

```
$ module load calcua/2022a
```

Always load an appropriate software stack module first

- this makes all the modules from the given software stack available for loading

```
$ module load
  OpenFOAM/v2206-foss-2022a
```

Load a specific version of a module (if available)

- the module name is case-sensitive
- this automatically pulls in all the required dependencies
- it is advised to **always explicitly specify the name AND version of a module**

```
$ module load OpenFOAM
```

Load the “default” version a module

- indicated by `(D)` in the output of `module av OpenFOAM`
- usually the most recent version (but here it will load `OpenFOAM/11-foss-2022a`)
- Depends on the currently loaded software stack module
- Can change if other versions of the module are installed

```
$ module list
```

List all loaded modules in the current session

```
$ module unload OpenFOAM
```

Unload the module and its settings

- this does not automatically unload the dependencies!

```
$ module purge
```

Unload all (non-sticky) modules – currently, we don’t have any sticky modules yet

- you can force the removal of a sticky module using `module --force`

```
$ ml [...]
```

A handy front end for the module command

Managing module collections

```
$ module purge
```

Start by resetting your environment back to a clean state
- this ensures that no version conflicts occur (…)

```
$ module load [...]
```

Load all the modules you need

```
$ module save work-modules
```

Save the currently loaded modules in a collection named “work-modules”
- this saves the collection to a file `~/.lmod.d/work-modules`

```
$ module restore work-modules
```

Restore the modules from a collection

```
$ module savelist
```

List all your saved collections

```
$ module describe work-modules
```

List the modules that are included in the collection

```
$ module disable work-modules
```

Disable a module collection
- this moves the collection file to `~/.lmod.d/work-modules~`
- so, you can restore the collection by removing the ~
- to remove the collection, delete the collection file from `~/lmod.d`
- note: Lmod is transitioning from using `~/.lmod.d` to `~/.config/lmod`

Best practices for using modules

- Some advice on using modules:
  - It is a good habit to start with setting your environment to a clean state (`module purge`)
    - this ensures that no version conflicts occur, e.g., if the user loads modules in `.bashrc`
  - Always load an appropriate software stack module first
  - Always explicitly specify both the name and version when loading a module
- **Do not load modules in your .bashrc** to set up your working environment
  - you will shoot yourself in the foot at some point
  - consider using module collections instead
- [VSC docs] Software stack (and using the module command)
  [Lmod] User's Tour of the Module Command

HPC@UAntwerp introduction

Exercises – block 1

Vlaanderen
is supercomputing

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Exercises – block 1

- Getting ready – see [VSC docs] Access VSC Infrastructure
  - Log on to the cluster
  - Go to your scratch directory : `cd $VSC_SCRATCH`
  - Copy the examples for the tutorial to that directory: `cp -r /apps/antwerpen/examples .`
- Follow the text from [VSC docs] Software stack and try out the module command.
- Load the cluster module `calcua/2022a`
  - Try to find the modules for the GCC compilers and the CMake tool.
    - Did you notice the difference between module av and module spider?
  - What is the difference between `HDF5/1.12.2-gompi-2022a` and `HDF5/1.12.2-iimpi-2022a`?
  - Which module offers support for VNC? Which command does that module offer?
- Load the cluster module `calcua/2020a`
  - Which module(s) offer the CMake tool in this software stack?
  - What's the difference between `HDF5/1.10.6-intel-2020a-MPI` and `HDF5/1.10.6-intel-2020a-noMPI`?
    - Well, you can guess it from the name, but how can you find more information?)

VLAAMS
SUPERCOMPUTER
CENTRUM

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

6 – Define and submit your job

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

A typical workflow

1. Connect to the cluster
2. Transfer your files to the clusters
3. Select the software and build your environment
4. **Define and submit your job**
5. Wait while
   - ➢ your job gets scheduled
   - ➢ your job gets executed
   - ➢ your job finishes
6. Move your results

Running batch jobs

- A cluster is a large and expensive machine
  - So, the cluster has got to be used as efficiently as possible
  - Which implies that we cannot waste time waiting for input like with an interactive program
- Only a few programs can use the whole capacity (also depends on the problem to solve)
  - Therefore, the cluster is a shared infrastructure
  - Each simultaneous user gets a fraction of the machine depending on his/her requirements
- Moreover, there are a lot of users, so one sometimes needs wait a little
- Hence **batch jobs** (script with resource specifications) submitted to a queueing system with a scheduler to select the next job in a fair way based on available resources and scheduling policies

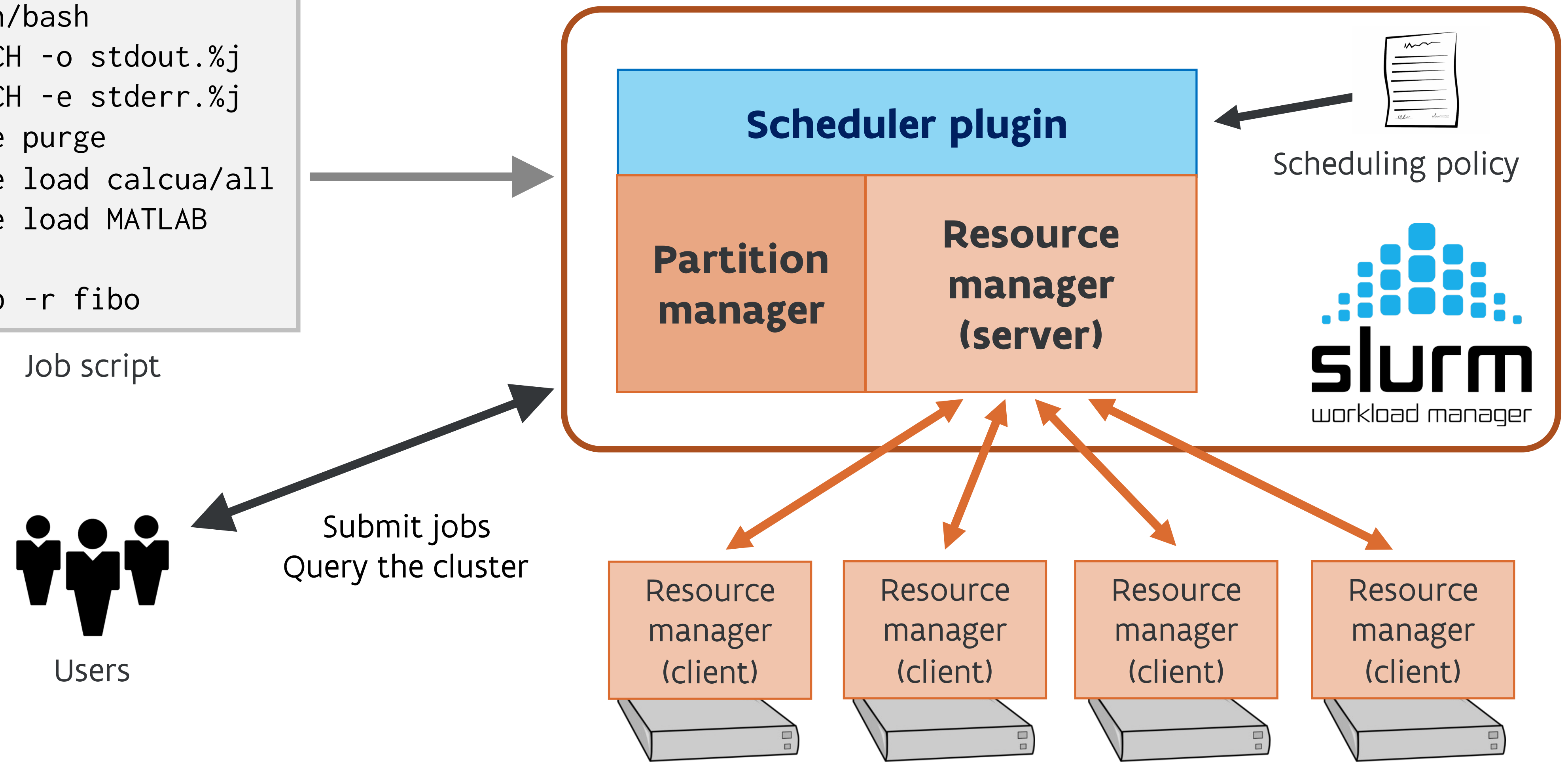
Job submission workflow

Behind the scenes

```
#!/bin/bash
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j
module purge
module load calcua/all
module load MATLAB

matlab -r fibo
```

Job script


Scheduler plugin
Partition manager
Resource manager (server)
Scheduling policy
slurm workload manager
Submit jobs
Query the cluster
Users
Resource manager (client)
Resource manager (client)
Resource manager (client)
Resource manager (client)

The batch job script

- Specifies the resources needed for the job
  - Number and type of CPUs (corresponding to the partition where the job will run)
  - Amount of (RAM) memory needed (often limited per CPU)
  - Set maximum run time (wall time limit) for the job
- Gives instructions to the resource manager
  - Writing output files (stdout and/or stderr) – optional
  - Can instruct the resource manager to send mail when a job starts, ends and/or fails
- Creates the runtime environment
  - By loading the modules needed for the job
- Does the actual computational work
  - Specify all the commands you want to execute
    - These are the commands that you would also execute interactively

An example job script

```
#!/bin/bash

#SBATCH --ntasks=1 --cpus-per-task=7
#SBATCH --mem-per-cpu=1g
#SBATCH --time=1:00:00
#SBATCH -o stdout.%j
#SBATCH -e stderr.%j

module purge
module load calcua/all
module load MATLAB/R2022a

matlab -r fibo
```

The shebang line determines the script language
- This is a bash script, but could also be, e.g., Perl

Specify the resources needed for the job and give some instructions to the resource manager
- Put this as the first block in the script
- These look like comments to Bash

Build a suitable environment for your job
- Load the software stack and application modules

The actual commands that you want to execute
- The script runs where you submitted the job!

