Table of Contents

1. THE PC² COMPUTE CENTER ........................................................................ 3

2. THE OULUS CLUSTER ............................................................................. 5
   2.1 FRONTENDS ......................................................................................... 6
      2.1.1 Login via ssh .............................................................................. 6
      2.1.2 Login from a Windows Based System ........................................... 6
      2.1.3 Tutorial for Linux Beginners ......................................................... 6
      2.1.4 Customize the OpenCCS Command Line Interface ....................... 6
   2.2 FILE-SYSTEMS .................................................................................... 7
      2.2.1 HOME: Home Directory ............................................................... 7
      2.2.2 PC2DATA: PC² Wide Group Data Directories ................................. 7
      2.2.3 PC2PFS: Cluster Local, BeeGFS Based Parallel File System .......... 8
      2.2.4 PC2SCRATCH: PC² Wide Group Scratch Directories ..................... 8
      2.2.5 PC2SW: Software Directory ......................................................... 8
      2.2.6 TMPDIR, CCS_TMPDIR: Node Local Directories for Jobs ............... 8
      2.2.7 FAQs .......................................................................................... 8

3. CHOOSING SOFTWARE PACKAGES ....................................................... 9
   3.1 USAGE HINTS .................................................................................... 9

4. WORKLOAD-MANAGER (OPENCCS) .................................................. 10
   4.1 WHAT IS A WORKLOAD MANAGER? .................................................. 10
   4.2 BASIC JOB FLOW ............................................................................ 10
   4.3 OVERVIEW OF CCS COMMANDS ................................................. 11
   4.4 DIFFERENCES TO QUEUEING BASED SYSTEMS ................................ 11
   4.5 RESOURCE REQUESTS .................................................................... 11
      4.5.1 Resource Sets (Chunks) ............................................................... 12
      4.5.2 Placement Directives ................................................................. 12
      4.5.3 The Boot-Node ......................................................................... 13
      4.5.4 Conversion from TORQUE ......................................................... 14
      4.5.5 MPI ........................................................................................... 15
      4.5.6 OpenMP .................................................................................. 16
      4.5.7 HYBRID .................................................................................. 17
      4.5.8 Grouping ................................................................................. 18
   4.6 WORKER CONCEPT ......................................................................... 18
   4.7 BATCH JOB SCRIPTS ....................................................................... 19
   4.8 JOB ARRAYS .................................................................................. 19
   4.9 PREDICTING JOB START TIMES .................................................... 20
      4.9.1 Examples ................................................................................ 20

5. JOB SUBMISSION EXAMPLES ............................................................. 21
   5.1 STANDARD APPLICATIONS ............................................................ 21
   5.2 MPI ................................................................................................. 21
   5.3 TESLA GRAPHIC CARD ................................................................. 21
   5.4 GTX 1080 GRAPHIC CARD ............................................................. 22
   5.5 (UN)SELECTING SPECIFIC NODE TYPES ...................................... 22
   5.6 LARGE COMPUTE NODES AND SMP NODES .................................. 22

6. ONLINE DOCUMENTATION AND CONTACT .................................... 23
1. The PC² Compute Center

The location provides 160m² floor space for offices and additional space for laboratories, training, and meeting rooms. The compute center consists of five computer rooms, with 195m² floor space for HPC systems operated by the PC² and 160m² floor space for systems operated by the IMT (Zentrum für Informations- und Medientechnologien). The building infrastructure provides a redundant cooling system (pipelines to the central cooling facility of the campus and chillers on the roof) and a fault tolerant electrical system with battery backup and emergency power generator. Together with state-of-the-art security and fire protection equipment, we are able to provide reliable services and HPC systems with up to 550 kWatt of electricity consumption.
OCULUS BRIEF INSTRUCTIONS

