Ukko2 User Guide

- In the News
  - Directories available now outside of the cluster
  - Filesystem Policies
  - Group Folders
  - Lustre
  - Intel Compilers
- 0.0 General information
- 1.0 Access
  - 1.1 Ukko 1 Resources - Deferred Until Further Notice
  - 1.2 Ukko 2 Resources
  - 1.3 I/O, disks and filesystems
- 2.0 Scientific software
- 3.0 Queuing system
  - 3.1 Creating a Batch job
    - 3.1.1 Environment Variables and Exit Codes
    - 3.1.2 Serial - Consumable resources
    - 3.1.3 Further job control
  - 3.2 Serial or Parallel Job?
    - 3.2.1 Testing and Development
    - 3.2.2 How to request GPU's
    - 3.2.3 Setting up e-mail notifications
  - 3.3 Interactive use
  - 3.4 Advance Reservations
  - 3.5 Actual Resource Utilisation
    - 3.5.1 Job Accounting Data
- 3.0 Further Reading

In the News

Directories available now outside of the cluster

We have added few more locations you can access your working directory from. These assume that you are within the university firewall.

- You can access /wrk/$USER and group directories from the internal network from your workstation with Samba. The samba server is: ukko2-smb.cs.helsinki.fi.
- You access your wrk files in pangolin: /home/ad/ukko2-wrk/$USER
- You can access your project directory in pangolin: /home/ad/turso-proj/$USER

Filesystem Policies

- Working directory $WRKDIR provides very fast I/O performance and it has no quota limit. Unfortunately we have resources to backup $WRKDIR.
- Project directory $PROJ is the long term storage, intended for programs, executables, source codes, results and data sets, but not for working data.
- Temp directory variable $TMPDIR points to a best available location for temporary files on the current system.
- Home directory $HOME is only to store keys, profile files etc. No data storage there. Note that you can redirect most software to use other than ~ as default for cache, temporary files etc. For this purpose, do use $WRKDIR. Note that $HOME has strict quota limit of 10MB.

When creating new jobs and submitting batch jobs, please use $WRKDIR, $TMPDIR and $PROJ as appropriate. You should define your working directory in the slurm script by adding following to the batch script. By default, job inherits the working directory where the batch job was submitted:

```
#SBATCH --workdir=/wrk/<Workdir path> // This defines the Slurm working directory
```

Group Folders

We will create group folders upon request. Group folders are created in two places to provide flexibility. For shared workspace you will have group folder in /wrk. This will provide you good I/O performance. Work group folder is also available through samba to your desktop:

```
/wrk/group/<groupname>
```
For sharing software or common datasets you will have NFS mounted group folder in /proj:

/proj/group/<groupname>

If your research group needs a group folder, please create a request through helpdesk with following information:

- Name of the folder
- Owner of the folder (this individual is responsible for the use)
- IDM group

**Lustre**

If you do not use databases, large, or very large datasets and/or large files in GB/TB scale, you can use the system default settings for $WRKDIR. If you wish to optimize I/O performance, or do I/O intensive tasks, please continue reading to get most out of the resources. If in doubt, we are happy to help you out.

Lustre is now available for Ukko2 users and you have a directory in $WRKDIR. Please do use $WRKDIR for your runtime storage needs, but note that $WRKDIR is a scratch FS and not a long term storage. Please see the Lustre user Guide for additional details.

**Intel Compilers**

Intel compiler suite, Parallel Studio XE Cluster Edition with 2 concurrent floating licenses is now available as a module on Ukko2. Please do note that if you are student you may be eligible for a free license.

**0.0 General information**

All publications produced using Computational resources must include a reference to the infrastructure. The reference to be used is the persistent identifier given to the infrastructure (urn:nbn:fi:research-infras-2016072533) through the Research Infrastructures service. Same applies also to processed data that you publish using data archives. This is very important for the future resource availability.

**1.0 Access**

To use the cluster, you need to either:

- be member of CS staff or
- have your research IDM group as member of grp-cs-ukko2 IDM group

Please note that it might take up to 2 hours after you have been added to either group before your home directory gets created. If you do not have access, but would like to have one, please request the manager/owner of your IDM group to send note to helpdesk(at)helsinki.fi, specifying which group needs to have access to the clusters. Please note that keeping the IDM groups up to date is the responsibility of the group owners.