Important Slurm concepts

Node	The hardware that runs a single operating system image
Core	A physical core in a system
CPU	A virtual core in a system (hardware thread) • **on the CalcUA clusters, CPU = Core** (since hyperthreading is disabled)
Partition	A group of nodes with job limits and access controls • a node can be part of multiple partitions • serves as general purpose job queue
Job	A resource allocation request
Job step	A set of (possibly parallel) tasks within a job • the job script itself is a special step, called the batch job step • e.g., a MPI application typically runs in its own job step
Task	Corresponds to a (single) Linux process, executed in a job step • a single task can not use more CPUs than available in a single Node • e.g., for a MPI application, each rank (MPI process) is a task but a shared memory program is a single task

Slurm resource requests

- Resource requests can be specified as options in various ways
  - Lowest priority – **In the batch job script** on lines starting with `#SBATCH`
    - they must be at the top of the script, before any executable commands
    - this is the most used mechanism to specify the resources
  - Several options can also be **passed through environment variables** starting with `SBATCH_`
    - the remaining part is derived from the matching command line option
    - when used, these overwrite values set using #SBATCH lines in the job script
    - be careful when using them, especially when hiding them in .bashrc, as they are easily forgotten
      - but they can be useful, e.g., to specify the account if you have access to only one account
  - Highest priority – **On the command line** of `sbatch` and the related command `srun` and `salloc`
    - command line options must be specified before the name of the job script, as otherwise they will be interpreted as arguments to that job script!
    - these overwrite values on both #SBATCH lines and environment variables

Slurm resource requests

- Two variants for the options
  - a long option format, starting with a double dash: `--long-option=<value>`
    - the '=' sign can be replaced by a space
  - sometimes there is a short single-letter flag, starting with single dash: `-S <value>`
    - the space between the flag and the value can be omitted
- Slurm also sets several `SLURM_*` variables in the environment of the running job
  - some of these variables are only available if the corresponding option has been explicitly set, while other variables are always set (with default values filled in, if appropriate)
  - so you can the values set by these options when calling your executables
- [VSC docs] Submission of job scripts: sbatch

  [Slurm] Online manual pages for sbatch, srun and salloc

VLAAMS
SUPERCOMPUTER
CENTRUM

Slurm resource requests

Overview of frequently used options

Long option	Short option	Job environment variable	Description
`--ntasks=<number>`	`-n <number>`	`SLURM_NTASKS` (if set)	Number of tasks
`--cpus-per-task=<ncpus>`	`-c <ncpus>`	`SLURM_CPUS_PER_TASK` (if set)	Number of CPUs per task
`--mem-per-cpu=<amount><unit>`		`SLURM_MEM_PER_CPU`	Amount of memory per CPU
`--time=<time>`	`-t <time>`	`SLURM_JOB_START_TIME` `SLURM_JOB_END_TIME`	Time limit (wall time)
`--account=<ap_proj>`	`-A <ap_proj>`	`SLURM_JOB_ACCOUNT`	Project account to use
`--partition=<pname>`	`-p <pname>`	`SLURM_JOB_PARTITION`	Partition to submit to
`--switches=<count>`			Max count of leaf switches
`--job-name=<jobname>`	`-J <jobname>`	`SLURM_JOB_NAME`	Name of the job
`--output=<outfile>`	`-o <outfile>`		Redirect stdout
`--error=<errfile>`	`-e <errfile>`		Redirect stderr
`--mail-type=<type>`			Event notification (start, end, …)
`--mail-user=<email>`			Email address

Slurm resource requests
Processing resources

Long option	Short option	Job environment variable	Description
`--ntasks=`*<number>*	`-n` *<number>*	`SLURM_NTASKS` (if set)	Number of tasks
`--cpus-per-task=`*<ncpus>*	`-c` *<ncpus>*	`SLURM_CPUS_PER_TASK` (if set)	Number of CPUs per task

- You don’t request processing as physical resources such as nodes and cores, but rather by specifying the number of (parallel) tasks and CPUs (cores) per task that you need.
  - Task : a space for a single process
  - CPUs per task : in most cases, the number of computational threads for a task
- For each task, all of the CPUs for that task are allocated on a single compute node.
- When using multiple nodes, the allocated CPUs for all tasks are distributed equally over all the nodes (except possibly for the last node).
- If not set, the **default values of 1 task and 1 CPU** are used.

Slurm resource requests
Memory

Long option	Job environment variable	Description
`--mem-per-cpu=<amount><unit>`	`SLURM_MEM_PER_CPU` (in megabytes)	Amount of memory per CPU

- Memory is specified per CPU and not per task
  - Just a single parameter corresponding to RAM (swap space cannot be requested)
  - The unit can be either kilobytes (`k`), megabytes (`m`) or gigabytes (`g`) – default is megabytes
  - The amount must be an integer (so `3.75g` is invalid and must be specified as `3840m` instead)
- The job will be rejected if the total amount of memory requested cannot be satisfied.
- On the CalcUA clusters, the memory available for jobs is somewhat less than the total memory installed on a node – i.e., 16 GB is reserved for the OS and file system buffers.
- If not set, a **default value** will be used, equal to the total memory available for jobs on that node, divided by the number of CPUs.
  - e.g., on a Vaughan compute node with 256 GB of (installed) memory, the default value is `3840m` – calculated from (256 GB - 16 GB) / 64 CPUs = 240 / 64 = 3,75GB = 3840 MB

VLAAMS SUPERCOMPUTER CENTRUM

Slurm resource requests

Wall time

Long option	Short option	Job environment variable	Description
`--time=`*<time>*	`-t` *<time>*	`SLURM_JOB_START_TIME` `SLURM_JOB_END_TIME`	Time limit (wall time)

- The time can be specified using in these formats:

  mm | mm:ss | hh:mm:ss | d-hh | d-hh:mm | d-hh:mm:ss

  where: d = days, hh = hours, mm = minutes, ss = seconds
- If the requested wall time exceeds the maximum time limit, the job will be left in a PENDING state.
- When the wall time is reached, each task in each job step is sent SIGTERM followed by SIGKILL.
- The environment variables are UNIX (Epoch) timestamps (in seconds).
- On the CalcUA clusters, the maximum time limit is set to **3 days** for regular compute nodes, 1 day for the GPU nodes, and 7 days for Breniac (Skylake) nodes.
- If not set, a **default wall time of 1 hour** will be assigned.

Slurm resource requests

Project account

Long option	Short option	Job environment variable	Description
`--account=`*<ap_proj>*	`-A` *<ap_proj>*	`SLURM_JOB_ACCOUNT`	Project account to use

- **Starting 1 March 2024**, we are introducing accounting for both compute (jobs) and storage (files)
  - it will be required to specify a **project account** when submitting jobs to the CalcUA clusters
    - ask your supervisor or project account manager to give you access to a project account
    - make sure to use the appropriate project account, according to the project for which you are submitting jobs
- Find out which project accounts you belong to: `myprojectaccounts`
  - every project account starts with ap_ and includes the name of the research group or institute, supervisor or project, course code, …
    - e.g., `ap_medchem`, `ap_tsm2_wh`, `ap_course_2400wetcmp`, …
- [CalcUA] Documentation (Introduction to Accounting @ CalcUA)

And this is enough to submit a job…

- If all resources are specified in the job script, submitting a job is as simple as

```
$ sbatch jobscript.slurm
Submitted batch job 1234567
```

  - sbatch returns with a message containing a **unique jobid** that you will need for use with subsequent job-related commands

- But all parameters specified in the job script in #SBATCH lines can also be specified on the command line, which will then overwrite those in the job script; e.g.,

```
$ sbatch -n <x> -c <y> --mem-per-cpu=<z>g
  -A ap_course_hpc_intro jobscript.slurm
```

where:
  - *x* = the number of tasks
  - *y* = the number of CPUs per task
  - *z* = the amount of memory needed per CPU (in gigabytes)

Slurm resource requests
Partitions

Long option	Short option	Job environment variable	Description
`--partition=`*<pname>*	`-p` *<pname>*	`SLURM_JOB_PARTITION`	Partition to submit to