One of two chillers on the roof

Cooling and Heating central in the cellar

Nitrogen fire suppression

Tichelmann ring and power supply of OCuLUS
2. The OCuLUS Cluster

With the financial support of the state North Rhine-Westphalia and the federal republic of Germany, the PC² established 2013 the OCuLUS cluster. The official opening was April 16th 2013.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak performance</td>
<td>200 TFLOP/s (CPU) and &gt; 40 TFLOP/s (GPU)</td>
</tr>
<tr>
<td>Processor cores</td>
<td>9,920 Intel Xeon “Sandy Bridge”</td>
</tr>
<tr>
<td>Total main memory</td>
<td>45 TByte</td>
</tr>
<tr>
<td>Cabinets</td>
<td>12 racks - backdoor cooling, 2 racks - air cooling</td>
</tr>
<tr>
<td>Compute nodes (small)</td>
<td>552, each with two Intel Xeon E5-2670 (16 cores), 2.6 GHz, 64 GByte main memory</td>
</tr>
<tr>
<td>Compute nodes (large)</td>
<td>20, each with two Intel Xeon E5-2670 (16 cores), 2.6 GHz, 256 GByte main memory</td>
</tr>
<tr>
<td>GPU nodes Typ 1 (Tesla)</td>
<td>32, each with two Intel Xeon E5-2670 (16 cores), 2.6 GHz, 64 GByte main memory, 1 nVIDIA K20 (2.496 cores, 5 GByte)</td>
</tr>
<tr>
<td>GPU nodes Typ 2 (GTX 1080 Ti)</td>
<td>8, each with two Intel Xeon E5-2670 (16 cores), 2.6 GHz, 64 GByte main memory, 2 nVIDIA GeForce GTX1080 Ti (11 GByte)</td>
</tr>
<tr>
<td>SMP nodes</td>
<td>4, each with four Intel Xeon E5-4670 (32 cores), 2.7 GHz, 1 TByte main memory</td>
</tr>
<tr>
<td>Service nodes</td>
<td>6, frontend and admin nodes</td>
</tr>
<tr>
<td>Interconnect</td>
<td>QDR InfiniBand PCIe3, 40 Gbit/s Mellanox</td>
</tr>
<tr>
<td>Storage system</td>
<td>500 TByte parallel file system, FhGFS, 200 TByte Campus Storage, NFS4, CIFS</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux (CentOS 6.5)</td>
</tr>
</tbody>
</table>

Table 1: Specification of the OCuLUS cluster
2.1 Frontends

The OCuLUS has two frontends:
1. fe.pc2.uni-paderborn.de
2. fe-2.cv2012.pc2.uni-paderborn.de

The access is via ssh.
If you are a member of the University Paderborn, use your IMT account and password, else use the credentials we have sent to you.

2.1.1 Login via ssh

```
ssh account@fe.pc2.uni-paderborn.de
```

2.1.2 Login from a Windows Based System

If you are login from a system running Windows, you should ask your local administrator to install one of the following two Open-Source tools:
   - SmarTTY is a free multi-tabbed SSH client that supports copying files and directories with SCP on-the-fly and editing files in-place.

2. Putty and XMing
   - Putty

2.1.3 Tutorial for Linux Beginners

Try the following if you are not familiar with the Linux Command Line Interface (CLI) [http://faculty.smu.edu/reynolds/unixtut/](http://faculty.smu.edu/reynolds/unixtut/)

2.1.4 Customize the OpenCCS Command Line Interface

At the first login you should call

```
ccsgenrcfiles
```

It creates the directory .ccsrc in the user’s home directory and it creates the default configuration file (.ccsr/uirc) for the OpenCCS command line interface. In this file one can customize default values like email address(es), format of ccsinfo output, job-name templates, etc.
2.2 File-Systems

The cluster provides three types of file systems:

1. A shared file system located on an external filer. This file system is available on all clusters and on all systems running in the University network.
2. A cluster local parallel file system (BeeGFS). It provides fast access.

We provide environment variables which point to the different directories. They are set automatically at login on a frontend. The following table gives an overview about the cluster file systems. They are explained in more detail below.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Source</th>
<th>Purpose</th>
<th>Initial Quota</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>Filer</td>
<td>Permanent small data. Per account</td>
<td>5GB</td>
<td>Yes</td>
</tr>
<tr>
<td>PC2DATA</td>
<td>Filer</td>
<td>Permanent group data. Per group</td>
<td>Requested at project application</td>
<td>Yes</td>
</tr>
<tr>
<td>PC2PFS</td>
<td>BeeGFS</td>
<td>Cluster local temporary working data. Per group</td>
<td>Requested at project application</td>
<td>No</td>
</tr>
<tr>
<td>PC2SCRATCH</td>
<td>Filer</td>
<td>Temporary working data. Per group</td>
<td>Requested at project application</td>
<td>No</td>
</tr>
<tr>
<td>PC2SW</td>
<td>Filer / Cluster</td>
<td>Pre-installed software Read only</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>TMPDIR, CCS_TMPDIR</td>
<td>Node local</td>
<td>Temporary, created by the WLM</td>
<td>None</td>
<td>No</td>
</tr>
</tbody>
</table>

2.2.1 HOME: Home Directory

This directory is accessible on all PC² systems and on all systems running in the University of Paderborn. It is restrictively quoted regarding file space and number of inodes. I/O-intensive applications should not use HOME. For this purpose, we provide PC2PFS and PC2SCRATCH. HOME is backed up in regular intervals.

