Report

Title: Preparing to run on Zenobe

Author(s): Danielle Coulon **Date :** 2015/10/26

Approver: Laurent Brognara

Ref: PRACE_T1FWB-NR-005-03

Project identification

Cenaero references	
Project code :	PRACE_T1FWB
Project short description :	PRACE Supercalculateur Tier-1 – Infrastructure de Recherche (lié au programme ESFRI)
Project manager :	Serge Bogaerts
Client / Partner references	
Purchase order reference :	Convention N°1117545
Other reference:	DGO6/DPR/DPR/ESFRI/Convention 1117545
Contact person :	Pol Flamend <pol.flamend@spw.wallonie.be>

Distribution list

	All users that are authorized to log in Zenobe.
CU PRACE DGO6	Pol Flamend
CU PRACE Universités	Benoit Champagne, UNamur et Président du CECI
	Bernard Knaepen, ULB
	Christophe Geuzaine, ULg
	Philippe Chatelain, UCL
	Roberto Lazzaroni, UMONS
CU PRACE Cenaero	Koen Hillewaert
	Serge Bogaerts
GT CU PRACE	Damien François, UCL
	David Colignon, ULg
	Raphaël Leplae, ULB
	Danielle Coulon, Cenaero

List of Revisions

Rev.	*Status*	*Date*	*Author(s)*	*Purpose*
00	DFT	2014/02/27	D. Coulon	First version for comments
01	DFT	2014/04/24	D. Coulon	First revision
02	FIN	2015/10/12	D. Coulon	Completing §2.3, §3.1 and §4.7. Updating §3.2, §4.1, §4.5.2, §4.6.1, §4.6.3 and §4.6.6 (MPI). New §4.6.9 (former moved as §4.6.10)
03	FIN	2015/10/26	D. Coulon	Updating §4.6.3 and §4.7.1

Centre de recherche en aéronautique – Cenaero
Association sans but lucratif
Bâtiment Eole
Rue des Frères Wright, 29 – 6041 Gosselies

Attachments

None

Abstract

The present document describes the Tier-1 supercomputer of the *Fédération Wallonie-Bruxelles* operated by Cenaero. The machine is named `zenobe` after Zenobe Gramme, the Belgian electrical engineer known as inventor of the direct current dynamo.

Contents

1. Introduction

The present document describes the Tier-1 supercomputer of the *Fédération Wallonie-Bruxelles* operated by Cenaero. The machine is named zenobe after Zenobe Gramme, the Belgian electrical engineer known as inventor of the direct current dynamo.

This document focuses on :

- The European PRACE context
- The Cenaero environment
- How to use zenobe computation facilities.

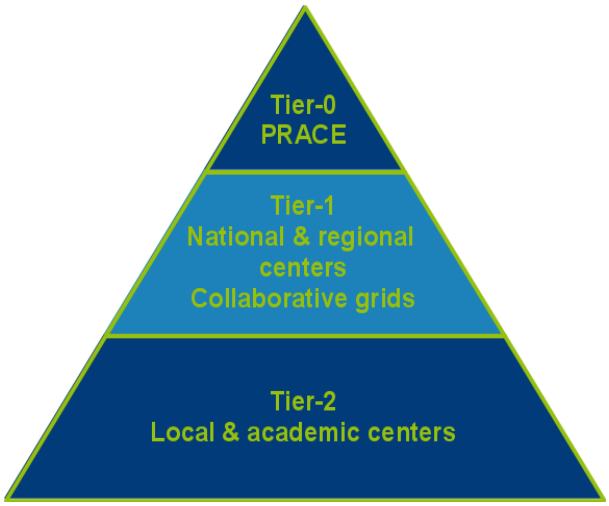
2. Tier-1 Supercomputer of the *Federation Wallonie Bruxelles*

2.1. PRACE research infrastructure

PRACE, the Partnership for Advanced Computing in Europe, created a persistent pan-European Research Infrastructure providing leading High Performance Computing services and top 10 supercomputers for scientific excellence. PRACE enables world-class science and engineering for European academia and industry. Belgium is a member of PRACE project since October 2012.

PRACE infrastructure follows a Tier approach:

- Lower levels to develop techniques
- Higher levels to push them to large scales



In *Fédération Wallonie-Bruxelles*, Tier-2 in universities are managed by the CÉCI (Consortium des Équipement de Calcul Intensif) and Cenaero has been appointed by the Minister Nollet for the procurement and operation of the Tier-1 machine. The largest share of the supercomputer is devoted to academic research. The use of the infrastructure for industrial applications and by an ever-growing panel of industrial users is an other objective of the project lead by Cenaero.

2.2. Tier1 supercomputer hosted at Cenaero

It is based on the existing cluster procured in 2011 within the SINUS project (Part of the FEDER 2007 Walloon project) That cluster was made of 274 compute nodes (Bull B500 blades) that provided 3288 compute cores. The extension installed in December 2013, added 8208 compute cores to reach a total of 11496 cores (the target was to procure a 10k core machine).

Computation facilities are installed in a mobile data center which hosting capacity has been extended to 480 kW with a N+1 redundancy.

2.3. How to acknowledge the use of zenobe.

In French:

Les présents travaux ont bénéficié de moyens de calcul mis à disposition sur le supercalculateur Tier-1 de la Fédération Wallonie-Bruxelles, infrastructure financée par la Région wallonne sous la convention n°1117545.

In English:

The present research benefited from computational resources made available on the Tier-1 supercomputer of the Fédération Wallonie-Bruxelles, infrastructure funded by the Walloon Region under the grant agreement n°1117545.

3. CENAERO environment

3.1. Getting access

3.1.1. Academic users

Potential users need a CECI account. Please follow, the CECI procedure on the CECI site in order to create this login.

http://www.ceci-hpc.be

Access to Zenobe is granted solely through 'projects'. A project must be created by a CÉCI user who holds a permanent position (e.g. Professor, etc.) within a member university. Other CÉCI users can then register to that project and will have access to the resources on Zenobe (CPU time, disk space, etc.) that are allocated to the project.