- Partitions are groups of nodes with:
  - Job defaults and limits: default memory per CPU (`DefMemPerCPU`), maximum memory per CPU (`MaxMemPerCPU`), maximum wall time (`MaxTime`), Default CPUs per GPU (`DefCpuPerGPU`), ...
  - Access controls and scheduling policies: restricted access to a certain group (`AllowGroups`), ...
- Slurm does not automatically assign a job to the optimal partition
  - but the **default partition** (see next slide) is OK for most jobs on our clusters
- Query partition information using `scontrol show partition` or in the output of `sinfo`

- Check [VSC docs] UAntwerp Tier-2 Infrastructure for the available partitions per individual cluster

Slurm resource requests

Partitions on CalcUA clusters

Cluster	Partition	Description	CPU – GPU	Memory per CPU	Max WT
Vaughan	**zen2**	AMD Zen 2 nodes with 256 GB RAM	64 CPU	3.75 GiB (3840m)	3 days
	zen3	AMD Zen 3 nodes with 256 GB RAM	64 CPU	3.75 GiB (3840m)	
	zen3_512	AMD Zen 3 nodes with 512 GB RAM	64 CPU	7.75 GiB (7936m)	
	ampere_gpu	Zen 2 node with NVIDIA Ampere GPUs	64 CPU – 4 GPU	3.75 GiB (3840m)	1 day
	arcturus_gpu	Zen 2 nodes with AMD Arcturus GPUs	64 CPU – 2 GPU	3.75 GiB (3840m)	
Leibniz	**broadwell**	Intel Broadwell nodes with 128 GB RAM	28 CPU	4 GiB (4096m)	3 days
	broadwell_256	Intel Broadwell nodes with 256 GB RAM	28 CPU	8,5 GiB (8704m)	
	pascal_gpu	Broadwell nodes with NVIDIA Pascal GPUs	28 CPU – 2 GPU	4 GiB (4096m)	1 day
Breniac	**skylake**	Intel Skylake nodes with 192 GB RAM	28 CPU	6,29 GiB (6436m)	7 days

- The partitions in **bold** are the default partition for the corresponding cluster.
- On the CalcUA clusters, we limit the maximum memory available per allocated CPU
  - If you request more memory, the number of allocated CPUs will be increased accordingly

Slurm resource requests
Faster communication

Long option	Description
`--switches=1`	Request all nodes to be connected to a single switch

- The communication network of the CalcUA clusters has a tree structure
    - Nodes are grouped on a number of edge switches
        - up to 44 nodes per switch on Vaughan, 24 nodes per switch on Leibniz, and all nodes in a single switch on Breniac
    - These switches are connected through a second level of switches, the top switches
    - Hence traffic between two nodes either passes through just a single switch or through three switches : edge-x – top-y – edge-z
- Some programs (e.g.: GROMACS) are extremely latency-sensitive and run much better if they only get nodes connected to a single switch
    - But this will increase your waiting time…

Non-resource-related options

Job name

Long option	Short option	Job environment variable	Description
`--job-name=`*<jobname>*	`-J` *<jobname>*	`SLURM_JOB_NAME`	Name of the job

- It is possible to assign a name to your job
  - The job name can be used when defining the output and error files
- If not given, the **default name** is set to be the name of the batch script
  - or just "sbatch" in the script is read from standard input

Non-resource-related options

Redirect standard output and error

Long option	Short option	Description
`--output=`*`<outfile>`*	`-o` *`<outfile>`*	Redirect stdout
`--error=`*`<errfile>`*	`-e` *`<errfile>`*	Redirect stderr

- By **default**, Slurm redirects both stdout and stderr to the file `slurm-<jobid>.out`
  - And that file is present as soon as the job starts and produces output
- If only `--output` is given, it will redirect both stdout and stderr to the given file
- It is possible to insert codes when specifying the file name that will be replaced at runtime with the corresponding Slurm information
  - Examples are **`%x`** for the job name or **`%j`** for jobid

- For a full list of codes, see [Slurm] sbatch manual page (section "filename pattern")

VLAAMS SUPERCOMPUTER CENTRUM

Non-resource-related options

Mail notifications

Long option	Description
`--mail-type=<type>`	Event notification (start, end, …)
`--mail-user=<email>`	Email address

- The cluster can notify you by mail when a job starts, ends or is aborted
  - `BEGIN` : at the start of the job
  - `END` : at the end of the job
  - `FAIL` : when the job fails
  - `ALL` : at a selection of important events
- Also possible after after certain fractions of the requested wall time have expired
  - `TIME_LIMIT_90` : when the job has reached 90 percent of its time limit
- If not specified, the **default** value is the email address associated with the VSC-account of the submitting user.

The job runtime environment
Default minimal environment

- Normally, a job inherits the environment from the shell from which the allocation was made
  - This includes loaded modules, which can be a problem if those modules were not loaded in the context of the compute node – e.g. the Intel MPI modules (`impi/*`) set several extra `I_MPI_*` environment variables when these modules are loaded in a batch job script
  - This can have unexpected results, since you may be working in a different environment than the one you used the previous time, and as a consequence the job script may behave differently
    - There may also be some differences between the login nodes and the compute nodes
- **On UAntwerp clusters, we only set a minimal environment for jobs by default**

```
--export=HOME,USER,TERM,PATH=/bin:/sbin
```

  - Hence you need to (re)build a suitable environment for your job, starting by loading an appropriate software stack module (calcua/*) and the application modules
- [VSC docs] The job environment

The job runtime environment

- Slurm also defines several other variables when a job is started (some are not always present), e.g.:

Environment variable	**Explanation**
`SLURM_SUBMIT_DIR`	The directory from which sbatch was invoked
`SLURM_JOB_ACCOUNT`	Account name selected for the job
`SLURM_JOB_NUM_NODES`	The total number of nodes for the job
`SLURM_JOB_NODELIST`	List of nodes allocated to the job
`SLURM_JOB_CPUS_PER_NODE`	CPUs available to the job on the nodes
`SLURM_TASKS_PER_NODE`	Number of tasks to be initiated on each node
`SLURM_ARRAY_*`	Additional variables for array jobs (see later)

- Also available: the **`VSC_*`** variables (for user directories, but also for cluster/os/architecture) and the various defined by the loaded modules, including the **`EB*`** variables

- For a full list, see [Slurm] manual page of sbatch (section "OUTPUT ENVIRONMENT VARIABLES")

VLAAMS
SUPERCOMPUTER
CENTRUM

Requesting resources – discussion
Other ways for requesting resources

- Slurm has other ways of requesting resources:
  - See the manual page for sbatch (man sbatch)
  - Be very careful when you experiment with those as they will more easily lead to inefficient use of the nodes
    - e.g., you may be allocating resources in a way that a node may only be used for a single job, even if you have more jobs in the queue
- If other options are needed, they will be documented and we will mention them in our mailings.
  - So please, if we send you a mail with suggestions on how to improve your jobs, do not just ignore it, but apply the suggestions (and try to understand why they are an improvement…)
    - Not knowing something because you didn't read our emails is not our fault!

Slurm resource requests

Request exclusive node access

Long option	Description
`--exclusive`	Request exclusive access to the node for the job

- Nodes are shared resources – no single-user per node policy anymore
  - If you don't request all cores, the remaining cores might be used by a job from another user
  - If you submit multiple jobs, those might end up on the same or on different nodes
- But sometimes it can be better if jobs do have exclusive access to compute nodes
  - e.g., parallel jobs can suffer badly if they don't have exclusive access to the full node, because jobs influence each other (L3 cache, memory bandwidth, communication channels, ….)
- You can request exclusive node access for your job with `--exclusive`
  - This prevents sharing of allocated nodes with other jobs (even from the same user)
  - But be aware that all CPUs will be allocated and thus that you will be charged for a full node (even if you only use a single core…)

Slurm resource requests

Bundling tasks

Long option	Short option	Job environment variable	Description
`--nodes=`*<number>*	`-N` *<number>*	`SLURM_JOB_NUM_NODES`	Number of nodes

- Nodes are shared resources – no single-user per node policy anymore
- For each task, all of the CPUs for that task are allocated on a single compute node.
  - But if your job has multiple (parallel) tasks, then those might end up either on the same or different compute nodes
    - This depends on whatever is already running on the compute nodes (from you or from another user…)
    - ~~Slurm will do its best to bundle as many jobs and/or tasks from the same user on as few nodes as possible~~
- It often is a good idea to bundle tasks from the same job on as few nodes as possible
  - e.g., if these tasks are part of a hybrid job, you preferrably want to make the communication latency between these tasks as small as possible
- You can limit the number of nodes the job may use (will get allocated) with **`--nodes`**
  - You could also specify a min/max number of nodes using `--nodes=<min>-<max>`

Slurm resource requests

- Full list of options in the sbatch manual page (man sbatch)
  - We discussed the most common ones.