2.2.2 PC2DATA: PC² Wide Group Data Directories

Upon request, we provide a directory to store persistent data. PC2DATA is accessible on all PC² systems and on all systems running in the University of Paderborn. It is restrictively quoted regarding file space and number of inodes. PC2DATA is backed up in regular intervals. It is mounted read-only on the compute
nodes and read-write on the frontends.

2.2.3 PC2PFS: Cluster Local, BeeGFS Based Parallel File System
Upon request, we provide a directory on the fast, BeeGFS based parallel file system on OCuLUS to store temporary data created by jobs running on OCuLUS. PC2PFS is accessible only on OCuLUS. There is no backup service.

2.2.4 PC2SCRATCH: PC² Wide Group Scratch Directories
Upon request, we provide a directory to store temporary data created by jobs running on the clusters. PC2SCRATCH is accessible on all PC² systems and on all systems running in the University of Paderborn. It is restrictively quoted regarding file space and number of inodes. There is no backup service.

2.2.5 PC2SW: Software Directory
This directory contains all pre-installed software.

2.2.6 TMPDIR, CCS_TMPDIR: Node Local Directories for Jobs
When a job is allocated via the Work Load Manager (WLM), a temporary directory for the job is created on each affected node. It will be deleted by the WLM, after the job has terminated.

2.2.7 FAQs
For more information about the provided file systems, refer to our Wiki.

https://wikis.uni-paderborn.de/pc2doc/index.php/PC2-FileSystems - FAQs
3. Choosing Software Packages

The PC² provides a wide range of development software (compilers, libraries, debuggers, profilers, etc.) as well as specific scientific applications. Many of the available programs require certain environment variables to be set or changed, e.g. \texttt{PATH}, \texttt{LD_LIBRARY_PATH}, \texttt{MANPATH}. The environment modules package is employed to access or switch between various, sometimes conflicting versions of software. It provides the means to change the environment dynamically by loading, switching or unloading specific software modules. The module installation is customized to automatically resolve dependencies between modules.

The following table shows the most used commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>module avail</td>
<td>list the available software packages</td>
</tr>
<tr>
<td>module list</td>
<td>show the loaded modules</td>
</tr>
<tr>
<td>module add</td>
<td>load a module</td>
</tr>
<tr>
<td>module del</td>
<td>unload a module</td>
</tr>
<tr>
<td>module display</td>
<td>show what the module does</td>
</tr>
</tbody>
</table>

A module can either be identified by its name only or its name together with a specific version, for example \texttt{intel} or \texttt{intel/11.1}, respectively. When only the name is specified, the default version, as shown by \texttt{module avail}, is loaded.

3.1 Usage Hints

1. Loading a module changes the environment of the shell in which it is loaded.

2. If other already loaded modules conflict with the module to load, an according error message is displayed and no further changes are made.

3. If other modules are required by the module to load, these are loaded recursively and an according message is displayed.

4. Removing a module reverts the changes in the environment of the shell.

5. If other modules depend on the module to remove, these are removed recursively and an according message is displayed.

For more information about \textit{Modules}, please read the man page or refer to the Modules Project home page: \url{http://modules.sourceforge.net/}
4. Workload-Manager (OpenCCS)

On OCuLUS, we use OpenCCS as Workload Management System (WLM) to control the usage of the compute nodes. Interaction with the system is done by using CCS commands, e.g., ccsalloc, ccsinfo, or ccskill. You will find the OpenCCS User-Manual in $PC2SW/doc.

4.1 What is a Workload Manager?

A compute cluster is a collection of computers and other resources (networks, storage systems, license servers, GPUs, and so forth). Using a WLM typically reduces technical administration of resources while offering a uniform view to users. A WLM abstracts away many of the details involved with running and managing jobs, allowing higher resource utilization. For example, users typically only need to specify the constraints of a job and do not need to know the individual machine names of each host on which they are running. Using a WLM, clusters can execute jobs simultaneously. Clusters often comprise four different components:

1. **Master Node**: A dedicated node hosting the WLM server components.
2. **Frontend Nodes**: They provide an entry point to the system for users to manage their workload. Users are able to compile, submit and track their jobs. Additionally, other job-specific pre- and post-processing is possible.
3. **Compute Nodes**: They execute submitted jobs, controlled by the WLM. A cluster may comprise hundreds or thousands of compute nodes.
4. **Resources**: Resources can include high-speed networks, storage systems, GPUs, compute nodes, software licenses, and so forth. A list of allocatable resources can be printed by ccsinfo –a or ccsinfo –a --classes

4.2 Basic Job Flow

The life cycle of a job can be divided into four stages: creation, submission, execution, and finalization.

