IB 131-2015/022

Dokumentation des Compute Clusters von FA

Michael Schäfer, Martin Geier

DLR

Institut für Faserverbundleichtbau und Adaptronik
Braunschweig

05/2015

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtzgemeinschaft
Institut für Faserverbundleichtbau und Adaptronik

IB 131-2015/022

Dokumentation des Compute Clusters von FA

Braunschweig, May 2015

This report contains:

13 pages

Director of the Institute:
Prof. Dr.-Ing. M. Wiedemann
[illegible]

Authors:
Michael Schäfer
[illegible]

Dipl.-Ing. Martin Geier
[illegible]

Head of Section:
Dr.-Ing. Tobias Wille
i.A. [illegible]

Compute Cluster Documentation

Compiled by
Martin Geier (DLR FA-STM)

Authors
Michael Schäfer (INIT GmbH)
Martin Geier (DLR FA-STM)

Date
22.05.2015

DLR

Table of contents

1. Introduction

The cluster consists of a master node and several computing nodes. In order to start a job, the user copies the input data to the master node (see section 2) and starts the job using the prefered interface (see section 3 for setup) and the wrapper commands (see section 4). The queuing system waits until the demanded resources (licenses, node) are available and copies all the specified data to the computing node. After the computing node has executed the job, the queuing system automatically copies the results back to the master node. A schematic of the cluster structrue and the typical job execution are shown in Figure 1 and 2. The details of the available hardware is given in Table 1. If you have not been added to the cluster user group, please ask the IT-Manager of your department.

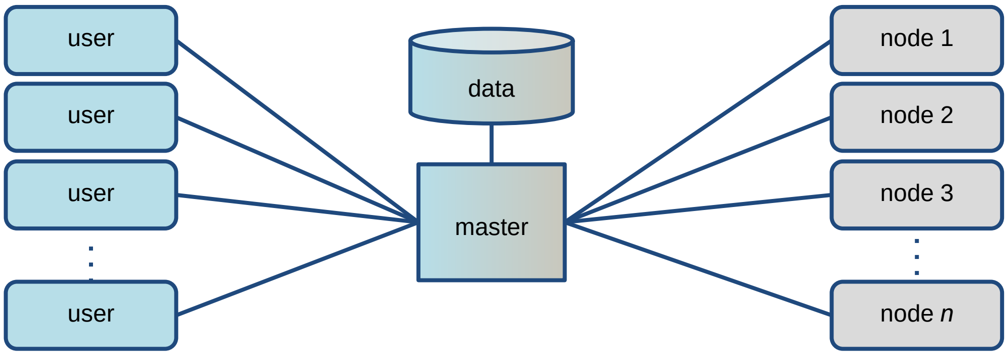


Figure 1: schematic structure of the cluster

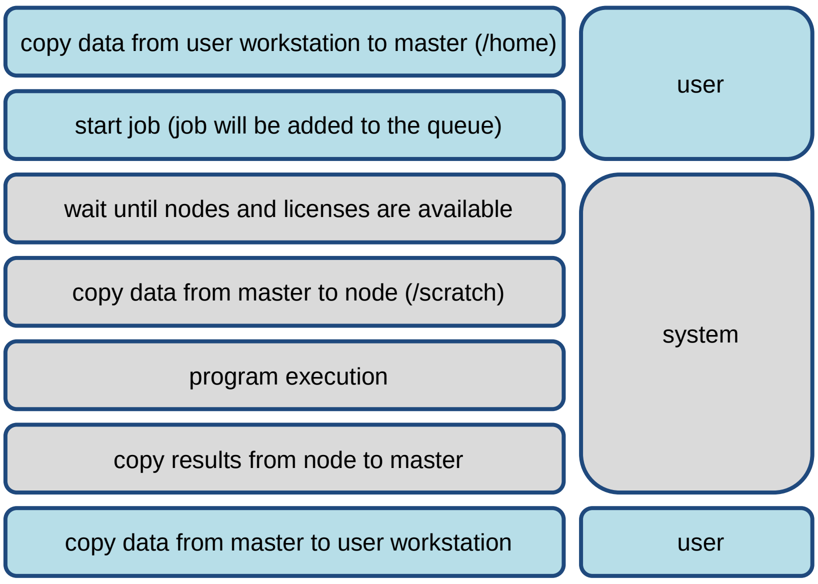

copy data from user workstation to master (/home)
start job (job will be added to the queue)
wait until nodes and licenses are available
copy data from master to node (/scratch)
program execution
copy results from node to master
copy data from master to user workstation
user
system
user