- Remember that available resources change over time as new hardware is installed and old hardware is decommissioned.
  - Share your scripts
  - But understand what those scripts do (and how they do it)
  - And don't forget to check the resources from time to time!
  - And realise that in parallel computing the optimal set of resources also depends on the size of the problem you are solving!

Share experience

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

7 – Slurm commands

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

Slurm commands

Overview

Command	Description
`sbatch`	Submit a batch script
`srun`	Run parallel tasks
`salloc`	Create a resource allocation
`squeue`	Check the status of your jobs
`scancel`	Cancel a job
`sstat`	Information about running jobs
`sacct`	Information about (terminated) jobs
`sinfo`	Get an overview of the cluster and partitions
`scontrol`	View current Slurm configuration and state

Slurm commands – sbatch

Submit a batch script

- `sbatch <sbatch arguments> jobscript <arguments of the job script>`
  - Exits immediately when the job is submitted, so it does not wait for the job to start or end
  - Can also read the job script from stdin instead
- What `sbatch` does:
  - Submits the job script to the selected partition (aka job queue)
  - Returns the jobid in a sentence: `Submitted batch job <jobid>`
    - To make sbatch return only the jobid for a successfully submitted script, use **`--parsable`**
- What Slurm then does after `sbatch` returns
  - Creates the allocation when resources become available
  - Creates the batch job step in which it runs the batch script, passing the arguments to the job script

- [Slurm] sbatch manual page

VLAAMS
SUPERCOMPUTER
CENTRUM

Slurm commands – srun

Run parallel tasks

- `srun` can be used in a job script to start parallel tasks
  - It can also be used from the login nodes with the same command line options as for `sbatch` (using `--pty`) and will then request an allocation before running the tasks
    - but the results may be unexpected, in particular with MPI programs
- In Slurm terminology, `srun` creates a job step that can run one or more parallel tasks
  - For advanced usage, it is possible to run multiple job steps simultaneously, each using a part of the allocated resources
- It is the "Swiss Army Knife" in Slurm to create and sometimes manage tasks within a job
  - The best way of starting MPI programs in Slurm jobs (preferred over using `mpirun`)
- Best shown through examples later in this tutorial, but it is impossible to cover all possibilities

- [VSC docs] Starting multiple copies of a process in a job script: srun
- [Slurm] srun manual page

Slurm commands – salloc

Create a resource allocation

- `salloc` creates a resource allocation
  - Very useful for interactive work, but you have to realise very well what you're doing
- What `salloc` does:
  - It request the resources, either specified on the command line or through environment variables, and waits until they are allocated
  - Then starts a shell on the node where you executed salloc (usually the login node)
    - And this is the confusing part, as most likely the shell prompt will look identical to the one you usually get, so you won't realise you're still working in the allocation…
  - It releases the resources when you exit the shell or when the wall time expires
- Important: **the shell is not running on the allocated nodes!**
  - But from the shell, you can then start job steps on the allocated resources using `srun`

- [Slurm] salloc manual page

Slurm commands – squeue
Check the status of your jobs

- `squeue` checks the status of your own jobs in the job queue:

```
$ squeue
     JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
     26170      zen2     bash vsc20259  R       6:04      1 r1c01cn4
```

  - The column ST shows the **state** of the job. Some popular ones are:

ST	Explanation
PD	Pending – waiting for resources
CF	Configuring – nodes becoming ready
R	Running
CD	Successful completion – exit code zero

ST	Explanation
F	Failed job – non-zero exit code
TO	Timeout – terminated on wall time limit
OOM	Job experienced out-of-memory error
NF	Job terminated due to node failure

- For a full list of job state codes, see [Slurm] squeue manual page (section "JOB STATE CODES")

VLAAMS
SUPERCOMPUTER
CENTRUM

Slurm commands – squeue
Check the status of your jobs

- `squeue` checks the status of your own jobs in the job queue:

```
$ squeue
     JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
     26170      zen2     bash vsc20259  R       6:04      1 r1c01cn4
```

  - The column NODELIST(REASON) shows the **reason** why a job is waiting for execution, e.g.:

Reason	**Explanation**
Priority	There are one or more higher priority jobs in the partition
QOSMaxNodePerUserLimit	The limit on the maximum number of nodes per user will be exceeded
AssocMaxJobsLimit	The limit on the number of running jobs for each user has been reached
JobHeldAdmin	The job is held by an administrator

- For a full list of reasons, see [Slurm] squeue manual page (section “JOB REASON CODES”)

Slurm commands – squeue

Get more information

- Getting more information:
  - `--jobs` or `-j` : request a comma separated list of jobids to display
  - `--steps` or `-s` : also include the currently active job steps for each job
  - `--long` or `-l` : reports slightly more information (also the requested wall time)
  - `--format` or `-o` : specify the information to be displayed, its size and position
    - e.g., the default is: `"%.18i %.9P %.8j %.8u %.2t %.10M %.6D %R"`
  - `--Format` or `-O` : request a comma separated list of job information to be displayed
    - e.g.: `"JobID:.18 ,Partition:.9 ,Name:.8 ,UserName:.9 ,` […]`"`
- Getting an **estimate** for the start time of jobs : `squeue --start`
  - But jobs may start later because higher priority jobs enter the queue
  - Or they may start sooner because other jobs earlier than their specified wall time

- [Slurm] squeue manual page

VLAAMS
SUPERCOMPUTER
CENTRUM

Slurm commands – scancel

Cancel a job

- `scancel <jobid>` cancel a single job, and all its job steps (if already running)
  - For a job array it is also possible to just kill some jobs of the array
  - It is also possible to kill only a specific job step : `scancel <jobid>.<stepid>`
    - e.g., if you suspect an MPI program hangs, but you still want to execute the remainder of the job script to clean up and move results
- Some frequently used options:
  - `--state <state>` or `-t <state>`: cancel only jobs which have the given state
    - `<state>` can be `pending`, `running`, or `suspended`
  - `--partition <part>` or `-p <part>`: cancel only jobs in the given partition
- [Slurm] scancel manual page

Slurm commands – sstat

Information about running jobs

- `sstat -j <jobid>[.<stepid>]` Show (a lot of) real-time information about a job or job step
  - It is possible to specify a subset of fields to display using the `-o`, `--format` or `--fields` option.
- Example for an MPI job, to get an idea of the load balancing:

```
$ sstat -j 12345 -o JobID,MinCPU,AveCPU
      JobCPU      MinCPU      AveCPU
------------ ---------- ----------
12345.extern  00:00.000  00:00.000
12345.batch   00:00.000  00:00.000
12345.0        22:54:20   23:03:50
```

  - shows for each job step the minimum and average amount of consumed CPU time
    - here, step 0 is an MPI job, and we see that the minimum CPU time consumed by the task is close to the average, which indicates that the job may be running fairly efficiently and that the load balance is likely OK

Slurm commands – sstat

Information about running jobs

- Checking resident memory usage:

```
$ sstat -a -j 12345 -o JobID,MaxRSS,MaxRSSTask,MaxRSSNode
       JobID     MaxRSS MaxRSSTask MaxRSSNode
------------ ---------- ---------- ----------
12345.extern
12345.batch       4768K          0 r1c06cn3.+
12345.0         708492K         16 r1c06cn3.+
```

  - shows that the largest process in the MPI job step is consuming roughly 700MB at the moment, and it is task 16 and running on r1c06cn3.vaughan

- [Slurm] sstat manual page.

VLAAMS SUPERCOMPUTER CENTRUM

Slurm commands – sacct

Information about (terminated) jobs

- `sacct` shows information kept in the job accounting database
  - For running jobs the information may enter only with a delay

```
$ sacct -j 12345
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
12345         NAMD-S-00+       zen2 antwerpen+         64  COMPLETED      0:0
12345.batch        batch            antwerpen+         64  COMPLETED      0:0
12345.extern      extern            antwerpen+         64  COMPLETED      0:0
12345.0            namd2            antwerpen+         64  COMPLETED      0:0
```

- Select what you want to see:
  - `--brief` : very little output, just the state and exit code of each step
  - `--long` : a lot of information, even more than `sstat`
  - `-o` or `--format` : specify the columns you want to see

Slurm commands – sacct

Information about (terminated) jobs

- Example: Get resource use of a job:

```
$ sacct -j 12345 --format JobID,AllocCPUS,MinCPU%15,AveCPU%15,MaxRSS,AveRSS --units=M
       JobID  AllocCPUS          MinCPU          AveCPU     MaxRSS     AveRSS
------------ ---------- --------------- --------------- ---------- ----------
12345                64
12345.batch          64        00:00:00        00:00:00      4.62M      4.62M
12345.extern         64        00:00:00        00:00:00          0          0
12345.0              64     11-03:40:46     11-03:40:46  28014.53M  28014.53M
```

  - This was a single node shared memory job, so the memory consumption per task is high.
    - `--units=M` to get output in megabytes rather than kilobytes
  - `%15` in some field names: use a 15 character wide field rather than the standard width
  - `--helpformat` or `-e` : list all possible fields