Projects can be submited though the dedicated web page, where a form must be filled with specific information. An overview of the process can be seen here.

Please note that access to Zenobe is subject to specific conditions.

To connect to zenobe, CECI Users are invited to follow the procedure described in CECI Zenobe quickstart.

3.1.2. Other users

All other users must contact Cenaero to get access to the supercomputer.

To connect to zenobe, these users will have to connect first to a gateway named hpc.cenaero.be via SSH and therefrom again via SSH connect to zenobe.

3.2. Policies

- **Password**
  - Do not share your password

- Password rules are enforced:
  - 8 characters minimum with at least 1 special character, 1 Uppercase, 1 lowercase and 1 digit.
  - Palindrome, case change only, similar, simple or rotated passwords are rejected.
- The maximum validity period of password is 180 days.

- **Frontal nodes**
  - No heavy tasks : the front-end nodes must be used for ssh session, editing or compiling. Do not run heavy computation on them.
  - 2000mb virtual memory limit is set per process. It can be increased up to 4000mb.

- **Computing nodes**
  - **Any computation tasks must be launched through the scheduler.**
  - **Use resources efficiently.**
  - **ssh on nodes is allowed for debugging and monitoring purpose only when users have a running job on this node.**

- **Filesystems**
  - /home directories are meant for codes, scripts and small data set.
  - /project directory is meant to share, store data and results during the project's life.
  - /SCRATCH directory is a working area and it must be used for temporary data created during jobs execution. It is not designed to store or archive data during a long period (use /project instead). Note that the SCRATCH directory can be “scratched” at any time with no guaranty of backup or recovery.

- **Quotas**
  - /home : per user quotas 130GB fixed. Check it with the command:

    ```
    $quota -u <username>
    ```

  - /project : per project quotas
    - minimum default quotas set to 0GB
    - requests credited until 300GB
    - Larger amount must be motivated.
  - /SCRATCH :
    - per team quotas : minimum/default fixed to 100GB
    - per project quotas : requests credited until 1000GB. Larger amount must be motivated.

- Hard limit is set to 130% of the soft limit with a 7 days grace period.
- Check it with the command:

```
$mmlsquota -g <groupname>
```

3.3. Working with project

For all projects :

- A unix group will be created.
- Specific directories will be created. Quotas will be set for the group upon request by the the project manager.
- A specific PBSpro directive must be used (see Running PBSpro jobs paragraph).

4. Computing on zenobe

4.1. Compute nodes hardware configuration

Summary

- 34 chassis Bull.
- 616 nodes
- 11496 cores
- 29664 GB Total memory
- 217.538 TFLOPS Theoretical Peak performance

Blades

	Processor	
CPU	**Westmere** **2 x Intel x5675 (6cores)**	**Ivy Bridge** **2 x Intel E5-2697v2 (12cores)**
#of cores per node	**12**	**24**
Total of nodes	**274**	**342**
Total number of cores	**3288**	**8208**
Instruction set	**SSE4.2**	**AVX**
CPU-Clock	3.06GHz	2.7GHz
Cache L1	Instruction: 32KB private to each core data: 32KB private to each core	Instruction: 32KB private to each core data: 32KB private to each core
Cache L2	256KB private to each core	256KB private to each core
Cache L3	12MB shared by 6 cores	30MB shared by 12 cores
2 QPI links (two sockets connection)	Total bandwidth: 25.6GB/s (Bus speed 6.4GT/s)	Total bandwidth: 32GB/s (Bus speed 8.0GT/s)

Maximum Double Precision Floating Point per cycle per core	4	8
Theoretical peak performance per core	12.24 GFLOPS	21.6 GFLOPS
Theoretical peak performance per node	146.88 GFLOPS	518.4 GFLOPS
Theoretical global peak performance	40.245 TFLOPS	177.293 TFLOPS
Hyper-Threading	OFF	OFF
Turbo-Boost	OFF	OFF
Memory		
Total memory per node	Fit nodes 24GB Fat nodes 48 GB XFat nodes 192 GB	64GB
Memory per core	Fit nodes 2GB Fat nodes 4GB XFat nodes 16GB	2.66GB
Memory type	Fit nodes DDR3 - 1333MHz Fat nodes DDR3 - 1333MHz XFat nodes DDR3 - 800MHz	DDR3 - 1866MHz
Memory bandwidth	32GB/s read/write	59.7GB/s read/write
Interconnect		
Ethernet	2 x Gigabit Ethernet ports	2 x Gigabit Ethernet ports
Infiniband	Mellanox ConnectX2 single QDR	Mellanox ConnectX3 single FDR
Disk		
Local disk	1 SSD 64GB	1 SSD 64GB

There are 18 Westmere Fat nodes (one chassis) and 4 Westmere Xfat nodes.

Note that memory values reported in the previous table are “commercial/physical” values not those effectively available for computation (pay attention to PBSpro limits, see advice paragraph).

The AVX technology is a major hardware evolution of Intel processors technology. Linux supports AVX since kernel version 2.6.30 and therefore, in order to fully take advantage of this new technology, the Operating System had to be upgraded from RedHat/Centos 5.7 to 6.4 at least.

The Ivy Bridge and Westmere nodes default OS is the RedHat/Centos release 6.4. Only nodes 0233 to 0270 remain in CentOS 5.7.

Hostnames

The compute nodes are labeled from 0001 to 0616 with the prefix “node” in the following way:

- Westmere nodes : chassis 0 to 15
  - Fit nodes: node[0001-0108,0126-0270].
  - Fat nodes : node[0109-0125] – chassis 7

- XFat nodes : node[0271-0274] – chassis 0

- Ivy Bridge nodes : chassis 16 to 34
  - node[0275-0616]

4.2. zenobe filesystems

4.2.1. Nodes local filesystem

Each compute node has a local SSD disk which is dedicated administration tasks. The local disk is not available for computation.