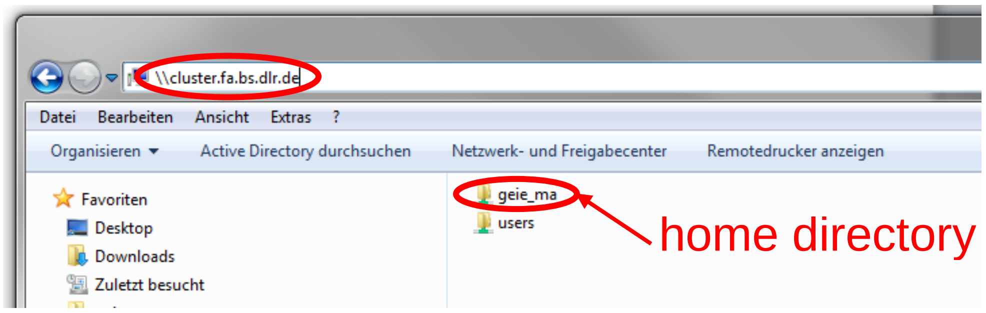
Figure 2: scheme of a typical job execution

description	cpu	memory	storage
master	2x Intel Xeon E5-2407 @2,2GHz	32 GB	15 TB (3x5)
node 1	2x Intel Xeon E5-2690 @2,9GHz / 8 Cores	128 GB	600 GB (2x300 SAS - Raid 0)
node 2	2x Intel Xeon E5-2643 @3,3GHz / 4 Cores	128 GB	600 GB (2x300 SAS - Raid 0)
node 3	2x Intel Xeon E5-2643 @3,3GHz / 4 Cores	128 GB	400 GB (2x200 SSD - Raid 0)
node 4	2x Intel Xeon E5-2690 @2,9GHz / 8 Cores	128 GB	600 GB (2x300 SAS - Raid 0)
node 5	2x Intel Xeon E5-2643 @3,3GHz / 4 Cores	128 GB	400 GB (2x200 SSD - Raid 0)
node 6	2x Intel Xeon E5-2643 @3,3GHz / 4 Cores	128 GB	400 GB (2x200 SSD - Raid 0)

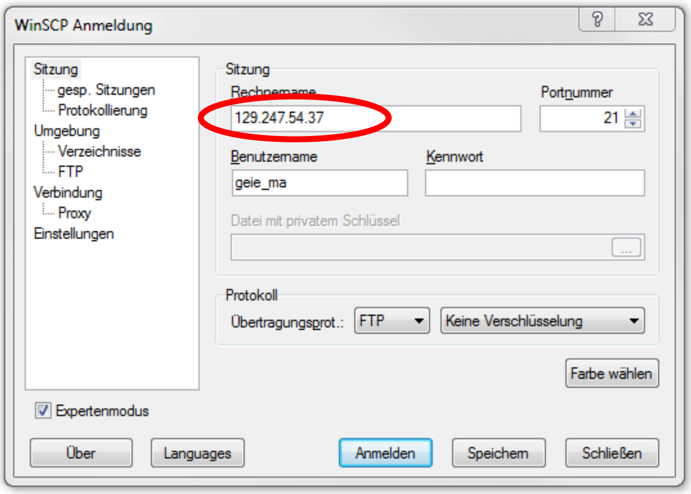
Table 1: list of the available hardware

2. File Transfer

- Windows file transfer (Windows Explorer) `\\cluster.fa.bs.dlr.de`



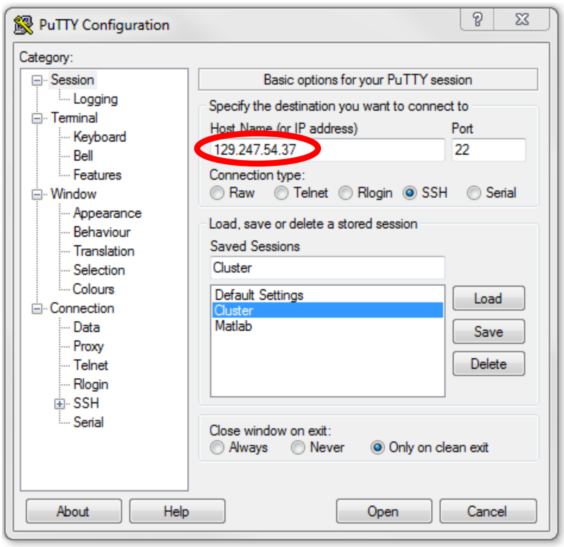
- (S)FTP (FileZilla[1], WinSCP[1])
- SCP (Putty[1], WinSCP[1])