VLAAMS SUPERCOMPUTER CENTRUM

Slurm commands – sacct

Information about (terminated) jobs

- Selecting jobs to show information about:
  - `--jobs` or `-j` : give information about a specific job or a comma separated list of jobs
    - if not specified, show all jobs that have run since midnight
  - `--starttime=`*`<time>`* or `-S` *`<time>`*: jobs that have been running since the indicated start time
  - `--endtime=<time>` or `-E <time>` : Jobs that have been running before the indicated end time
- Accepted time formats (where [] denotes an optional part, and | the alternatives):

```
HH:MM[:SS] [AM|PM] or MMDD[YY] or MM/DD[/YY] or MM.DD[.YY] or
MM/DD[/YY]-HH:MM[:SS] or YYYY-MM-DD[THH:MM[:SS]]
```

  - also possible: `now[{+|-}<count>[seconds(default)|minutes|hours|days|weeks]]` or …
- Print the batch script of a job : `sacct -j` *`<jobid>`* `-B` (or `--batch-script`)

- [Slurm] sacct manual page

Slurm commands – sinfo

Get an overview of the cluster

- `sinfo` show information about the partitions and their nodes in the cluster

```
$ sinfo
PARTITION     AVAIL  TIMELIMIT  NODES  STATE NODELIST
zen2             up 3-00:00:00     38    mix r1c01cn1.vaughan, ...
zen2             up 3-00:00:00    112  alloc r1c01cn2.vaughan, ...
zen2             up 3-00:00:00      1   idle r4c05cn2.vaughan
zen3             up 3-00:00:00     24  idle~ r6c01cn1.vaughan, ...
broadwell        up 3-00:00:00      2  down~ r2c08cn1.leibniz, ...
ampere_gpu       up 1-00:00:00      1   idle nvam1.vaughan
```

- By default, show information per partition
  - the maximum wall time limit for jobs in the given partition
  - the number of allocated/mixed/idle/down nodes
    - ~ indicates that the node is currently in powersave mode (tuned profile)

VLAAMS SUPERCOMPUTER CENTRUM

Slurm commands – sinfo

Get an overview of the cluster

- Node-oriented overview, e.g. (trimmed):

```
$ sinfo -N -l -n r6c01cn4.vaughan,r1c02cn3.leibniz,amdarc2.vaughan
NODELIST            NODES    PARTITION        STATE CPUS    S:C:T MEMORY AVAIL_FE
amdarc2.vaughan         1 arcturus_gpu         idle 64     2:32:1 245760 gpu,amd_
r1c02cn3.leibniz        1    broadwell    allocated 28     2:14:1 114688 leibniz,
r6c01cn4.vaughan        1         zen3        idle~ 64     2:32:1 245760 vaughan,
```

  - MEMORY column : shows the memory that can be allocated (in kilobytes)
  - `S:C:T` column : shows the structure of the node w.r.t. sockets/cores/(hardware) threads
    - e.g.: `2:32:1` means 2 sockets with 32 cores per socket and 1 hardware thread per physical core
      - remember, on UAntwerp clusters: CPU = core
  - `AVAIL_FE` column: shows the available features

- [Slurm] sinfo manual page

VLAAMS SUPERCOMPUTER CENTRUM

Slurm commands – scontrol

View Slurm configuration and state

- `scontrol` view (or modify) Slurm configuration and state
  - Mostly for administrators, but the show subcommand is useful for regular users also
- **`scontrol show`** : show informaton about jobs (and job steps), nodes, partitions, reservations, …
  - `scontrol -d show job` *`<jobid>`*: show detailed information about a running job
    - also shows the `CPU_IDs` set of CPUs that have been assigned to the job
  - `scontrol show hostnames` : get a list of node names in a job script, one per line
    - SLURM_JOB_NODELIST contains the same list but in one line, separated by commas
  - `scontrol show part` *`[<part>]`*: show information about the partitions
  - `scontrol show config` : show the current Slurm configuration settings
- [Slurm] scontrol manual page

VLAAMS
SUPERCOMPUTER
CENTRUM

HPC@UAntwerp introduction

Exercises – block 2

Exercises – block 2

- See [VSC docs] Job submission
- `$VSC_SCRATCH/examples/Slurm/04_Running-batch-jobs` contains a short bash script (`fibo.slurm`) that will print Fibonacci numbers
  - Extend the job script: include a request for 1 core on 1 node, with 1 Gb of memory and with a wall time of 10 minutes
  - Run the job, and note which files are generated
  - Extend the script: give the job a name, and change the output and error formats
  - Run and note which files are generated
  - While the job is running, try some of the `squeue` and `sstat` or `sacct` commands
    - Note: you might want to add a sleep 300 at the end of the script, because otherwise the job will finish so quickly that you don't get the chance to see anything...
- From now on, use the project account ap_course_hpc_intro with the commands, e.g.

```
sbatch -A ap_course_hpc_intro
```

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

8 – Working interactively

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

GUI and visualisation support

- The cluster is not your primary GUI working platform, but sometimes running GUI programs is necessary:
  - Licensed program with a GUI for setting parameters of a run that cannot be run locally due to licensing restrictions (e.g., NUMECA FINE-Marine)
  - Visualisations that need to be done while a job runs or for data that is too large to transfer
- Technologies in Linux:
  - **X Window System** : rendering on your local computer – needs an X server on your PC
    - Sends the drawing commands from the application (X client) to the display device (X server)
    - Extensions add features, e.g., GLX for OpenGL support
    - Often too slow due to network latency
  - **VNC** (Virtual Network Computing) : rendering on the remote computer – needs a VNC client on your PC
    - Rendering on the remote machine by the VNC server, images compressed and sent over the network
    - Works pretty well even over not very fast connections, but hard to set up
  - NoMachine NX / X2Go : sits between those options
    - Not supported at UAntwerp, but used on the clusters at KU Leuven and UGent

Using the visualisation node

- Leibniz has one **visualisation node** : viz1.leibniz
  - Has a NVIDIA Quadro Pascal P5000 GPU and a VNC & VirtualGL-based setup
  - VNC is the software to create a remote desktop
    - Acts as a X server to programs running locally
  - Simple desktop environment with the Xfce desktop/window manager
    - GNOME may be more popular, but does not work well with VNC
  - **VirtualGL** : redirect OpenGL calls to the graphics accelerator and send the resulting image to the VNC software
- Same software stack minus the OpenGL libraries on the login nodes
  - For 2D applications or applications with built-in 3D emulation (e.g., Matlab)
- Need a VNC client on your PC, such as TurboVNC or TigerVNC
- [VSC docs] Remote visualisation @ UAntwerp

Running GUI programs – vglrun

- Using OpenGL:
  - The VNC server currently emulates an X-server without the GLX extension, so OpenGL programs will not run right away.
  - On the visualisation node: Start OpenGL programs through the command **`vglrun`**
    - `vglrun` will redirect OpenGL commands from the application to the GPU on the node
    - and transfer the rendered 3D image to the VNC server
  - e.g.: `vglrun glxgears` will run a sample OpenGL program
- Our default desktop for VNC is **`Xfce`**
  GNOME is not supported due to compatibility problems

Interactive jobs

Method 1: srun for a non-X11 job

- Use the regular resource request options on the command line of srun and end with `--pty bash`
- Example: An interactive session to run a shared memory application

```
login$ srun -n 1 -c 16 -t 1:00:00 --pty bash
rXcYYcnZ$ module --force purge
rXcYYcnZ$ ml calcua/2020a vsc-tutorial
rXcYYcnZ$ omp_hello
…
rXcYYcnZ$ exit
```

- Example: Starting an MPI program in an interactive session

```
login$ srun -n 64 -c 1 -t 1:00:00 --pty bash
rXcYYcnZ$ module --force purge
rXcYYcnZ$ ml calcua/2020a vsc-tutorial
rXcYYcnZ$ srun --overlap mpi_hello
…
rXcYYcnZ$ exit
```

Interactive jobs

Method 1: srun for an X11 job

- First make sure that your login session supports X11 programs:
  - Log in to the cluster using `ssh -X` to forward X11 traffic
  - Or work from a terminal window in a VNC session
- Same as for non-X11 jobs but simply add the `--x11` option before `--pty bash`
- Few or no X11 programs support distributed memory computing, so usually you'll only be using one task…

```
login$ srun -n 1 -c 64 -t 1:00:00 --x11 --pty bash
rXcYYcnZ$ module --force purge
rXcYYcnZ$ ml calcua/2020a …
rXcYYcnZ$ xclock
rXcYYcnZ$ exit
```

- You can even start X11 programs directly through srun, e.g.,

```
login$ srun -n 1 -c 1 -t 1:00:00 --x11 xclock
```

but this has the same problems as the next approach…

Interactive jobs

Method 2: salloc for a shared memory program

- Here you have to really understand how Linux environments work.
  - This method will have to change if the compute nodes and login nodes have a different architecture.
- Example for a shared memory program:

```
login$ salloc -n 1 -c 64 -t 1:00:00
login$ module --force purge
login$ ml calcua/2020a vsc-tutorial
login$ srun omp_hello
…
login$ exit
```