4.2.2. NetApp

New configuration	
/home	**13 To**
/home/acad	**23.75 To**
/home/ta	**2.3 To**
/projects	**23.75 To**
/projects/acad	**23.75 To**
/softs	**475 Go**

Quotas are set per user on /home and per project (group) on /projects.

Currently, there is no backup.

Snapshots are done for /home, /projects and /softs. For /home and /projects, they are taken nightly at midnight and hourly at 8:00, 12:00, 16:00 and 20:00. The 6 last hourly and the 2 last nightly snapshots are kept. For /softs only the 1 nightly is kept.

4.2.3. GPFS

New configuration	
/SCRATCH	**349 To**

Quotas are set per project and per organizational group. GPFS is reliable but data are not backed up.

http://en.wikipedia.org/wiki/IBM_General_Parallel_File_System

4.3. Front-end nodes

zenobe

Bullx R423-E3 server equipped with :

- 2 x Intel E5-2697 v2 (12 cores) 2.7GHz

- 64GB of memory

OS : RedHat 6.4

This node is the main front-end.

theophile

Bullx R423-E3 server equipped with :

- 2 x Intel E5-2697 v2 (12 cores) 2.7GHz
- 64GB of memory

OS : RedHat 5.7

This node is available as front-end only for porting purpose and backward compatibility. The access is opened on demand.

4.4. Interconnect

4.4.1. Ethernet network

The Ethernet network interconnects all the entities of the cluster and is used for administrative tasks (installation, management, supervision, NFS, PBS communications,...)

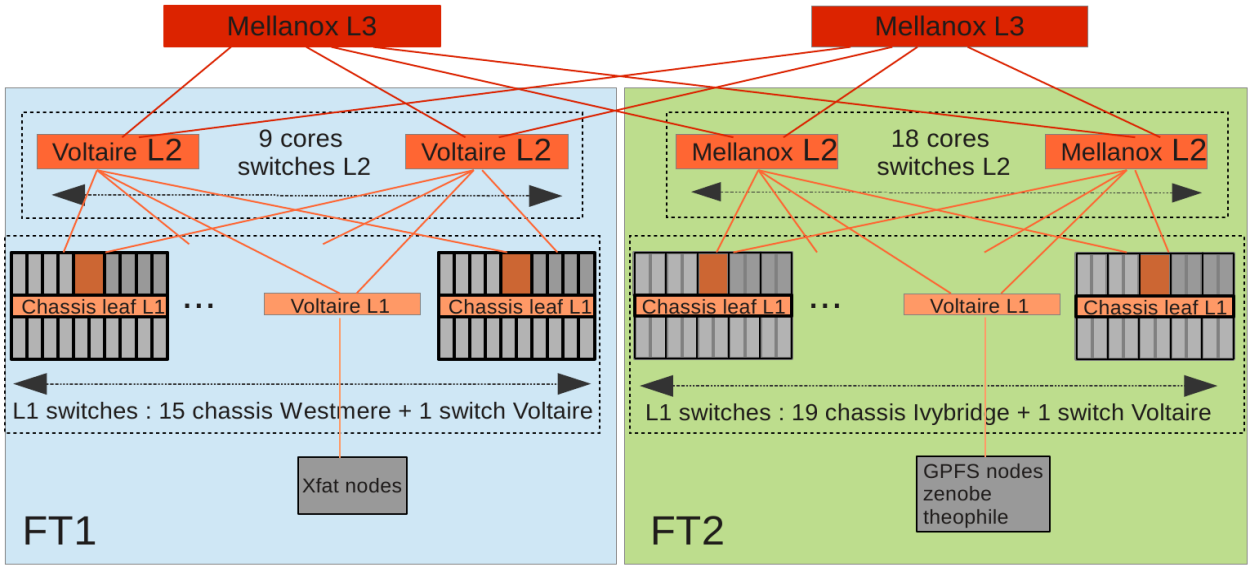
4.4.2. Infiniband network

The infiniband network is composed of 3 levels:

- Level 1 L1 : leaf level
- Level 2 L2 : core level
- Level 3 L3 : top level

The higher level L3 connects 2 non-blocking fat-tree which include :

- FT1 : 15 Westmere nodes chassis + XFat nodes
- FT2 : 19 Ivy Bridge nodes chassis + frontal nodes + GPFS nodes



Inside each fat tree, communication between entities are non-blocking. Between the 2 fat-tree, communications are blocking. The link between Level 3 switches has been designed to satisfy GPFS traffic but not to support computation between the two blocks.

4.5. Porting and developing

4.5.1. Software organization

Users can install tools in their home directory with respect to their quotas, but it is a good idea to discuss with the HPC and Infrastructure team in order to optimize the collaborative work, not duplicate applications and save disk space.

Shared tools and applications are installed in the /softs partition.

Note that the access to commercial software is restricted (Samcef, Abaqus, Ansys, Fluent, Morfeo, elsA,...).

New software can be installed with the agreement and collaboration of the HPC and Infrastructure team.

These applications are loaded into your environment through the use of modules.

When you log in, no modules are loaded by default.

To see all available modules, issue this command:

```
$module avail
```

It displays a complete list of modules that are available to be loaded into your environment.