Sitzung
Rechnername: 129.247.54.37
Portnummer: 21
Benutzername: geie_ma
Kennwort:
Datei mit privatem Schlüssel:
Protokoll
Übertragungsprot.: FTP
Keine Verschlüsselung
☑ Expertenmodus

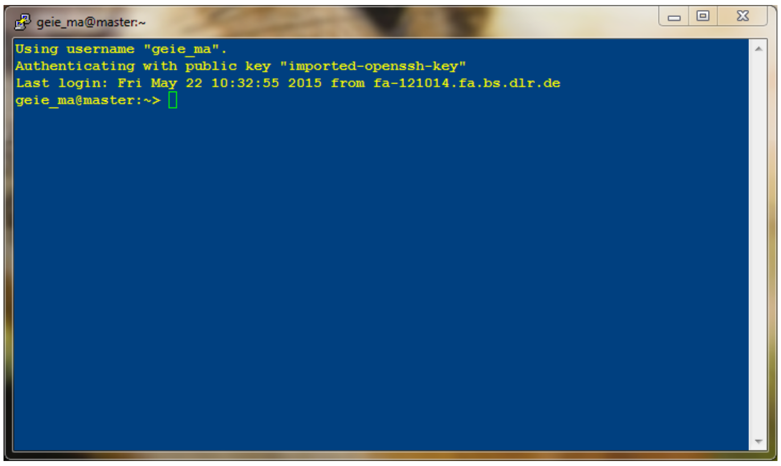
3. Interface Setup

- Secure Shell SSH (Putty[1])
    - Host Name or IP address: `129.247.54.37`



Specify the destination you want to connect to
Host Name (or IP address): 129.247.54.37
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Load, save or delete a stored session
Saved Sessions: Cluster
Default Settings
Cluster
Matlab
Close window on exit: Always Never [✓ Only on clean exit]

- Press "Open" and enter a command as shown in 4.2



```
Using username "geie_ma".
Authenticating with public key "imported-openssh-key"
Last login: Fri May 22 10:32:55 2015 from fa-121014.fa.bs.dlr.de
geie_ma@master:~>
```

- X11 via SSH (NX, Xming[1], VcXsrv, Exceed)

[1] available at \\bsfait00\fa\Cdrom or \\bsfait00\fa\Cdrom_FA-BASIS-SFR

4. Compute-Cluster Documetation

4.1. Access

The cluster is located at `cluster.fa.bs.dlr.de` (`129.247.54.37`). Shell access is possible using ssh (putty). It is also possible to tunnel X-Windows through ssh using for example Xming on Windows or native Linux. It is mandatory to use trusted X11 forwarding. The cluster also offers the secure shell file copy protocol (scp) and allows native ftp access for efficient transfer of large files.

It is very convenient to use key-based login instead of the user password. To achieve this, a rsa-key needs ot be generated and the public-key installed in the users homedirectory on the cluster. For detailed information on how to achieve this with putty, have a look at the tutorial here or see http://www.howtoforge.com/ssh_key_based_logins_putty. Please keep in mind when using vi to paste the key into the authorized_keys file that vi is a peculiar beast and needs to be put into input mode with the i key before paste, otherwise it doesn't work properly and parts of the paste will be lost. To leave input mode and save, press ESC then : followed by q and ENTER.

Examples

Start a XTerm on the cluster from a Linux host using ssh

```
> ssh -C -X -Y -f username@cluster.fa.bs.dlr.de /usr/bin/xterm -title
username@cluster
```

4.2. Commands

Common usage

```
<wrapper command> [wrapper arguments*] [inputname/ inputfile] [job
arguements*]
```

```
<wrapper command> [-d] [-f file] [-O mode] [-t time] [-p priority]
[-C constraint][-D dependency] [-w node] [-L licenses] [-e env] [-i]
[-S] [-X] [-n] [-B spec] [-x pattern] [-J jobname] [--] [inputname]
[job arguments]
```

* = optinal
Exampels see below on page 9.

4.2.1. Wrapper commands

abq6131	Queue a job for abaqus 6.13
abq6141	Queue a job for abaqus 6.14

ansys145	Queue a job for ansys 14.5
ansys150	Queue a job for ansys 15.0
ansys160	Queue a job for ansys 16.0
lsdyna145	Queue a job for lsdyna 14.5
lsdyna150	Queue a job for lsdyna 15.0
lsdyna160	Queue a job for lsdyna 16.0
nast20130	Queue a job for nastran 2013
nast20131	Queue a job for nastran 2013.1
nast20140	Queue a job for nastran 2014
pat20122	Queue a job for patran 2012.2
pat20130	Queue a job for patran 2013
pat20140	Queue a job for patran 2014
marc2013	Queue a job for marc 2013
mat2013a	Queue a job for matlab 2013a
py27	Queue a job for python 2.7

To check for new versions on the cluster, you can type the fixed part of the command (e.g. "abq") and press the tab button three times.