Problem: Software for the login nodes may not work on the compute nodes or vice-versa

VLAAMS SUPERCOMPUTER CENTRUM

Interactive jobs

Method 2: salloc for a distributed memory program

- Example for an MPI program

```
login$ salloc -n 64 -c 1 -t 1:00:00
login$ module --force purge
login$ ml calcua/2020a vsc-tutorial
login$ srun mpi_hello
…
login$ exit
```

Problem: Software for the login nodes may not work on the compute nodes or vice-versa

VLAAMS SUPERCOMPUTER CENTRUM

Interactive jobs

Method 2: `salloc` for running X11 programs

- First make sure that your login session supports X11 programs:
  - Log in to the cluster using `ssh -X` to forward X11 traffic
  - Or work from a terminal window in a VNC session
- Next use salloc to ask for an allocation
  - Note: it usually doesn’t make sense to use more than 1 task when running X11 programs

```
login$ salloc -n 1 -c 64 -t 1:00:00 --mem-per-cpu=3g
```

- From the login shell in your allocation, log in to the compute node using

```
login$ ssh -X $SLURM_NODELIST
```

- You are now on the compute node in your home directory (because of ssh) and can now load the modules you need and start the programs you want to use.

Interactive jobs

Method 2: salloc for running X11 programs

- First make sure that your login session supports X11 programs:
  - Log in to the cluster using `ssh -X` to forward X11 traffic
  - Or work from a terminal window in a VNC session
- Then:

```
login$ salloc -n 1 -c 64 -t 1:00:00
login$ ssh -X $SLURM_NODELIST
rXcYYcnZ$ xclock
…
rXcYYcnZ$ exit
login$ exit
```

HPC@UAntwerp introduction

9 — Multi-core parallel jobs

Vlaanderen
is supercomputing

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Why parallel computing?

1. Faster time to solution
   - distributing code over N cores
   - hope for a speedup by a factor of N
2. Larger problem size
   - distributing your code over N nodes
   - increase the available memory by a factor N
   - hope to tackle problems which are N times bigger
3. In practice
   - gain limited due to communication and memory overhead and sequential fractions in the code
   - optimal number of cores/nodes is problem-dependent
   - but no escape possible as computers don't really become faster for serial code...

➢ **Parallel computing is here to stay.**

Running a shared memory job

- A shared memory job involves a single task with multiple CPUs per task
- Shared memory programs start like any other program, but you will likely need to tell the program how many threads it can use (unless it can use the whole node).
  - Depends on the program, and the autodetect feature of a program usually only works when the program gets the whole node.
    - e.g., for MATLAB, use `maxNumCompThreads(N)`
  - Many OpenMP programs use the environment variable **`OMP_NUM_THREADS`**
    - Intel OpenMP recognizes Slurm CPU allocations
  - For MKL-based code, **`MKL_NUM_THREADS`** can overwrite OMP_NUM_THREADS for MKL operations
  - OpenBLAS (FOSS toolchain): **`OPENBLAS_NUM_THREADS`**
- **Check the manual of the program you use!**
  - e.g., NumPy has several options depending on how it was compiled…

Running a shared memory job

- Example script :

generic-omp.slurm

```
#!/bin/bash
#SBATCH --job-name=OpenMP-demo
#SBATCH --ntasks=1 --cpus-per-task=64        ← 1 task with 64 CPUs (so 64 threads)
#SBATCH --mem=2g                             ← 2 gb per CPU, so 128 GB total memory
#SBATCH --time=00:05:00
module --force purge
module load calcua/2020a                     ← load the software stack module
module load vsc-tutorial                     ← load vsc-tutorial, which loads the Intel
                                               toolchain (for the OpenMP run time)
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK  ← Set the number of threads to use
export OMP_PROC_BIND=true                    ← Will cause the threads to be nicely spread
                                               over the cores and stay on the core
omp_hello                                    ← Run the program
```

Running a MPI job

- Every distributed memory program needs a program starter as it uses more than one process
  - Some packages come with their own command to launch the program that uses the system starter internally
  - Check the manual, it may have an option to explicitly set the program starter that is used internally
- Preferred program starter for Slurm: **`srun`**
  - It knows best about how Slurm wants to distribute processes across nodes and CPUs
  - It usually needs no further arguments if you requested resources in the right way, i.e. with `--ntasks` equal to the number of MPI ranks and `--cpus-per-task=1`
- `mpirun` will work but it does depend on variables that are set in the intel module
  - So ensure that the module is reloaded in your job script as those variables are not set on the login nodes
  - Don't set `I_MPI_HYDRA_BOOTSTRAP` or `I_MPI_HYDRA_RMK` in your job script!

Running a MPI job

➢ (Intel MPI) example script :

generic-mpi.slurm

```
#!/bin/bash
#SBATCH --job-name mpihello
#SBATCH --ntasks=128 --cpus-per-task=1
#SBATCH --mem-per-cpu=1g
#SBATCH --time=00:05:00
module --force purge
module load calcua/2020a
module load vsc-tutorial

srun mpi_hello
```

← 128 MPI processes, i.e., 2 nodes with 64 processes each on Vaughan

← load the software stack module

← load vsc-tutorial, which loads the Intel toolchain (for the MPI libraries)

← run the MPI program (mpi_hello) srun communicates with the resource manager to acquire the number of nodes, cores per node, node list etc.

VLAAMS SUPERCOMPUTER CENTRUM

Running a hybrid MPI job

- We need no additional tools to start hybrid programs.
  - So, no need for torque-tools or vsc-mympirun or other tools you may find in old job scripts or VSC web pages (vsc-mympirun is still being used on some VSC sites)
  - **`srun` does all the miracle work**
    - or `mpirun` in Intel MPI, provided the environment is set up correctly

- Example (next slide)
  - 8 MPI processes (tasks)
  - Each MPI process has 16 threads
    - that is two full Vaughan compute nodes
  - In fact, the OMP_NUM_THREADS line isn't needed with most programs that use the Intel OpenMP implementation

Starting a hybrid MPI job on Slurm (2)

generic-hybrid.slurm

```
#!/bin/bash
#SBATCH --job-name hybrid_hello
#SBATCH --ntasks=8 --cpus-per-task=16
#SBATCH --mem-per-cpu=1g
#SBATCH --time=00:05:00
module --force purge
module load calcua/2020a
module load vsc-tutorial
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_PROC_BIND=true
srun mpi_omp_hello
```

- ← 8 MPI processes with 16 threads each, i.e., 2 nodes with 64 processes each on Vaughan
- ← load the software stack module
- ← load vsc-tutorial, which also loads the Intel toolchain (for the MPI libraries)
- ← Set the number of OpenMP threads
- ← Will cause the threads to be nicely spread over the cores and stay on the core
- ← run the MPI program (mpi_omp_hello) srun does all the magic, but mpirun would work too with Intel MPI

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

10 – Organizing job workflows

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Examples of job workflows

- Some scenarios:
  - Need to run a sequence of simulations, each one using the result of the previous one, but bumping into the 3-day wall time limit so not possible in a single job
    - Or need to split a >3 day simulation in shorter ones that run one after another
  - Need to run simulations using results of the previous one, but with a different optimal number of nodes
    - e.g., in CFD: first a coarse grid computation, then refining the solution on a finer grid
  - Extensive sequential pre- or postprocessing of a parallel job
  - Run a simulation, then apply various perturbations to the solution and run another simulation for each of these perturbations
- Support in Slurm
  - Passing environment variables to job scripts: Can act like arguments to a procedure
    Alternative: Passing command line arguments to job scripts
  - Specifying dependencies between job scripts with additional sbatch arguments

Passing environment variables to job scripts

- On UAntwerp clusters, we only pass a minimal environment to the job
  - So we need to make sure that variables that need to be passed are explicitly exported as otherwise `sbatch` will not see them
  - `--export=`*<myenv>* : propagates the myenv environment variable to the job environment
  - `--export=`*<myenv1>=<value1>*, *<myenv2>* : sets the myenv1 environment variable to the given value1 and propagates both myenv1 and myenv2 (which elready has a value) to the job environment.
  - Note that SLURM_* variables are always propagated
- Remember you can also pass environment variables to a command on the command itself, e.g.,

```
multiplier=5 sbatch --export=multiplier my_job_script.slurm
```

will pass the environment variable multiplier with value 5 to sbatch

VLAAMS
SUPERCOMPUTER
CENTRUM

Passing command line arguments to job scripts

- Any command line argument after the name of the job script is considered to be a command line argument for the job batch script and passed to it as such
  - Create the batch script:

get_parameter.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=500m
#SBATCH --time=5:00

echo "The command line argument is $1."
```

  - Now run:

```
sbatch get_parameter.slurm 5
```

  - The output file contains:

```
The command line argument is 5.
```

Dependent jobs

- It is possible to instruct Slurm to only start a job after finishing one or more other jobs:

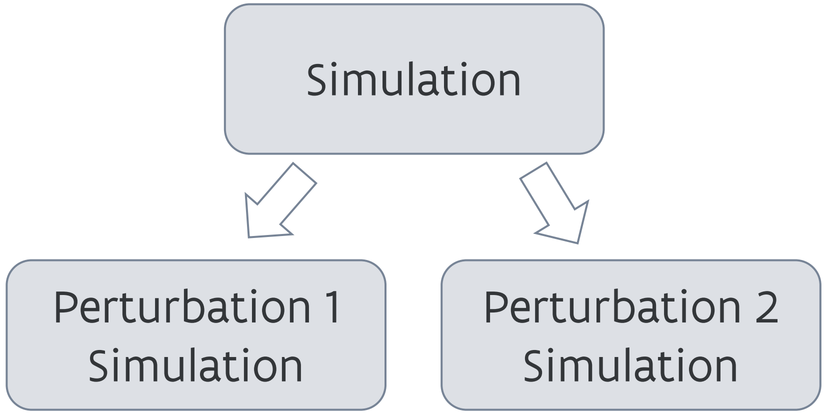
```
sbatch --dependency=afterok:<jobid> jobdepend.slurm
```

  will only start the job defined by `jobdepend.slurm` after the job with job ID <jobid> has finished successfully

- Many other possibilities, including
  - Start another job only after a a list of jobs have ended
  - Start another job only after a job has failed
  - And many more, check the sbatch manual page and look for `--dependency`.

- Useful to organize jobs and powerful in combination with passing environment variables or specifying command line arguments to job scripts

Dependent jobs
Example



job.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=1g
#SBATCH --time=30:00

echo "10" >outputfile ; sleep 300

multiplier=5
mkdir mult-$multiplier ; pushd mult-$multiplier
number=$(cat ../outputfile)
echo $(($number*$multiplier)) >outputfile; sleep 300
popd

multiplier=10
mkdir mult-$multiplier ; pushd mult-$multiplier
number=$(cat ../outputfile)
echo $(($number*$multiplier)) >outputfile; sleep 300
```

Dependent jobs
Example

- A job whose result is used by 2 other jobs

job_first.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=1g
#SBATCH --time=10:00

echo "10" >outputfile ; sleep 300
```

job_depend.slurm

```
!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=1g
#SBATCH --time=10:00

mkdir mult-$multiplier ; cd mult-$multiplier
number=$(cat ../outputfile)
echo $(($number*$multiplier)) >outputfile; sleep 300
```

job_launch.sh

```
#!/bin/bash
first=$(sbatch --parsable --job-name job_leader job_first.slurm)
multiplier=5  sbatch --job-name job_mult_5  --dependency=afterok:$first job_depend.slurm
multiplier=10 sbatch --job-name job_mult_10 --dependency=afterok:$first job_depend.slurm
```

Dependent jobs
Example

- After start of the first job – squeue

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
24869      zen2 job_mult vsc20259 PD       0:00      1 (Dependency)
24870      zen2 job_mult vsc20259 PD       0:00      1 (Dependency)
24868      zen2 job_lead vsc20259  R       0:25      1 r1c01cn1
```

- Some time later:

```
JOBID  PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 24869      zen2 job_mult vsc20259  R       0:01      1 r1c01cn1
 24870      zen2 job_mult vsc20259  R       0:01      1 r1c01cn1
```

- When finished: Output of ls:

```
job_depend.slurm  job_launch.sh  mult-10  outputfile      slurm-24869.out
job_first.slurm   job.slurm      mult-5   slurm-24868.out slurm-24870.out
```

-
  ```
  cat outputfile                10
  cat mult-5/outputfile         50
  cat mult-10/outputfile        100
  ```

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

11 – Multi-job submission

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Running a large batch of small jobs

- Many users run many serial jobs, e.g., for a parameter exploration
- Problem: submitting many short jobs to the queueing system puts quite a burden on the scheduler, so try to avoid it

- Solutions:
  - **Job arrays** : submit a large number of related jobs at once to the scheduler
    - `atools`, a (VSC-developed) package that makes managing array jobs easier
  - ~~Worker framework does not work with Slurm~~
  - Slurm `srun` can be used to launch more tasks than requested in the job request running no more than the indicated number simultaneously
    - Contact us if you have a case where you think you can use that
  - GNU `parallel` (module name: parallel) can work with Slurm, but you'll have to combine it with `srun` and more advanced options than we can cover here.
    - Like the previous bullet, but using GNU parallel to generate arguments and input for the commands

Job arrays

- Starts from a job script for a single job in the array:

job_array.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=512M
#SBATCH --time 15:00

INPUT_FILE="input_${SLURM_ARRAY_TASK_ID}.dat"
OUTPUT_FILE="output_${SLURM_ARRAY_TASK_ID}.dat"

./test_set -input ${INPUT_FILE} -output ${OUTPUT_FILE}
```

← for every run, there is a separate input file and an associated output file

```
$ sbatch --array 1-100 job_array.slurm
```

- program will be run for all input files (100)

Job arrays – atools

Example: Parameter exploration

- Assume we have a range of parameter combinations we want to test in a .csv file (easy to make with Excel)
- Help offered by atools:
  - *How many jobs should we submit?* atools has a command that will return the index range based on the .csv file
  - *How to get parameters from the .csv file to the program?* atools offers a command to parse a line from the .csv file and store the values in environment variables.
  - *How to check if the code produced results for a particular parameter combination*? atools provides a logging facility and commands to investigate the logs.

Job arrays – atools

Example: Parameter exploration

weather.slurm

```
#!/bin/bash
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --mem-per-cpu=512m
#SBATCH --time=10:00
module --force purge
ml calcua/2020a atools/slurm

source <(aenv --data data.csv)
./weather -t $temperature -p $pressure -v $volume
```

data.csv

```
temperature, pressure, volume
293.0,       1.0e05,   87
...,         ...,      ...
313,         1.0e05,   75
```

(data in CSV format)