1. **Creation**: Typically, a submit script is written to hold all of the parameters of a job. These parameters could include how long a job should run (walltime), what resources are necessary to run, and what to execute. The following is an example submit file:

   ```sh
   #!/usr/bin/sh
   #CCS --N localBlast
   #CCS --res=rset=1:ncpus=2:mem=10g
   #CCS -t 10d
   #CCS --M user@my.organization.com
   #CCS -mea
   cd $PC2SCRATCH
   myBlast --i -v
   ```

   This submit script specifies the name of the job (localBlast), that it needs two cores and 10 GiByte on a single node (rset=1:ncpus=2:mem=10g), that it will run for at
most 10 days, and that the WLM should email "user@my.organization.com" when the job exits or aborts. Additionally, the user specifies where and what to execute.

(2) **Submission**: A job is submitted with the `ccsalloc` command. Once submitted, the policies set by the administration of the site dictate the priority of the job and therefore, when it will start executing.

(3) **Execution**: Jobs often spend most of their lifecycle executing. While a job is running, its status can be queried with `ccsinfo`.

(4) **Finalization**: When a job completed, information about the job is available via `ccsinfo` for about 30 minutes.

### 4.3 Overview of CCS commands

The OpenCCS user interface provides seven commands:

1. `ccsalloc` Submit a job, a job-array, or reserve resources
2. `ccsalter` Alter jobs or reservations
3. `ccsbind` Reconnect to an interactive job
4. `ccsinfo` About nodes, schedule, users, jobs, limits, resources, …
5. `ccskill` Kill jobs and reservations
6. `ccsmmsg` Send messages to the job’s output files
7. `ccssignal` Send signals to jobs

### 4.4 Differences to Queueing Based Systems

CCS has no explicit queues. Things like limits, privileges, default values, etc. are assigned to projects and / or single users. The assignment of resources is based on the resource requests and the related limits. Limits may be time dependent.

`ccsinfo -l` shows your limits.

### 4.5 Resource Requests

Running a job requires to specify the resources the job wants to use.

CCS uses a resource specification syntax which is similar to that of PBSPro (but not Torque). The syntax is:

```
--res=<resource[=value][,resource[=value],...]>  
```

resource may be

- a name of an allocatable resource,
- a resource set specification,
- a placement specification.
4.5.1 Resource Sets (Chunks)
A resource set (also named chunk) specifies a set of resources that has to be allocated as a unit on a single node. Chunks cannot be split across nodes. Resource sets are specified using the keyword "rset".

Syntax: rset=[N:]chunk+[N:]chunk...

If N is not specified, it is set to 1.

A chunk comprises one or more resource=value statements separated by a colon. Examples:

- ncpus=2:mem=10g:hostname=Host1
- ncpus=27:vmem=20g:arch=linux+4:npcus=15:mem=128g

ccsinfo –a shows a summary of the available resources.
ccsinfo –a --classes shows the resource classes.

4.5.2 Placement Directives
One may specify how the chunks should be placed on the nodes.

Syntax: place=[arrangement] [:sharing ] [:grouping ] [:ignore]

- arrangement is one of free, pack, or scatter.
- sharing is one of excl or shared.
- grouping can have only one instance of group=resource.
- Ignore is a ';' separated list of hostnames, which should be excluded from mapping (ignore='H1;H2,...').

Default is free:shared. All keywords are described in the following table.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>No restriction</td>
</tr>
<tr>
<td>Pack</td>
<td>All chunks must be placed on one node only</td>
</tr>
<tr>
<td>Scatter</td>
<td>Only one chunk per node</td>
</tr>
<tr>
<td>Exclusive</td>
<td>Only this job may use the node</td>
</tr>
<tr>
<td>Shared</td>
<td>This chunk may share the node with other chunks</td>
</tr>
<tr>
<td>Grouping</td>
<td>Group all chunks by the specified resource</td>
</tr>
<tr>
<td>Ignore</td>
<td>Ignore these hosts while mapping</td>
</tr>
</tbody>
</table>
4.5.2.1 Grouping on a Resource
One can specify that all of the chunks of a job should run on nodes that have the same value for a selected resource. To group the chunks this way, use the following format: place=group=<resource> Chunk grouping will be ignored if:

- the OpenCCS administration disabled grouping,
- it is a single host job, i.e., all chunks are mapped to a single host,
- the job is part of a reservation,
- <resource> is
  - a job wide resource,
  - one of cput, hostname, mpiprocs, ompthreads, or walltime.

For example, lets assume there is a resource named ibswitch which reflects to which Infiniband switches a node is connected. The value can be "s10,s1" at one node, and "s11,s1" at another node. Then these nodes can be grouped by place=group=ibswitch because they share the string "s1".

All classes printed by ccsinfo -c --classes can be used for grouping (except the ones listed above). Using the method of grouping on a resource, one cannot specify what the value of the resource should be, only that all nodes have the same value. If one needs the resource to have a specific value, specify that value in the description of the chunks. Depending on the settings of the OpenCCS administration a job will be rejected at submit time, if no group is found which is large enough.