4.2.2. Wrapper arguments

-d	The current directory including all subdirectories is copied to the compute node prior to execution. Cluster results are excluded (cluster.r*).
-f <file>	The given file is treated as input, copied to the compute node and in case of an archive extracted prior to execution. Supported as of now are text files, tar, tar.gz and zip archives. (The extension doesn't matter, the type is determined by calling 'file -b --mime-type <file>')
-N	Do not copy any input files to the compute node

-O <mode>	Set the output mode for the job	
	dir	The result is returned in a subdirectory
	tar	The result is returned as a tar archive
	tgz	The result is returned as a gzip'd tar archive
	zip	The result is returned as a zip archive
	sync	The result is returned in the current directory, only newer files are copied back from the compute node
	none	The result is ignored and lost once the job is completed
-t <time>	Execute the job at the given time, for details see the manpage of sbatch, argument --begin	
-p <priority>	Set the job priority, for now supported priorities are "high", "normal' and "low"	
-C <constr>	Request a specific feature for execution, this determines the node the job is executed on. For details see the manpage of sbatch, argument -C. As of now there is only one defined feature, "ssd", which requests the job to be executed on a node with a solid state drive instead of a regular hard drive.	
-X	Request an exclusive node, which is not shared with other running jobs	
-S	Allow job to be shared with others, this allows for more jobs to run on a single node than the default, but not an unlimited amount.	
-w <node>	Request a specific node for execution	
-L <count>	Request a specific amount of licenses	
-D <dep>	Set job dependency, this can either be a numeric jobid, in which case the job will be executed after the given job terminates, or it can be given in sbatch dependency syntax. When used with wildcard-input the dependency will be assigned to the first job created, except in parallel mode where the dependency is assigned to all created jobs.	
-P	Allow parallel job execution when using wildcards	

-e <env>	Request a specific environment. For now we support ifort9_1, ifort11_0 and ifort2011. It is possible to request multiple environments by given a comma-separated list.
-n	Do not execute the actual command, but schedule the job (for testing)
-B <spec>	Request a specific node amount of sockets, cpus and threads in the format socket:core:thread, see the manpage of sbatch for details.
-x <pattern>	Exclude the given pattern from copy after the job has been executed, i.e. don't copy matching files back to the cluster head. Wildcards must be passed in single quotes ('). It is possible to use -x multiple times.
-J <jobname>	Use the given jobname, this can be useful in combination with -D singleton, see the manpage of sbatch for details regarding singleton.
-i	Wait for job completion
--	Force end of arguments, this is required if the inputname is omited and the job arguments start with an - (see matlab example below)

- When neither -d nor -f are given, the inputname is assumed to be the input file and that file is copied to the compute node. The default output mode if to place the result in a subdirectory.
- The inputname may contain wildcards, but it is mandatory to include the argument in single-quotes (') in this case, otherwise the command won't work as expected, see the examples.
- For Abaqus it is possible to omit the input name completly, in that case it is mandatory to use -d or -f and all arguments are treated as abaqus commandline.
- Any arguments given after the input name are passed on to the job.
- The default job priority is normal and the job is queued for immediate execution.
- Any output has the format `cluster.r<jobid>` with appended suffixes. The console standard and error output is placed in a file with the extension .log.

Examples

Run a simple ansys job
```
> ansys145 ansys.txt
```

Run all .txt-files as a sequence of ansys jobs
```
> ansys145 '*.txt'
```

Run all .txt-files as parallel executed absys jobs
```
> ansys145 -P '*.txt'
```

Run a simple ansys job, exclude *.err and *.log from copy back
```
> ansys145 -x '*.err' -x '*.log' ansys.txt
```

Run a simple ansys job on a node with two sockets and eight cores
```
> ansys145 -B 2:8 ansys.txt
```

Run a simple ansys job in singleton mode with a special job name (all jobs with that name will be serialized)
```
> ansys145 -J "Ansys Series 2" -D singleton ansys.txt
```