**Remote connection**

Connections are only allowed from the helsinki.fi domain. VPN or eduroam is not sufficient. To access ukko outside of the domain (e.g. home), add following to your ~/.ssh/config:

```
Host ukko2.cs.helsinki.fi
ProxyCommand ssh username@melkinpaasi.cs.helsinki.fi -W %h:%p
```

Note: with OpenSSH versions > 7.3, you can substitute the ProxyCommand with more human readable line: ProxyJump melkinpaasi.cs.helsinki.fi

To access a login node:

```
ssh <username>@ukko2.cs.helsinki.fi
```

Batch scheduling system is the most prominent difference between ukko and ukko2. Instead of logging directly into computing node and executing jobs there interactively, you now log in to a login node and submit the jobs via batch scheduler. The login node is for batch job management and for compiler/development environment only. Do not execute any production jobs there. Slurm handles the resource requests and optimises the resource allocation.
Another change is a **Module System**. Modules are used to manage software packages, compiler environments etc. Users can load or unload modules freely.

### 1.1 Ukko 1 Resources - Deferred Until Further Notice

- At this time there is a single Ukko 1 Cubbli Linux node available, see instructions. Node has 4 cores, 32GB of RAM.

### 1.2 Ukko 2 Resources

- Single **login** node **ukkan** serves logins, compiler environments and all batch scheduling functions
- 31 regular **compute** nodes (ukkan-02 - ukko-32): 28 cores, 2 threads and 256 GB RAM.
- 2 **big memory** nodes (ukkan-pekka, ukko2-paavo): 96 cores, 2 threads and 3 TB RAM.
- 2 GPU nodes (ukkan-g01, ukko2-g02): 28 cores, 2 threads, 512 GB RAM and 4 Tesla P100 GPU cards.

### 1.3 I/O, disks and filesystems

User will have access to $PROJ and $WRKDIR from other department computers. Request has been made and implementation is in the works.

- Lustre work directory is $WRKDIR
- Project and long term data storage for data and executables is at $PROJ
- Users own programs can be placed in $USERAPPL

Please do make appropriate changes to your programs to reflect the new work and project directories. To accomplish this easily, you can use environment variables $WRKDIR and $PROJ and then use the environment variables across the code. This enables much flexibility.

Ukko2 has very little local disk space. Please do not use local drives of the nodes. If you need local drives as a resource, please consider Kale instead.

### 2.0 Scientific software

Most development tools and software packages are available through modules, but not all. Scientific Software use cases give additional instructions and details. Comprehensive details about the available resources can be found from the Resources for Research Guide.

### 3.0 Queuing system

Jobs are managed by SLURM and runtime environment is managed with aforementioned modules. The jobs are submitted from the ukko2 login node (ukkan.cs.helsinki.fi). There are six production queues and one short queue to test jobs. Some queues overlap resources for better system utilisation. There is no limit for simultaneous user jobs that can stay waiting in queues at any one time.

<table>
<thead>
<tr>
<th>Queue name</th>
<th>Wall Time limit</th>
<th>Cores*</th>
<th>Memory per core**</th>
<th>Memory *</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>24h</td>
<td>1032</td>
<td>8GB - 32GB</td>
<td>257 GB</td>
<td>257 GB - to 3TB</td>
</tr>
<tr>
<td>long</td>
<td>14 days</td>
<td>868</td>
<td>8GB</td>
<td>257 GB</td>
<td></td>
</tr>
<tr>
<td>extralong</td>
<td>60 days</td>
<td>28</td>
<td>8GB</td>
<td>257 GB</td>
<td></td>
</tr>
<tr>
<td>bigmem</td>
<td>7 days</td>
<td>192</td>
<td>32GB</td>
<td>3TB</td>
<td>Two nodes.</td>
</tr>
</tbody>
</table>
| gpu        | 7 days          | 56     | 18GB              |          | Have to reserve with #SBATCH p gpu and --Gres:gpu=<nbr# gpu/s>

**Notes**

* Cores refer to the number of cores available.
** Memory per core refers to the memory available per core.
*Core count indicates the amount of cores available in the queue. Cores are distributed across multiple nodes. Some nodes have more cores, some have less.