4.5.3 The Boot-Node
At allocating a partition one of the nodes becomes the boot node. This node controls the jobs started by the user (e.g. starts the job, holds the connection to the UI). The boot node is the first node in the environment variable CCS NODES and in all mapping infos. OpenCCS assigns the boot node to one of the nodes which satisfy the last specified chunk.

Example: --res=rset=ncpus=12:gtx1080=2+4:ncpus=8:mem=32g.

The boot node will be one of (4:ncpus=8:mem=32g)
4.5.4 Conversion from TORQUE

4.5.4.1 Rules

Torque: 
- `lnodes=X`  
CCS:  
- `--res=rset=X:ncpus=1,place=scatter`

Torque:  
- `ncpus=X` or `cpp=X`  
CCS:  
- `ncpus=X`

Torque:  
- `ppn=P`  
CCS:  
- `mpiprocs=P`

Torque:  
- `lnodes=N:ppn=P`  
CCS:  
- `--res=rset=N:ncpus=P:mpiprocs=P,place=scatter`

4.5.4.2 Examples

1. Request 5 cores and 10GiByte memory on a single node
   Torque:  
   - `-lncpus=5,mem=10g`  
   CCS:  
   - `--res=rset=1:ncpus=5:mem=10g`

2. Submit the same job as a job-array
   Torque:  
   - `-t 1-100 -lncpus=5,mem=10g`  
   CCS:  
   - `-J 1-100 --res=rset=1:ncpus=5:mem=10g`
   NOTE: Refer to the OpenCCS User-Manual for more information on job-arrays.

3. Request one node with a certain property
   Torque:  
   - `lnodes=1:property`  
   CCS:  
   - `--res=rset=1:ncpus=1:property=true`

4. Request 2 cores on each of four nodes with a given property
   Torque:  
   - `lnodes=4:property:ncpus=2`  
   CCS:  
   - `--res=rset=4:ncpus=2:property=true,place=scatter`

5. Request 1 core on each of 14 nodes asking for certain software, licenses and a job memory limit of 280MiByte
   Torque:  
   - `lnodes=14:mpi-fluent:ncpus=1`  
     `--lfluent=1`  
     `--lmem=280mb`  
   CCS:  
   - `--res=rset=14:ncpus=1:mem=20m`  
     -- `--res=place=scatter,fluent=1`

6. Request 1 node of type "fserver" and 15 nodes which are not of type "fserver"
   Torque:  
   - `lnodes=1:fserver+15:noserver`  
   CCS:  
   - `--res=rset=1:ncpus=1:fserver=True`  
     -- `--res=rset=15:ncpus=1:fserver=false`  
     -- `--res=place=scatter`
7. Allocate 4 nodes, each with 6 cores with 3 MPI processes per node, with each node on a separate node.
   Torque: `-lnodes=4:ppn=3:ncpus=2`
   CCS: `--res=rset=4:ncpus=6:mpiprocs=3,place=scatter`

8. Allocate 4 separate nodes, with the property blue. The amount of memory allocated from each node is 2560MB ( = 10GB / 4) rather than 10GB from each node.
   Torque: `-lnodes=4:blue:ncpus=2`
   `-l mem=10GB`
   CCS: `--res=rset=4:blue=true:ncpus=2:mem=2560M`
   `--res=place=scatter`

4.5.5 **MPI**

How you request chunks matters.
The number of MPI processes per chunk defaults to 1, unless you specify this value using the `mpiprocs` resource.

4.5.5.1 **Examples**

1. Request two MPI processes for each of four chunks, where each chunk has two cores:
   `--res=rset=4:ncpus=2:mpiprocs=2`

2. Request one chunk with two MPI processes and one chunk with one MPI process, where both chunks have two cores:
   `--res=rset=ncpus=2:mpiprocs=2+ncpus=2`

3. Request three chunks, each with one MPI process:
   `--res=rset=3:ncpus=2`
   This results in the following node file:
   Node A
   Node B
   Node C
   NOTE: CCS may run all chunks on the same node unless you use `place=scatter`.

4. If you want to run two MPI processes on each of three hosts and have the MPI processes share a single core on each host, request:
   `--res=rset=3:ncpus=1:mpiprocs=2`
   The node file then contains the following list:
   NodeA
   NodeA
   NodeB
   NodeB
   NodeC
   NodeC
NOTE: CCS may run all chunks on the same node unless you use place=scatter.