You can load, unload, and even swap/switch modules using the module command, i.e.,

```
$ module load <module_name>
$ module unload <module_name>
$ module switch <loaded_old_module> <new_module_name>
```

To see which modules you currently have loaded, issue the following command:

```
$ module list
```

To display a list of environment variables and other information about an individual module "module_name", invoke the following:

```
$ module show <module_name>
```

To remove all modules from the environment use:

```
$ module purge
```

For advanced topics use :

```
$module help
```

4.5.2. Compiling your application

1. **The zenobe/frontal1 OS is RedHat 6, programs compiled on it will almost certainly not run on the subset of Westmere nodes where OS remains RedHat 5.**

2. For language compilers, Intel® C++ Compiler version 11.1 and later and Intel® Fortran Compilers support Intel® AVX through compiler switches, and both compilers support automatic vectorization of floating-point loops.

   Compiler options:

   - To run only on Ivy Bridge processors: `-O2 (or -O3) -xAVX`
   - To run only on Westmere processors: `-O2 (or -O3) -xSSE4.2`
   - To run on all types : `-O2 (or -O3) -axAVX -xSSE4.2`
   - The -m*code* and -x*code* options tell the compiler to generate code specialized for the processor that executes your program. But code generated with -m options should execute on any compatible, non-Intel processor with support to the specified instruction set (*code*) rather than code generated with -x option can only execute on a subset (*code*) of Intel processors. The -m and -x options are mutually exclusive.

     The -x option enables additional optimizations not enabled with options -m nor with option –ax.
   - The -ax option tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.
   - Releases available on `zenobe` (Previous versions installed in /softs do not support AVX)

     composer_xe_2011_sp1.7.256 → icc 12.1.0 (gcc 4.4.7 compatibility)

     composer_xe_2013.2.146 → icc 13.1.0 (gcc 4.4.7 compatibility)

     composer_xe_2013_sp1.1.106 → icc 14.0.1 (gcc 4.4.7 compatibility)

3. In the GNU Compiler Collection (GCC), version 4.4 supports Intel AVX intrinsic through the same header, `<immintrin.h>`. Other GNU tool-chain support is found in Linux Binutils 2.20.51.0.1 and later, gdb 6.8.50.20090915 and later, recent GNU Assembler (GAS) versions, and `objdump`.

GCC4.4 : Support for Intel AVX built-in functions and code generation is available via -mavx.

Release gcc 4.4.7 is available on zenobe.

4. Recommended compilers are those provided with Intel composer_xe_2013_sp1.1.106 tools suite.

4.5.3. MPI

Intel MPI and OpenMPI librairies are available on zenobe.

OpenMPI are compiled with gcc4.1.2 unless those tag with “-el6” which are built with gcc4.4.7.

Use module command to list the different releases.

The recommended tool is Intelmpi 4.1.3.045.

Prior to using an MPI library, you will have to load an appropriate module for a supported compiler suite.

4.5.4. Debugging your application

Gdb (gcc4.4.7 tool suite), Intel idb provided with Intel composer XE tools suite and Allinea DDT are available for debugging purpose.

Use module command to list the different releases.

4.5.5. Tuning your application

Allinea MAP and Intel Cluster Studio XE tools suite are available for tuning purpose.

Use module command to list the different releases.

TAU, PAPI, SCALASCA and likwid installations are not finalized yet.

4.6. Running PBSpro jobs

PBSpro is the jobs scheduler installed on zenobe. Current release is 12.1.

Consult PBSpro home page for the complete documentation:

http://www.pbsworks.com/SupportDocuments.aspx

4.6.1. Scheduling policies

- Jobs are scheduled by queue priority first and next by jobs priority formula (ncpus+vmem/2097152).
- **Strict_ordering :** Runs jobs exactly in the order determined by the scheduling option settings, ie run the "most deserving job" as soon as possible.
- **Backfill :** Allows smaller jobs to be scheduled around more deserving jobs.

- **Sharing**
  - jobs share nodes by default except if it is explicitly specified in the queue (see queue large) or in the job requirements.
  - Check the nodes default properties, limits and status with the pbsnodes command.
- **Quarterly maintenance window** (dedicated time) : To be confirmed 1 month prior to the maintenance

4.6.2. Jobs Policy

- **Jobs that alter the sake of optimal global functioning of the cluster or that negatively impact other jobs through an abnormal resources usage will be killed.**
- **HPC administrators will wait 12 hours prior to do a non-crucial intervention and stop jobs.**
- **We require that jobs which walltime lasts more than 12 hours must be re-runnable.**
  - Re-runnable means that the job can be terminated and restarted **from the beginning** without harmful side effects (PBS Reference guide terminology).
  - This must be materialized in your PBSpro script by the directive:

    ```
    #PBS -r y
    ```
  - In case of cancellation (node crash, operator intervention,...) re-runnable jobs will be automatically resubmitted by PBSpro and they may belong to one of the following cases:
    - **worst case :**
      1. The job is actually not re-runnable (for instance it is influenced by the output of a previous run in an uncontrolled manner) and will most probably crash (possibly corrupting generated results). The job's owner knows and accepts it;
      2. The job is not restartable but it is not influenced by previous output files, then it will rerun automatically from the beginning (and the job's owner knows and accepts it);
    - **Ad hoc case:** The job's owner is ready to take manual actions to make sure the input and previous output files are adapted adequately for the restart. Then, insert at the beginning of the PBS script, just after the PBS directives:

      ```
      qhold -h u $PBS_JOBID
      ```

      In the case of cancellation and rerun, the PBS server will put the job on hold. At this step, after the modification of your data, you can release the job with the command qrls.
    - **Ideal case:** The job does modify the input files and/or checks the output generated by a previous run adequately. In the case of cancellation and rerun, the job will restart and continue automatically from the last checkpoint.
- Using Westmere and Ivy Bridge nodes in a same job is not allowed.
- Project/accounting

Jobs submission is only allowed through project.