**Memory per core is not a limit of memory that one core can reserve.

***Node memory is the maximum amount of memory single core job can reserve. In this case you are reserving the entire node.

### 3.1 Creating a Batch job

Simplest way to submit job into the system is to do it as a simple serial job. Following example is requesting 1 core and 10M of memory for 10 minutes and placement in a test queue. At the end of the script, srun command is used to start the program. You can think the scheduler as elaborate time and resource reservation system. #SBATCH -section describes the required resources and the rest is just a regular Linux. You can execute pretty much any Linux script or command you would be able to execute on the login node. There are few exceptions such as setting up daemons, but nothing you would ordinarily encounter.

Batch script needs to start with shebang (`#!/bin/bash`) and the batch parameters have to be set in the script before the actual program.

#### Time conventions

Sbatch syntax for time may not be obvious. Normally times are set as DD-hh:mm:ss where DD=days, hh=hours, mm=minutes, and ss=seconds. However, you can also use 2-0 to represent 2 days, while 10:00 would indicate 10 minutes.

Let's create an example batch script (we call this file in example: batch-submit.job) which is used to launch the user application in the system. Batch script has two sections, first the lines with #SBATCH indicating the scheduler directions, and then below it the actual payload:

```bash
#!/bin/bash
#SBATCH --job-name=test
#SBATCH -o result.txt
#SBATCH --workdir=<Workdir path>
#SBATCH -p test
#SBATCH -c 1
#SBATCH -t 10:00
#SBATCH --mem=10M
srun hostname
srun sleep 60
```

Following command submits the batch-script into the system and , and the scheduler takes care of the job placement (sbatch accepts additional options on the command line):

```bash
sbatch batch-submit.job
```

Batch job inherits the environment from the session you have submitted the job. This includes for example python virtualenv. Additionally, the job inherits the directory you are in as the default working directory, unless it is specified.

#### 3.1.1 Environment Variables and Exit Codes

It is possible to limit the environment variables for the batch job, or interactive session. If you need only some environment variables to be propagated from your session, or none, you can choose export option (default is ALL. See a special case for Cubbli Linux nodes):

```bash
--export=<environment variables | ALL | NONE>
```

When batch job is submitted and launched, Slurm sets number of environment variables which can be used for job control. Standard linux exit codes are used at job exit. Please see this page for a full compendium of the variables and error codes.

#### 3.1.2 Serial - Consumable resources

Below are some of the most common batch job options for serial jobs. If no values are given, system defaults are used. These values are used to determine the job priority.

**Job Wall Time limit:**

```bash
#SBATCH -t <Wall Time limit>
```
Job CPU count equals to the cores:

```
#SBATCH -c <CPU count>
```

Job memory limit (Please Note: memory reservation is per core):

```
#SBATCH --mem-per-cpu=<MB>
```

### 3.1.3 Further job control

When a job is submitted, you can use following commands to view the status of the queues, and change the job status if needed:

**Show queue information:**

```
sinfo -l
```

If you want to cancel your job:

```
scancel <jobID>
```

Useful way to see the system overall usage:

```
slurm s
```

For a full view:

```
slurm f
```

More traditional view to the system utilisation:

```
squeue
```

To check the status of your jobs that are in the queue:

```
squeue -u yourusername
```

For information about a job that is running:

```
scontrol show jobid -dd <jobID>
```

For information about a completed job’s efficiency. Output of seff is automatically included in the end of job mail notifications, if notifications are set to be sent in the batch script.

```
seff <jobID>
```

**Seff**

Note that seff produces meaningful output only from successfully completed job.