5. If you want three chunks, each with two cores and running two MPI processes, use:
   --res=rset=3:ncpus=2:mpiprocs=2...
   The node file then contains the following list:
   NodeA
   NodeA
   NodeB
   NodeB
   NodeC
   NodeC

NOTES:
- The node file is the same as the previous example, even though the number of cores used is different.
- CCS may run all chunks on the same node unless you use place=scatter.

6. To request two different nodes, each with eight cores and eight MPI ranks, for a total of 16 cores and 16 ranks:
   --res=rset=2:ncpus=8:mpiprocs=8,place=scatter

7. To request 504 cores each with 10GiByte mem and 20GiByte vmem and 8 cores with 20GiByte mem for the boot-node:
   --res=rset=512:ncpus=1:mem=10g:vmem=20g+ncpus=8:mem=20g

4.5.6 OpenMP
OpenCCS supports OpenMP applications by setting the OMP_NUM_THREADS variable in the job’s environment, based on the resource request of the job. The OpenMP run-time picks up the value of OMP_NUM_THREADS and creates threads appropriately.
- OpenCCS sets the value of OMP_NUM_THREADS based on the last chunk of the rset statement.
- If you request ompthreads in the last chunk, OpenCCS sets the environment variable to the value of ompthreads.
- If you do not request ompthreads in the last chunk, then OMP_NUM_THREADS is set to the value of the ncpus resource of that chunk.
- If you do not request either ncpus or ompthreads for the last chunk of the select statement, then OMP_NUM_THREADS is set to 1.

4.5.6.1 Examples
1. Submit an OpenMP job as a single chunk, for a two-core, two-thread job requiring 10GiByte of memory:
   --res=rset=1:ncpus=2:mem=10g
2. Run an MPI application with 64 MPI processes, and one thread per process:
   --res=rset=64:ncpus=1

3. Run an MPI application with 64 MPI processes, and four OpenMP threads per process:
   --res=rset=64:ncpus=4
   or
   --res=rset=64:ncpus=4:ompthreads=4

4.5.6.2 Fewer Threads than Cores
You might be running an OpenMP application on a host and wish to run fewer threads than the number of cores requested. This might be because the threads need exclusive access to shared resources in a multi-core processor system, such as to a cache shared between cores, or to the memory shared between cores.
You want one chunk, with 16 cores and eight threads:
   --res=rset=1:ncpus=16:ompthreads=8

4.5.6.3 More Threads than Cores
You might be running an OpenMP application on a host and wish to run more threads than the number of cores requested, perhaps because each thread is I/O bound.
You want one chunk, with eight cores and 16 threads:
   --res=rset=1:ncpus=8:ompthreads=16

   NOTE: Your job may be killed by OpenCCS due to the ncpus limit if your job uses more than the requested number of ncpus. Refer to the OpenCCS User Manual chapter „Limit Enforcement“ for more information.

4.5.7 HYBRID
For jobs that are both MPI and multi-threaded, the number of threads per chunk, for all chunks, is set to the number of threads requested (explicitly or implicitly) in the last chunk.
For most MPIs, the OMP_NUM_THREADS and NCPUS environment variables default to the number of ncpus requested for the last chunk.
Should you have a job that is both MPI and multi-threaded, you can request one chunk for each MPI process, or set mpiprocs to the number of MPI processes you want on each chunk.

4.5.7.1 Examples
1. To request four chunks, each with one MPI process, two cores and two threads:
   --res=rset=4:ncpus=2
   or
   --res=rset=4:ncpus=2:ompthreads=2

2. To request four chunks, each with two cores and four threads:
   --res=rset=4:ncpus=2:ompthreads=4
3. To request 16 MPI processes each with two threads on two cores:
   --res=rset=16:ncpus=2

4. To request two chunks, each with eight cores and eight MPI tasks and four threads:
   --res=rset=2:ncpus=8:mpiprocs=8:ompthreads=4

5. --res=rset=4:ncpus=2
   If this request is satisfied by four cores from NodeA, two from NodeB and two from NodeC, the following is written to $CCS_NODEFILE:
   NodeA
   NodeA
   NodeB
   NodeC

   On the boot node (nodeA) OpenCCS sets: OMP_NUM_THREADS=2 and NCPUS=2

6. For the following:
   --res=rset=3:ncpus=2:mpiprocs=2:ompthreads=1
   If this is satisfied by two cores from each of three nodes (NodeA, NodeB, NodeC), so the following is written to $CCS_NODEFILE:
   NodeA
   NodeA
   NodeB
   NodeB
   NodeC
   NodeC

   On the boot node (nodeA) OpenCCS sets: OMP_NUM_THREADS=1 and NCPUS=1

4.5.8 Grouping

1. To group the chunks on a single rack use place=free:group=rack
2. To group by Infiniband switches use place=group=ibswitch

4.6 Worker Concept
Cluster systems comprise nodes with full operating system capabilities and software packages like debuggers, performance analyzers, numerical libraries, and runtime environments. Often these software packages require specific pre- and post-processing. OpenCCS supports this with the so-called worker concept. Workers are tools to start jobs under specific runtime environments. They hide specific procedures (e.g. starting of a daemon or setting of environment variables) and provide a convenient way to start and control programs.

csinfo --worker shows a list of the available workers.

csinfo --whelp=WORKER gives a worker specific help.