Run a simple ansys job with low priority
```
> ansys145 -p low ansys.txt
```

Run a simple ansys job on node 4
```
> ansys145 -w node4 ansys.txt
```

Run a simple ansys job at a 18:00 today
```
> ansys145 -t 18:00 ansys.txt
```

Run a simple ansys job after the job with the id 12345 has finished
```
> ansys145 -D 12345 ansys.txt
```

Run an abaqus job with multiple input files in the current directory and 001_abaqus_input as job name
```
> abq6131 -d 001_abaqus_input
```

Run a nastran job with multiple input files in the current directory and copy the results back to the current directory
```
> nast20130 -d -O sync myjob.bdf
```

Run an abaqus job with multiple input files and an additonal user routine
```
> abq6131 -d 001_abaqus_input user=my_subroutine
```

Run an abaqus job with an archive as input
```
> abq6131 -f 001_abaqus_input.tar.gz 001_abaqus_input
```

Run an abaqus job with full command line and the current directory as input
```
> abq6131 -d job=001_abaqus_input
```

Run an ansys job on a node with a solid state drive
```
> ansys145 -C ssd ansys.txt
```

Run an ansys job with the full research feature aa_r
```
> ansys145 ansys.txt -p aa_r
```

Run an ansys job with the full research feature aa_r and high priority
```
> ansys145 -p high ansys.txt -p aa_r
```

Run a matlab job with full commandline, the current directory as input and a custom license
```
> mat2013a -d -- -r "mycommand" -c /home/f_fainit/.licenses/matlab.lic
```

4.2.3. Job history

Usage

```
shistory <date-spec>
```

Description

The command `shistory` provides a simplified access to the job history from the accounting database. The script uses the `sacct` command, which can be used to access the underlying data in a more general way.
Optionally, a date specification can be given, this is the date from which onwards the history will be shown. It is very literal, for example "5 days ago" or "2 weeks ago". The specification has to be given in quotes.

Examples

List all jobs that have been queued today

```
> shistory
```

List all jobs that have been queued last week

```
> shistory "1 week ago"
```

4.2.4. License availability

Usage

```
slminfo [-a] [-m] [-l] feature
```

Description

The command slminfo provides information about the floating license availabilty of a feature. A feature can be just the program name, like abaqus or in some cases a program, for example with ansys, a specific feature, like aa_r or aa_t_a.

-a	Show the full result of the underlying license server query
-m	Return the result machine-readable
-l	List all supported features

Examples

Show the license availability for abaqus

```
> slminfo abaqus
```

List all available features

```
> slminfo -l
```

4.2.5. Interactive node allocation

A node can be allocated for interactive use by using `salloc`.

Examples

Request any node for a shell, for details see the manpage.
```
> salloc
```

Request a specific node, node 5, for a shell
```
> salloc -w node5
```

4.2.6. Native Cluster Commands

squeue	Display the contents of the cluster queue
sinfo	Displays the status of the cluster nodes
scancel	Cancels a job by its job id
salloc	Allocate a node for interactive use
sbatch	Submits a shell script to the cluster queue
srun	Executes a command on the cluster without queue
scontrol	Cluster configuration, control and job modification
sacct	List past jobs from accounting database

All commands have man pages, use 'man command' to read them.

Temporary directories

When a job is running on a compute node, the temporary data of the job is located at `/scratch/job-<id>` (for example `/scratch/job-12345`) and the temporary data at `/tmp/job-<id>`. Both directories are only accessible by the job owner.

Executed Commands

When using the wrapper script the following commands are executed with the real executables (This is not the way the cluster is used, but to understand what will happen).

Ansys 14.5 / 15.0

```
ansys145 -b -p aa_t_a -i <inputfile> [job arguments]
```

If -p is given within the job arguments the -p aa_t_a will be omited to allow the user to use a different feature (like aa_r for the full research one).

Nastran 2013

```
nast20130 <inputfile> batch=no [job arguments]
```

Patran 2012-2

```
p3 -b -sfp <inputfile> [job arguments]
```

Marc 2013

```
run_marc -jid <inputfile> -b no -v no [job arguments]
```

Abaqus 6.13 Job

```
abq6131 job=<inp_without_extension> interactive [job arguments]
```

Matlab R2013a

```
matlab -nodesktop -nosplash -nodisplay -nojvm -r <inputfile> [job
arguments]
```

Python 2.7

```
python2.7 <inputfile> [job arguments]
```

4.3. Access Control

The access to the cluster is limited to the group `fa_compute_mf`, which is managed using CoMet. New users automatically get their home directory created upon first login. The pathname of the home directory can be configured using the Active Directory Managment Tools, but the default settings is ok.