Job control summary of less common commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sacct</td>
<td>Displays accounting data for all jobs.</td>
</tr>
<tr>
<td>scontrol</td>
<td>View SLURM configuration and state.</td>
</tr>
<tr>
<td>sjstat</td>
<td>Display statistics of jobs under control of SLURM (combines data from sinfo, squeue and scontrol).</td>
</tr>
<tr>
<td>sprio</td>
<td>Display the priorities of the pending jobs. Jobs with higher priorities are launched first.</td>
</tr>
<tr>
<td>smap</td>
<td>Graphically view information about SLURM jobs, partitions, and set configurations parameters.</td>
</tr>
</tbody>
</table>

Comprehensive Slurm Quick Reference & Cheat Sheet which can be printed out.

Page for additional information if you prefer PBS-like job control.

### 3.2 Serial or Parallel Job?
Serial job is any program that runs on a single machine. In case of Ukko2, it means a program running on a single core.

Parallel job is composed of multiple processes which run on multiple machines. Simplest case would be a job that uses two cpu's and sets of related processes. Processes talk to each other through a medium shared between the cpu's, like a local memory space.

More about parallel processing, and related options, please see page about Parallel Processing. If you are interested in Spark deployment, please see a Spark User Guide.

### 3.2.1 Testing and Development

Before running jobs on the production queues, resource requirements, and in case of MPI jobs, scalability should be tested. 1h test queue is available for this purpose. "-p" parameter is mandatory for test jobs.

<table>
<thead>
<tr>
<th>Queues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-GPU Production jobs do not require -p option. Leaving it out allows greater flexibility for a job placement.</td>
</tr>
</tbody>
</table>

```
#SBATCH -p test

When submitting job for test queue, following parameters could be set in the batch .sh file. Mail parameters can be used with any jobs, they are not limited to test:

```
#!/bin/bash #SBATCH --job-name=test #SBATCH --workdir=<Workdir path> #SBATCH --output=foobar.out #SBATCH -p test #SBATCH -c 1 #SBATCH -e foobar.err #SBATCH --mail-type=END #SBATCH --mail-user=user@address.mail

srun hostname
srun sleep 60
```

### 3.2.2 How to request GPU's

Below an example script for GPU usage, assuming single GPU (note that for this you need to use --gres:gpu=1), two cores and 10M of memory to be used for a default time:

```
#!/bin/bash #SBATCH --job-name=test #SBATCH --workdir=<Workdir path> #SBATCH -o result.txt #SBATCH -p gpu #SBATCH -c 2 #SBATCH --gres=gpu:1 #SBATCH --mem=10M

srun hostname
srun sleep 60
```

### 3.2.3 Setting up e-mail notifications

You can set up e-mail notifications for batch job. If set, changes in the job status will be sent to specified user. Default is the user who submits the job.

```
E-mail Notification

Job END and ALL -notifications include the seff output. It is very useful for finding out the actual job resource utilisation.
```

### GPU Grant

You may not have heard of Nvidia GPU Grant Program. Professors and Researchers may be eligible to participate to the program. If in doubt, please do check the requirements and eligibility.
Most commonly chosen mail options are: NONE, BEGIN, END, FAIL or ALL. To set the option, following line is needed in the batch script. Multiple options can be set as comma separated list:

```
#SBATCH --mail-type=<option>,<option>
```

User may also specify mail address other than default:

```
#SBATCH --mail-user=<mail@address>
```

### 3.3 Interactive use

There is no need for direct access to node to start interactive session. Slurm allows interactive sessions to be started with `srun` and it is a great way to do testing and debugging. After entering the `srun` command, interactive job request is sent to the normal queue to wait for resources to become available. Once resources are available the session starts on a compute node, and you are put into the directory from which you ran the launched the session. To change the working directory within the `srun` session, before execution starts, you can use following:

```
--chdir=$WRKDIR
```

You can then run commands or programs like you would do in any ordinary Linux session. Your running environment on compute node is determined by:

1. The environment as set in your session from which you launch the `srun` command.
2. Any extra variables set by Slurm
3. Settings from your `.bashrc` file

Below an example of starting 1 core, 1 task interactive session with bash shell. If values are not set, they are inherited from the system or queue defaults.