For example, we provide workers for different MPI releases (e.g., Intel-MPI,
OpenMPI, MVAPICH). All of them load the default module of the related MPI and start mpirun with appropriate arguments. They also ensure that the needed environment is available on all affected nodes. Additional mpirun arguments may be added. They are forwarded to mpirun.

4.7 Batch job scripts

Job scripts are executable (shell) scripts, which contain resource requirements as well as the actual commands to be executed.

Example job script

```bash
#!/usr/bin/sh
#CCS --res=rset=5:ncpus=4:mem=10g
#CCS --res=matlab=5
##CCS --res=place=scatter:excl
#CCS -t 1h
my_preproc
ccsworker g09 -- my_gaussianjob
```

Arguments given to ccsalloc override script directives.

You will find example scripts in $PC2SW/examples. Copy them to your working directory and adapt them to your requirements.

4.8 Job Arrays

Job arrays may be used to group closely related work into a set so that you can submit, query, modify, and display the set as a unit. Job arrays are useful whenever you want to run the same program over and over on different input files.

```bash
#!/usr/bin/sh
#CCS --res=rset=5:ncpus=4:mem=10g
#CCS --res=matlab=5
##CCS --res=place=scatter:excl
#CCS -t 1h
#CCS -J"1-100:2,200-500:50"
my_preproc
ccsworker g09 -- my_gaussianjob.${CCS_ARRAY_INDEX}
```

You will find an example script in $PC2SW/examples/arrayJob.sh. Copy it to your working directory and adapt it to your requirements.

For detailed information on how to submit and manage job arrays, refer to the OpenCCS User-Manual.
4.9 Predicting Job Start Times

It is sometimes useful to know which resources are when available. For example, how many GPUs can I get now or how long is the waiting time if requesting chunks with 5 cores and 6GB per core.

For this purposes OpenCCS provides ccsinfo --predict. It allows to specify resource requests together with iterators and OpenCCS will print a list with the earliest start times. The syntax is:
ccsinfo [-g GROUP] [--raw] -p "<resources>[;<iterator>;...]"

OpenCCS will replace all found iterators in resources by their specified values and plan this request(s) including all user / group related limitations. It returns the planned start times. Please note that the situation may change within seconds, if other users are submitting jobs in the meanwhile.

For detailed information about the syntax, refer to the OpenCCS User-Manual chapter 7 or the ccsinfo man page "PREDICTING START TIMES".

4.9.1 Examples

- Predict the start times of 10 up to 100 nodes (stepping 10) with a duration of 1 hour.
  o ccsinfo -p "-n %C; %C=10-100:10; %D=1h"
- Predict the start times of 32-256 chunks with 5 cores and 30g with a duration 1 or 2 hours.
  o ccsinfo -p "%C:ncpus=5:mem=30g+ncpus=1:mem=60g; %C=32-256:16; %D=1-2h"
- Predict the start times of 15-20 chunks with Tesla GPUs, 5 cores and 30GiByte RAM, duration 1-2 hours.
  o ccsinfo -p "%C:tesla=1:ncpus=5:mem=30g, place=scatter; %C=15-20; %D=1-2h"
- Predict 1-4 chunks with 1-5 cores, and 4GB and 8GiByte per core, duration 1 day.
  o ccsinfo -p "%C:ncpus=%1:mem=%2G:vmem=%3g; %1=1-5; %2=4-64:4; %3=8-128:8; %C=1-4; %D=1d"

The output of the last example, may look like:

<table>
<thead>
<tr>
<th>Duration Starts at (in)</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1d now</td>
<td>1:ncpus=1:mem=4G:vmem=8g</td>
</tr>
<tr>
<td>1d 23:20:19.08</td>
<td>(4d10h5m) 1:ncpus=2:mem=8G:vmem=16g</td>
</tr>
<tr>
<td>1d 23:30:19.08</td>
<td>(4d10h15m) 1:ncpus=3:mem=12G:vmem=24g</td>
</tr>
<tr>
<td>1d 23:30:19.08</td>
<td>(4d10h15m) 1:ncpus=4:mem=16G:vmem=32g</td>
</tr>
<tr>
<td>1d 20:56:20.08</td>
<td>(5d7h41m) 1:ncpus=5:mem=20G:vmem=40g</td>
</tr>
<tr>
<td>1d now</td>
<td>2:ncpus=1:mem=4G:vmem=8g</td>
</tr>
<tr>
<td>1d 23:30:19.08</td>
<td>(4d10h15m) 2:ncpus=2:mem=8G:vmem=16g</td>
</tr>
<tr>
<td>1d 19:20:22.08</td>
<td>(7d6h5m) 2:ncpus=3:mem=12G:vmem=24g</td>
</tr>
<tr>
<td>1d 19:20:22.08</td>
<td>(7d6h5m) 2:ncpus=4:mem=16G:vmem=32g</td>
</tr>
<tr>
<td>1d 19:40:26.08</td>
<td>(11d6h25m) 2:ncpus=5:mem=20G:vmem=40g</td>
</tr>
</tbody>
</table>
5. Job Submission Examples
The standard way of submitting jobs to OpenCCS is using the command ccsalloc. For a submission, OpenCCS needs this information:

- The resources to allocate (refer to chapter 4.3).
  - The job will be killed by CCS if it tries to use more resources than requested.
- How long the job will run.
  - The job will be killed by CCS if the runtime is exceeded.
  - Using ccsalter one can request to increase the runtime.
- The corresponding executable.
  - This may be a job script or an executable. Arguments may be added.
    E.g.: ccsalloc job.sh --x 1 --y 3

If you are familiar with PBSPro or Torque, you will find several similar directives. Refer to the ccsalloc man page or the OpenCCS User-Manual.

5.1 Standard Applications
Refer to $PC2SW/examples. There are a lot of example script.
E.g., Abaqus, Ansys, Comsol, Gaussian, Matlab, Orca, StarCCM,Turbomole, ...

5.2 MPI
Refer also to sections 4.3.5 until 4.3.6
- Run an OpenMPI application on 4 chunks each with 4 cores and 32 GiByte of memory:
  ccsalloc --res=rset=4:ncpus=10:mem=32g ompi -- my_OMPI_app

5.3 Tesla Graphic Card
For Tesla K20, we provide the consumable resource tesla. This avoids that more than one job will be scheduled to a card at the same time. Hence, to request 2 chunks each with 8 cpus and one Tesla card use:
  --res=rset=2:ncpus=8:tesla=1

For offload jobs CCS sets the environment variable: CUDA_VISIBLE_DEVICES=0

For jobs mapped on a GPU node but not requesting the Tesla accelerator CCS sets the environment variable: CUDA_VISIBLE_DEVICES=1024 which is an invalid value. Refer to $PC2SW/examples for example scripts.
5.4 GTX 1080 Graphic Card
To request 2 chunks each with 8 cpus and two GTX1080 cards use:

```
--res=rset=2:ncpus=8:gtx1080=2
```

For offload jobs CCS sets the environment variable: `CUDA_VISIBLE_DEVICES=0,1`

For jobs mapped on a GPU node but not requesting the GTX1080 accelerator CCS sets the environment variable: `CUDA_VISIBLE_DEVICES=1024` which is an invalid value. Refer to `$PC2SW/examples` for example scripts.

5.5 (Un)selecting specific node types
Normally, you won't care about this question, because you just request cores, memory, accelerators, or licences and CCS cares about the mapping. However, for benchmarking purposes it may be useful to (un)select specific node types. For this purpose, we provide resources of type Boolean to (un)select node types. `ccsinfo -a` shows the available resources.

For example: if you want to run a job only on the large compute nodes (128GB memory) set wash=true

```
--res=rset=2:ncpus=5:wash=t
```

This requests 2 chunks each with 5 cores and wash==true.

To exclude washington and smp nodes use:

```
--res=rset=2:ncpus=5:wash=f:smp=f
```

5.6 Large Compute Nodes and SMP Nodes
Using an SMP node is implicitly specified by the amount of the requested memory. If mem is larger than 64g, then the chunk will be mapped to such a node.
6. Online Documentation and Contact

![Fig. 2: The PC² Wiki](image)

We use a Wiki to provide up to date information about systems. Refer to [https://wikis.uni-paderborn.de/pc2doc](https://wikis.uni-paderborn.de/pc2doc).

In the Wiki we document relevant hardware and software changes, inform about system status (e.g., maintenance or failures), describe the system architecture, available software packages and available file systems. Further we provide HowTos, FAQs and give hints for tuning application performance.

In `$PC2SW/examples` you will find example scripts for submitting jobs.

In `$PC2SW/doc`, you will find manuals of the workload-manager and this document.

Use the man pages. E.g., `man ccsalloc` or `man module`.

In case you need assistance send an Email to [pc2-support@upb.de](mailto:pc2-support@upb.de)