In order to do accounting and to work with the resources (walltime, ncpus, disk space, …) allocated to the project <project_name>, add in your PBSpro script, the directive:

```
#PBS -W group_list=<project_name>
```

4.6.3. Queues

The different queues are :

- **large** : this queue addresses jobs only to Ivy Bridge nodes and is dedicated to large massively parallel jobs. The resource model must not be specified. Following limits are applied to this queue:
  - **Job placement on nodes is exclusive.**
  - Minimum number of cpus = 96 – Maximum number of cpus = 4320 – Walltime maximum = 24 hours
- **main** : This is the default queue. The main queue is a routing queue dispatching jobs in four execution queues :
  - main_wes queue (default): with Westmere nodes.
  - main_ivy queue: with Ivy Bridge nodes.
  - main_wes_fat : this queue accesses Westmere Fat nodes.
  - main_wes_xfat : this queue accesses Westmere XFat nodes and is dedicated to jobs that required large amount of memory. Following limits are applied to this queue:
    - Minimum memory requirement = 47000mb
  - **Job placement on nodes in queue main is shared.**
  - **Use either "#PBS -l model=westmere" or "#PBS -l model=westmere_fat" or "#PBS -l model=westmere_xfat" or "#PBS -l model=ivybridge" to access respectively the main_wes, main_wes_fat, main_wes_xfat or main_ivy queue.**
  - main_wes, main_wes_fat, main_wes_xfat and main_ivy queues can only be accessed through the main routing queue. Do not submit jobs directly in these queues.
  - **Use "#PBS -l select.....osrel=5.7" to access the Westmere nodes still in RedHat 5.7 in the main_wes queue.**
- other restricted access queues (diags, zen, TA_excl, TA_flex, …)
- Reservation can be done by the support on demand .

Check the queue properties and limits with the command : qstat -Qf

The cstat command (non standard) displays nodes/queues jobs repartition.

4.6.4. Scheduler basic commands

- Qsub :

  The command `qsub` allows to submit jobs to the batch-system. `qsub` uses the following syntax:

  `qsub [options] job_script`

  Description of the most commonly used options to qsub:

Input/output	
`-o path`	standard output file
`-e path`	path standard error file
`-j oe (eo)`	joins standard error to standard output (standard output to standard error). oe is the default.
Queue	
`-q <queue_name>`	runs jobs in queue <queue_name>
Notification	
`-M email address`	notifications will be sent to this email address
`-m b|e|a|n`	notifications on the following events: **b**egin, **e**nd, **a**bort, **n**o mail (default) Do not forget to specify an email address (with -M) if you want to get these notifications.
Resources	
`-l walltime=[hours:minutes:]seconds`	requests real time; the default is 12 hours.
`-l select=N:ncpus=NCPU`	requests *N* times *NCPU* slots (=CPU cores) for the job (default for *NCPU*: 1)
`-l select=N:vmem=size`	requests *N* times *size* bytes of memory for each chunk (default is 1GB).
`-l pvmem=size`	request a maximum of size bytes of memory for all processes of the job. Default is 1GB.
`-W depend=afterok:job-id`	starts job only if the job with job id *job-id* has finished successfully
`-l place=`	chooses the sharing, grouping and the placement of nodes when it is allowed in the queue (default is free).
`-l model=<model_type>`	request Westmere or Ivy Bridge nodes when allowed in the queue. model_type values available are westmere or ivybridge.
Miscellaneous	
`-I`	interactive job
`-r y|n`	notifies that job is rerunnable (default no)
`-v`	specifies the environment variables and shell functions to be exported to the job.
`-V`	Declares that all environment variables and shell functions in the user's login environment where qsub is run are to be exported to the job.

- qstat: gets information about running or waiting jobs use.

  Useful qstat options:

-u <username>	to get information about running or waiting jobs use
-f <jobid>	prints full information of the job with the given *job-id*. Note that information such as resources_used.
-n <jobid>	displays on which nodes, the job <jobid> is running
-w	wide format
-T	displays estimated jobs start times
-a	lists all jobs
-x <jobid>	displays information for the finished job <jobid> (Only available during 48h after the job's end)

- qdel <jobid> : deletes the job <jobid>
- tracejob <jobid> : provides full PBSpro status information
- qpeek <jobid> : displays jobs output while it is running
- pbsnodes : gets information on nodes

  Useful pbsnodes options:

<nodename>	provides information on node <nodename>
-a	provides information on all nodes
-l	lists offline nodes

4.6.5. Allocating PBS resources and placing jobs

PBS resources represent things such as CPUs, memory, switches, hosts, Operating System, chassis, time.... They can also represent whether or not something is true or not, for example whether a node is dedicated to a particular project or group.

A chunk is a set of resources that are allocated as a unit to a job. All parts of a chunk come from the same host/node.

A chunk-level resource or a host-level resource is a resource available at host level, per example CPUs or memory. The resources of a chunk are to be applied to the portion of the job running in that chunk. Chunk resources are requested inside a select statement.

Job-wide resource, also called queue-level or server-level resource, is a resource that is available to the entire job at the server or the queue. For example, walltime or pvmem are job-wide resources.

Format for requesting both job-wide and chunk resources:

```
qsub ….                              (non-resource portion of the job)
-l <resource>=<value>                (job-wide request)
```

-l select=[N:][chunk specification][+[N:]<chunk specification>] (chunk-level request)

PBS assigns chunks to job processes in the order in which the chunks appear in the select statement.

Users can specify how the job should be placed on nodes. Users can choose to place each chunk on a different host, or to share a specific value for some resource.

Example: You want four chunks, where the first has two CPUs and 20GB of memory, the second has 4 CPUs and 4GB of memory and the two last ones, one CPU and 40GB of memory:

```
-l select=1:ncpus=2:vmem=20GB+1:ncpus=4:vmem=4GB+2:ncpus=1:vmem=40GB
```

The *place* statement must be used to specify how the job's chunk are placed. The place statement has the form:

```
-l place=[arrangement][:sharing][:grouping]
```

where

- *arrangement* is one of free | pack | scatter| vscatter
- *sharing* is excl | shared | exclhost
- *grouping* can have only one instance group=<resource>

Some resources or placement can be requested by users, other ones are read-only and cannot be modified by users (queues limits).