```
srun -c 1 --ntasks-per-node=1 --pty bash
```

To show slurm variables when session starts:

```
export | grep SLURM
```

If you wish to use X forwarding, then you need to add following to your `srun` command. **THIS IS NOT AVAILABLE YET (11.10.2018).**

```
--x11
```

Meanwhile as a workaround you can use tool called `sinteractive` in place of `srun` like this (*note that `sinteractive` opens interactive shell and to use X11 forwarding, you should use `ssh -YA` flags when logging to the login node)*:

```
sinteractive -c 1 --ntasks-per-node=1 --mem=10M
```

**Interactive node availability**

At this time, there are no dedicated nodes for pure interactive use but we have acknowledged the need and will chance the policy accordingly. Estimated time for this feature is 2018Q4. This said, you can always submit interactive session requests to the system with `srun`. As long as there are free resources, and you request modest amount of resources, you will get session immediately.

### 3.4 Advance Reservations

Slurm supports Advance Reservations. You or your group may ask for a specific resources for a dedicated time slot. However, because Advance Reservations are not ordinary user option to choose, specific request needs to be submitted to helpdesk(at)helsinki(dot)fi to enable the reservation. Advance Reservations are disruptive to the system operation (*jobs need to be drained from the system to enable empty slot at the given time*) and the resource requirements have to be justified.

Once reservation is created, you can use your reservation by issuing:

```
sbatch --reservation=<reservation-name> <batch.script>
```

### 3.5 Actual Resource Utilisation

Slurm features simple utility to provide job utilisation details from any job that has completed. Using this utility helps to determine actual resource needs, of the job for future reference. You can run job once with much higher resource requests, and then use `self` to find out the actual use, which you can then use for later runs:
seff <completed job ID>

**Resource Requests**

Accurate resource requests *(memory, CPU and time)* will expedite the execution of your jobs - and lead to better system utilisation. Less you request, more expedient execution.

### 3.5.1 Job Accounting Data

Slurm has a powerful accounting feature with myriad options to choose from. Below a line featuring some of the more useful details:

```bash
sacct -oJobID,JobName,ExitCode,NNodes,NCPUS,MaxRSS,Elapsed,End
```

Provides easy to read list formatted output, where fields are:

- **JobID**: Job identification number
- **JobName**: Job name given in the Slurm batch script
- **ExitCode**: Exit code once job was terminated
- **NNodes**: Node count
- **NCPUS**: CPU's (Core) reserved by the job
- **MaxRSS**: Memory peak usage during job execution, returns value when job has finished. This value can be used to adjust the requested memory value in the batch script accordingly.
- **Elapsed**: Time batch job was in execution
- **End**: End time of batch job

Other examples:

- Lists details when JobID is known:
  ```bash
  sacct -j <jobID> -oJobID,JobName,ExitCode,NNodes,NCPUS,MaxRSS,Elapsed,End
  ```
- Jobs listed by UserID:
  ```bash
  sacct -u <userID> -oJobID,JobName,ExitCode,NNodes,NCPUS,MaxRSS,Elapsed,End
  ```

**sacct man page**

Comprehensive accounting options, and parameters can be found from the accounting man page.

### 3.5.2 Scheduling Policy

Job execution priorities depend upon user resource requests. If no resource limits are requested in the batch script, then system and queue defaults are used. Job priority and scheduling decisions are based on available system resources. Fair Share is applied to allocate everyone near equal share of the system. Below a list of resources considered with most "expensive" resource on the top:

1. GPU requested
2. Memory requested
3. Wall Time requested
4. CPUs requested

**System Defaults**

System defaults: Job placement in short queue, allocation of 512MB of memory per CPU (core), 1 hour of Wall Time and 1 CPU (core). Defaults can be changed by user specified resource requests in the job batch script.
3.0 Further Reading

- Aalto University Triton User Guide
- Technical Specifications of Ukko2
- Parallel Processing
- GDB Debugger Cheat Sheet
- PBS Command Wrappers
- Module System
- CSC's Taito cluster's documentation may be useful
- CSC SLURM instructions
- Slurm Quick Reference & Cheat Sheet
- Ukko Cubbli Linux Instructions
- Spark User Guide
- GCC Optimization Guide
- MVAPICH in Depth