```
login$ module load atools/slurm
login$ sbatch --array $(arange --data data.csv) weather.slurm
```

- weather will be run <u>for all data</u>, until all computations are done
- Can also run across multiple nodes

VLAAMS SUPERCOMPUTER CENTRUM

Job arrays – atools
Remarks

- Atools has a nice logging feature that helps to see which work items failed or did not complete and to restart those.
- Advanced feature of atools: Limited support for some Map-Reduce scenarios:
  - Preparation phase that splits up the data in manageable chunks needs to be done on the login nodes or on separate nodes
  - Parallel processing of these chunks
  - Atools does offer features to aggregate the results
- Atools is really just a bunch of Python scripts and it does rely on the scheduler to start all work items
  - It supports all types of jobs the scheduler supports
  - But it is less efficient than Worker for very small jobs as worker does all the job management for the work items (including starting them)
  - A version of worker that can be ported to Slurm is under development

Exercises – block 3

- See the exercise sheet and select those that are most relevant to your work on the cluster
- Do not forget to go to `$VSC_SCRATCH` where your files from the previous session are stored!

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

12 – Job monitoring

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

FAQ

- How do I know how much memory my job used?
- How can I check if a job is running properly?
- …

Logging on to the compute node(s)

- While your job is running, you can log on to the compute nodes assigned to the job through ssh from the login nodes
  - Check which compute nodes a job uses : `squeue -j`
- Run **`htop`** on the compute node and check the use of the cores and memory
  - Quit htop: q
  - Only show your own processes: press u, select your userid from the list to the left and press the return key
- `htop` also shows the load average, but this may be misleading
  - load = processes that can run
  - If this number is much higher than the number of cores: You are typically running several shared memory applications each using all cores without realising it.
  - But a load > the number of cores does not always mean something is going wrong
- Also use other commands such as **`vmstat`**, **`pstree`**, **`sar`**, … to check the behaviour on the node

VLAAMS
SUPERCOMPUTER
CENTRUM

Add monitoring in your job script

- The `monitor` module
  - See Tutorial text §11.2.2
  - Start your (shared memory) application through the monitor command
    `monitor ./eat_mem 10`
    would start `eat_mem` with the command line argument 10
  - Usefull command line arguments:
    - `monitor -d 300 ./eat_mem 10`
      print output every 300 seconds
      - It does not make sense to sample a 3-day job every second
    - `monitor -d 300 -n 12 ./eat_mem 10`
      will only show the last 12 values (so roughly the last hour of this job)
- Note: `eat_mem` is available in the vsc-tutorial module

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

13 – Site and scheduling policies

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Some site policies

- Our policies on the cluster:
  - ~~Single user per node policy~~ – nodes are shared resources
  - Priority based scheduling: so not “first come, first get”
    - we reserve the freedom to change the parameters of the priority computation to ensure the cluster is used well
  - Fairshare mechanism: make sure one user cannot monopolise the cluster
  - Accounting @ CalcUA: using a project account will be mandatory **starting 1 March 2024**
- Implicit user agreement:
  - The cluster is valuable research equipment
  - Do not use it for other purposes than your research for the university
    - No cryptocurrency mining or SETI@home and similar initiatives
    - Not for private use
  - And you have to acknowledge the VSC in your publications
- Don’t share your account

Project accounts

- At UAntwerp we monitor cluster use and send periodic reports to group leaders
  - Starting January 1st, 2024, an **active project account** is needed to be able to submit jobs
  - You will be notified when your user account is associated with at least one project account
  - This will be implemented in coordination with your supervisor, so no user action needed
- On the VSC Tier-1 system, you get a compute time allocation (number of node days)
  - This is enforced through project credits
  - A free test ride (motivation required), so-called “Starting Grant”
  - Requesting compute time is done through a project proposal
- On the KU Leuven Tier-2, you need compute credits, which are bought directly via the KU Leuven
  - The charge policy (subject to change) is based upon:
    - a fixed start-up cost
    - the used resources (number and type of nodes)
    - actual duration (used wall time)

Debug & test

- Before starting you should always check :
  - are there any errors in the script?
  - are the required modules loaded?
  - Is the correct executable used?
  - Did you use the right process starter (srun) and start in the right directory?
- Check your jobs at runtime
  - login to a compute node and use commands such as, `top`, `htop`, `vmstat`, `pstree`, `sar`, …
  - `monitor` command (see User Portal information on running jobs)
  - look at `sstat` and compare `MinCPU` and `AvgCPU`
  - alternatively: run an interactive job for the first run of a set of similar runs
- If you will be doing a lot of runs with a package, try to benchmark the software for scaling issues when using MPI
  - I/O issues
    - If you see that the CPU is idle most of the time that might be the problem

Hints & tips

- Job will logon to the first assigned compute node and start executing the commands in the job script
  - don’t forget to load the software with module load
- Why is my job not running?
  - use “`squeue`” or look at the fourth line of output of “`scontrol show job <jobid>`”
  - commands might timeout with overloaded scheduler (rather unlikely recently)
- Submit your job and wait (be patient) …
- Using `$VSC_SCRATCH_NODE` (/tmp on the compute node) might be beneficial for jobs with many small files (even though the bandwidth is much, much lower than on `$VSC_SCRATCH`).
  - But that scratch space is node-specific and hence will disappear when your job ends
  - So copy what you need to your data directory or to the shared scratch
  - `$VSC_SCRATCH_NODE` is available on all partitions, but the capacity may differ.

Warnings

- Avoid submitting many small jobs (in number of cores) by grouping them
  - using a job array
  - or using the atools or the Worker framework
- Runtime is limited by the maximum wall time of the queues
  - for longer wall time, use checkpointing
  - Properly written applications have built-in checkpoint-and-restart options
- Requesting many processors could imply long waiting times (though we're still trying to favour parallel jobs)

what a cluster is not

a computer that will automatically run the code of your (PC) application much faster or for much bigger problems

Vlaanderen
is supercomputing

HPC@UAntwerp introduction

User support

VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

User support

- Try to get a bit self-organized
  - limited resources for user support
  - best effort, no code fixing (AI and bioinformatics codes tend to be a disaster to install…)
  - contact hpc@uantwerpen.be, and be as precise as possible
- VSC documentation on ReadTheDocs: docs.vscentrum.be
- External documentation : slurm.schedmd.com/documentation.html
  - mailing-lists

VSC

www.vscentrum.be

CalcUA

hpc.uantwerpen.be

User support

- mailing-lists for announcements : calcua-announce@sympa.uantwerpen.be
  (for official announcements and communications)
- Every now and then a more formal “HPC newsletter”
  - We do expect that users read the newsletter and mailing list messages!
- contacts :
  - e-mail : hpc@uantwerpen.be
  - phone : +32 3 265 XXXX with XXXX
    3860 (Stefan), 3855 (Franky), 3852 (Kurt), 3879 (Bert), 8980 (Carl), Robin (9229)
  - office : G.309-311 (CMI)

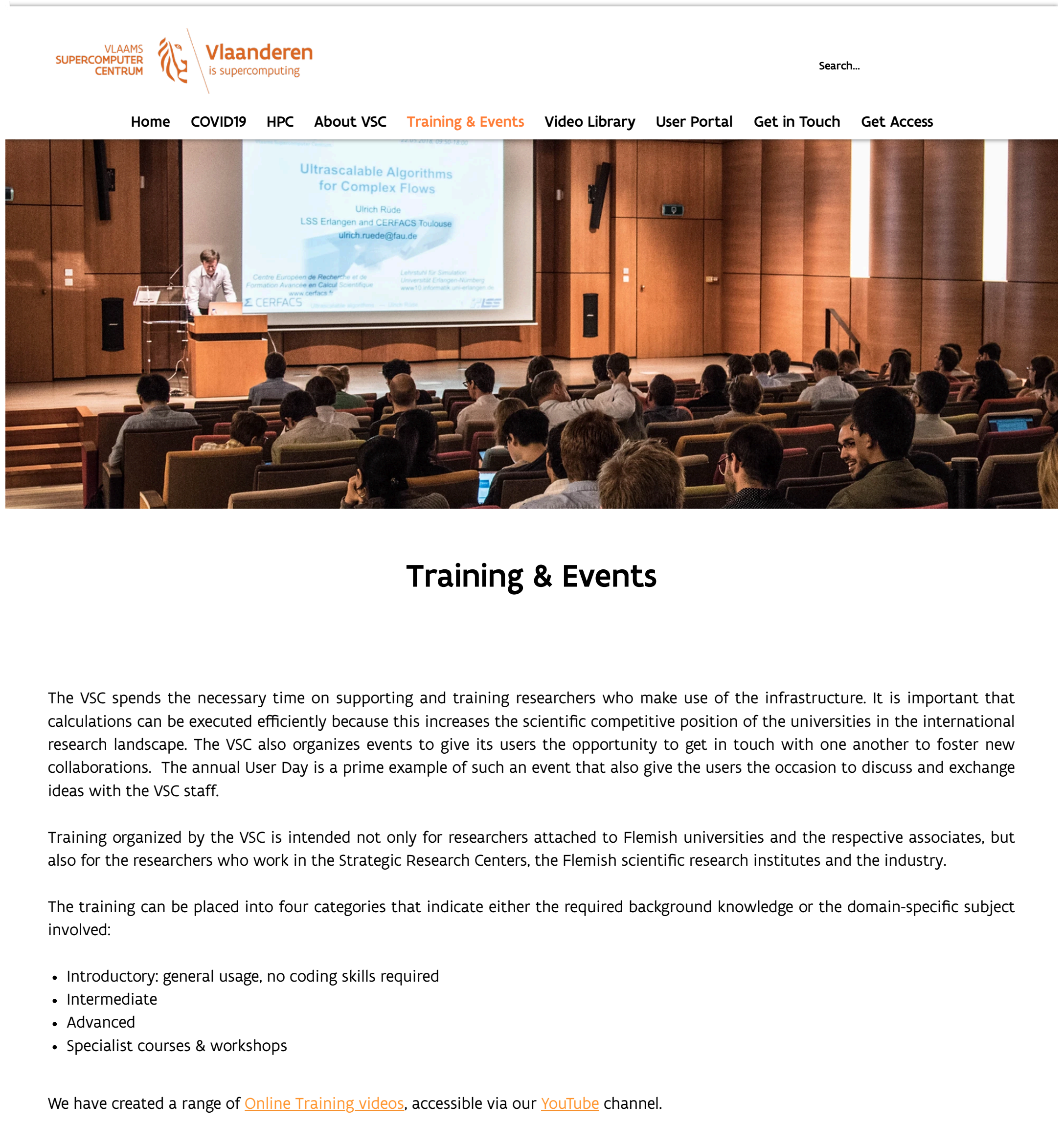
Don’t be afraid to ask, but being as precise as possible when specifying your problem definitely helps in getting a quick answer.

Evaluation

- Please fill in the questionnaire at tiny.cc/calcua-hpc-intro-survey before 17 March

More training:

- ➢ www.vscentrum.be/training



Training & Events

The VSC spends the necessary time on supporting and training researchers who make use of the infrastructure. It is important that calculations can be executed efficiently because this increases the scientific competitive position of the universities in the international research landscape. The VSC also organizes events to give its users the opportunity to get in touch with one another to foster new collaborations. The annual User Day is a prime example of such an event that also give the users the occasion to discuss and exchange ideas with the VSC staff.

Training organized by the VSC is intended not only for researchers attached to Flemish universities and the respective associates, but also for the researchers who work in the Strategic Research Centers, the Flemish scientific research institutes and the industry.

The training can be placed into four categories that indicate either the required background knowledge or the domain-specific subject involved:

- Introductory: general usage, no coding skills required
- Intermediate
- Advanced
- Specialist courses & workshops

We have created a range of Online Training videos, accessible via our YouTube channel.