4.6.6. Multi-processors jobs

Job's node file

For each job, PBS creates a job-specific “host file” which is a text file containing the name of the nodes allocated to that job, one per line. The file is created by PBS on the primary execution host and is only available on that host. The order in which hosts appear in the node file is the order in which chunks are specified.

The full path and name for the node file is set in the job's environment variable $PBS_NODEFILE.

MPI

The number of MPI processes per chunk defaults to 1 unless it is explicitly specified using the mpiprocs resource.

For example, to request two MPI processes for each of the three chunk, where each chunk has two CPUs can be done with the following select statement:

```
-l select=3:ncpus=2:mpiprocs=2
```

The node file then contains the following list:

```
hostnameA
hostnameA
```

```
hostnameB
hostnameB
hostnameC
hostnameC
```

A request of three nodes,such that :

```
-l select=3:ncpus=2
```

will lead to the following node file :

```
hostnameA
hostnameB
hostnameC
```

Open MPI and IntelMPI automatically obtain both the list of hosts and how many processes to start on each host from PBS Pro directly through the $PBS_NODEFILE. Hence, it is unnecessary to specify the `--hostfile`, `--host`, or `-np` options to `mpirun` **if the MPI software default interpretation of this file corresponds to what you want**. For example:

- IntelMPI : default is hostfile which means that duplicated hostname lines are removed.
- OpenMPI : The reordering of the lines is performed in order to group the same nodes.

Open MPI and IntelMPI versions installed on zenobe use PBS mechanisms to launch and kill processes. PBS can track resource usage, control jobs, clean up job processes and perform accounting for all of the tasks run under the MPI.

OpenMP

PBSpro supports OpenMP applications by setting the OMP_NUM_THREADS variable in the job's environment, based on the request of the job.

The value of OMP_NUM_THREADS is set based on the first chunk of the select statement. If ompthreads is requested, OMP_NUM_THREADS is set to this value. If ompthreads is not requested, OMP_NUM_THREADS is set to the value of ncpus resource of the chunk.

Examples:

To run an OpenMP job as a single chunk, for a two-CPU, two-threads job:

```
-l select=1:ncpus=2
```

To run an MPI application with 64 MPI processes and 4 OpenMP threads per process :

```
-l select=64:ncpus=4
```

or

```
-l select=64:ncpus=4:ompthreads=4
```

4.6.7. PBSpro script

A PBSpro script is composed of 3 parts

- Shell (PBSpro always executes the shell startup script)
- PBSpro directives
- Instructions set of your code

Useful PBS Environmental variables available in script:

PBS_JOBID	The job identifier assigned to the job by PBSpro
PBS_JOBNAME	The jobname supplied by the user.
PBS_NODEFILE	The filename containing a list of the nodes assigned to the job.
PBS_O_QUEUE	The original queue name to which the job was submitted
PBS_QUEUE	The name of the queue from which job is executed
PBS_TASKNUM	The task (process) number for the job on this node.
PBS_O_WORKDIR	The absolute path of the directory where qsub was executed.
NCPUS	Number of threads, defaulting to the number of CPUs on the node
OMP_NUM_THREADS	Same as NCPUS.

Basic example:

```
#!/bin/bash
#PBS -q main
#PBS -l walltime=00:15:00
#PBS -l select=1:ncpus=1:vmem=23000mb:mpiprocs=1:ompthreads=1
#PBS -l pvmem=23000mb
#PBS -l place=excl
#PBS -W group_list=bio
echo "---------------- Work dir -------------------"
echo $PBS_O_WORKDIR
cd ${PBS_O_WORKDIR}
echo "------------------ Job Info --------------------"
echo "jobid      : $PBS_JOBID"
echo "jobname    : $PBS_JOBNAME"
echo "job type   : $PBS_ENVIRONMENT"
echo "submit dir : $PBS_O_WORKDIR"
echo "exec dir   : $PBS_JOBDIR"
echo "queue      : $PBS_O_QUEUE"
echo "user       : $PBS_O_LOGNAME"
echo "threads    : $OMP_NUM_THREADS"
echo "------------------ $PBS_NODEFILE ---------------"
cat $PBS_NODEFILE
echo "------------------ Checking limits -------------"
ulimit
echo "------------- Loading environment --------------"
…
```

This script requests :

- a 15 minutes walltime
- 1 chunk with 1 cpu, 23000mb of memory, 1 mpi task, 1 thread.
- Any process in the job using more than 23000mb will be killed.
- An entire node will be dedicated to this job as an exclusive placement is requested.
- It is a bio project.

Note that if more than 23000mb of memory is requested per chunk/process, the job will run on Fat nodes.

MPI script :

```
#PBS -q main
#PBS -l model=ivybridge
#PBS -l walltime=15:00:00
#PBS -l select=12:ncpus=1:vmem=2625mb:mpiprocs=1:ompthreads=1
#PBS -l pvmem=2625mb
#PBS -W group_list=hpc
#PBS -r y
qhold -h u $PBS_JOBID
…
module load ….
…
mpirun your_code
```

This script requests :

- the queue main and Ivy Bridge nodes
- a 15 hours wall-time
- 12 chunks with 1 cpu, 2625mb of memory, 1 MPI task and 1 thread per chunk
- Any process in the job using more than 2625mb will be killed.
- It specifies that the job is re-runnable and that in the case of cancellation and rerun, the job must be put on hold by the server.
- IntelMPI and OpenMPI are PBSpro aware, the machine file ($PBS_NODEFILE) must not be specify. With this resources requirement, the machine file can have the following form as in the main queue the nodes are shared:

```
node0275
node0275
node0275
node0280
node0284
node0284
node0284
node0284
node0292
node0276
node0276
node0277
```

If, instead of selecting 12 chunks of 1 ncpus and 1 mpiprocs per chunk, 1 chunk of 12 ncpus and 12 mpiprocs, is requested, the machine file has the following form :

```
node0275
node0275
node0275
node0275
node0275
node0275
node0275
node0275
node0275
node0275
```

```
node0275
node0275
```

Mixed MPI/OpenMP script :

```
#PBS -q large
#PBS -l walltime=1:00:00
#PBS -l select=5:ncpus=24:vmem=63000mb:mpiprocs=2:ompthreads=12
#PBS -l pvmem=63000mb
#PBS -W group_list=PRACE_T1FWB
…
```

This script requests :

- the queue large
- a 1 hour wall-time
- 5 chunks with 24 cpus, 63000mb of memory, 2 MPI tasks per chunk and 12 threads per MPI task.
- OMP_NUM_THREADS is set to 12 by PBSpro.
- It is a PRACE_T1FWB project.
- The MPI machine file can have the following form:

```
node0340
node0340
node0346
node0346
node0431
node0431
node0425
node0425
```

4.6.8. Job Array

PBSpro provides jobs arrays for running collections of almost-identical jobs.

Job arrays are useful where you want to run the same program over and over on different input files. PBS can process a job array more efficiently than it can the same number of individual jobs.

Each job in a job array is called a “subjob”. All subjobs in a job array share a single job script, including PBS directives and the shell script portion. All subjobs have the same attributes, including resource requirements and limits.

The job script may invoke different commands based on the subjob index. The commands may be scripts themselves.

Job arrays are required to be rerunnable.

Subjobs must satisfy individually the limits set on queues.

Submitting a Job array :

```
qsub -J <index start>-<index end>[:stepping factor]
```

Environmental variables for Job Arrays:

PBS_ARRAY_INDEX	Used for subjobs	Subjob index in job array, e.g. 7

PBS_ARRAY_ID	Used for subjobs	Identifier for a job arrays. Sequence number of job array, e.g. 5222371[].frontal1
PBS_JOBID	Used for jobs, subjobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 5222371[2].frontal1

Example:

```
#!/bin/csh
#PBS -q main
#PBS -l model=ivybridge
#PBS -j oe
#PBS -l select=1:ncpus=1:vmem=1600mb:mpiprocs=1:ompthreads=1
#PBS -l pvmem=1600mb
#PBS -l walltime=00:30:00
#PBS -r y

echo $PBS_O_WORKDIR
cd ${PBS_O_WORKDIR}

if ( ! ($?wk) ) then
 echo  "ERROR: wk variable must be defined <master|slave>"
 exit
endif

if ( $wk == "master" ) then
 echo I am the master process, I prepare the data and launch 4 workers
 master_application
 qsub -J 1-4 -v wk=slave $0
else if ( $wk == "slave" ) then
 echo I am the slave process number ${PBS_ARRAY_INDEX}
 slave_application_${PBS_ARRAY_INDEX} < input.file_${PBS_ARRAY_INDEX}
else
 echo "ERROR : Unknown worker $wk"
 exit
endif
exit 0
```

This can be launched with the command:

```
qsub -v wk=master <script_name>
```

4.6.9. pbsdsh

PBS scripts provide a way to combine calculations into single jobs. The `pbsdsh` command is used to have specific cores execute shell scripts or programs within a multi-core job.

When run without option -c or -n, pbsdsh will spawn the program on all vnodes allocated to the job. With the “-c <copies>” option, the script is spawned copies times. With the “-n <node_index>”, the script is spawned on the node_index-th vnodes allocated. With the -o option, no obit request is made for spawned tasks and the pbsdsh will not wait for the tasks to finish.

Double dash must come after the pbsdsh options and before the program and its arguments.

Each shell script begins execution from the user's home directory, and each shell script needs to include any appropriate change directory commands. Each shell script begins execution with only the PBS and default settings, so each shell script needs to contain the appropriate environment settings or/and the module load commands for any further software modules required for its calculations.

PBS script example using pbsdsh command :

```
#!/bin/bash
#PBS -q main
#PBS -l model=westmere
#PBS -j oe
#PBS -l select=5:ncpus=1:vmem=1600mb:mpiprocs=1:ompthreads=1
#PBS -l pvmem=1600mb
#PBS -l walltime=00:03:00
#PBS -r y

echo $PBS_O_WORKDIR
cd ${PBS_O_WORKDIR}
#
echo $PBS_NODEFILE
cat $PBS_NODEFILE
echo
#
pbsdsh -v -- $PBS_O_WORKDIR/myscript.sh
wait
#
for proc in `seq 0 4`
do
pbsdsh -o  -v -n $proc -- $PBS_O_WORKDIR/myscript$proc.sh
done
wait
#
exit 0
```

With “-o” option or with background launching (&), do not forget to wait until the end of tasks.

4.6.10. Advice

- More examples of scripts are given in the /softs/pbs_users_guide/examples directory.
- When a job is submitted, a “session” is opened on the primary execution host, the shell startup script is executed and you are located in the user home directory (unless if has been modified in the user shell startup script). In order to be located in the directory where the qsub was executed, change the working directory to $PBS_O_WORKDIR.
- You can submit jobs from any nodes in the complex.
- Wall-time must correspond to what it is really needed and not be excessively overestimated. The requested wall-times affect the estimated start times of queued jobs and therefore the scheduling process.
- Avoid to load modules or complex environment settings in your .bashrc or .tcshrc shell initialization file or purge them at the beginning of your PBS script.

- Redirect the outputs of your program to a log file located somewhere in your directories. (on nodes, the PBSpro partition is limited to 7GB).

- Do not perform qstat with a period smaller than 5 seconds (as this may overload the PBS server in terms of number of simultaneous connections).

- Check the PBSpro resources and limits on nodes with pbsnodes command.

- Memory limits per chunk :

  - Westmere Fit nodes : 23000mb

  - Westmere Fat nodes : 47000mb

  - Westmere XFat nodes : minimum 47000mb – maximum 192000mb

  - Ivy Bridge nodes : 63000mb

- Memory needs have to be specified twice :

  - Per chunk in a select PBSpro directive (default 1 GB) : vmem

  - Per job for each process through job-wide resource (default 1 GB) : pvmem

- In queue large, as nodes are not shared by several jobs, the vmem and pvem values can be set to the maximum available, 63000mb.

- For shared nodes (queue main), the definition of chunks impacts the amount of resources allocated to the jobs, the placement on nodes and the waiting time. Two types of jobs can be consider :

  - Homogeneous jobs : all phases require the same amount of resources.

    ```
    #PBS -l select=32:vmem=1900mb
    #PBS -l pvmem=1900mb
    ```

    Or

    ```
    #PBS -l select=16:ncpus=2:mpiprocs=2:vmem=3800mb
    #PBS -l pvmem=1900mb
    ```

  - Both requests need exactly the same amount of resources. Selecting one of them depends on what your application does and what the other users/jobs do.

  - Heterogeneous job: As an example, consider the following analysis where the first step is sequential and requires 23000mb of memory and where the second step is parallel and requires homogeneously 1900mb per MPI process. This can be select following 2 ways:

    ```
    #PBS -l
    select=1:ncpus=1:mpiprocs=1:vmem=23000mb+31:npcus=1:mpiprocs=1:vmem=
    1900mb
    #PBS -l pvmem=23000mb
    ```

    Or

```
#PBS -l
select=1:ncpus=12:mpiprocs=12:vmem=23000mb+20:npcus=1:mpiprocs=1:vme
m=1900mb
#PBS -l pvmem=23000mb
```

In the first case the total amount of memory required is 81900mb. In the second one, the amount of is 61000mb.

4.7. Accounting

4.7.1. Accounting metric

The jobs accounting is based on the R_Wall_Time metric that accounts for the resources actually mobilized by the jobs. The metric definition is :

R_Wall_Time = Wall_Time x ncpus_equiv_pjob

where :

- Wall_Time is the execution time of the job (end_time – start_time),
- and ncpus_equiv_pjob is defined as follows :
  - if the job is run in a dedicated, reserved or exclusive queue or environment, complete nodes are associated to your job, and then
    ncpus_equiv_pjob = nodes_pjob x ncpus_pnode
  - else the job is run in a shared mode (sharing nodes with other jobs or projects) and then
    ncpus_equiv_pjob = max (ncpus_pjob , vmem_pjob / vmem_pcpu)

in which

- ncpus_pjob and vmem_pjob are respectively the total number of cores and total amount of memory requested by the job (respectively resource_list.ncpus and resource_list.vmem reported in a qstat -f on the job) and
- vmem_pcpu is the amount of memory available per core depending on the type of node requested, i.e.
  - 1917MB by default (Westmere fit nodes, -l model=westmere)
  - 3917MB if you requested Westmere fat nodes (-l model=westmere_fat)
  - 16000MB if you requested Westmere xfat nodes (-l model=westmere_xfat)
  - 2625MB if you requested Ivy Bridge nodes (-l model=ivybridge)

4.7.2. Reporting

- At the end of the job PBS output file, a summary of the resources requested and used is provided.

Example:

```
------------ INFO : JOB <job number>.frontal1 <job name> -------------

USER               : <user>
PROJECT            : <project>
QUEUE              : main_wes
```

```
JOB EXIT CODE      : 0

REQUESTED RESOURCES

  Number of Cores Requested per Job
  NCPUS_PJOB       : 94
  Total Virtual Memory Requested per Job
  VMEM_PJOB        : 535800mb
  Maximum Virtual Memory Requested per process
  PVMEM            : 23000mb
  Placement Requested
  PLACE            : free
  Execution Time Requested
  WALLTIME         : 23:59:00

USED RESOURCES

  Virtual Memory used on the master node
  VMEM             : 2489112kb

  CPU_Time USED    - To be corrected
  CPU_Time         : 00:01:20
  Wall_Time USED   - Execution Time
  Wall_Time        : 00:01:25

  N_Wall_Time USED - NCPUS_PJOB x Wall_Time
  N_Wall_Time      : 2:13:10

  Number of Mobilized Resources per Job :
      NCPUS_EQUIV_PJOB = max ( NCPUS_PJOB , VMEM_PJOB / VMEM_PCPU )
      VMEM_PCPU is the amount of memory available per core depending on the type
of node requested
  NCPUS_EQUIV_PJOB : 280

  R_Wall_Time USED - NCPUS_EQUIV_PJOB x Wall_Time
  R_Wall_Time      : 6:36:40


For more details, use qstat -f -x <job number>.frontal1

------------ END INFO : JOB <job number>.frontal1 <job name> ---------

[EPILOGUE] Ending epilogue on node node0007 at date Fri Oct  9 23:31:00 CEST
2015
```

- Reports are sent weekly and monthly to project managers. Persistent and scratch storage usages and some jobs statistics are provided with the R_Wall_time credit used during the period by queue and/or globally and the remaining credit.

  Some ratios are also provided:

  - **η =** Total CPU_Time / (Total Wall_Time * NCPUS_PJOB)
  - **α =** Total CPU_Time / (Total Wall_Time * NCPUS_EQUIV_PJOB)

Example:

```
================================================================================
Project <project name> created Friday Oct 31 2014
================================================================================

 SCRATCH storage quota                     : 400 GiB
 SCRATCH storage used                      : 196 GiB
 Persistent storage quota                  : 200 GiB
 Persistent storage used                   : 48 GiB


 --------------------------------------------------------------------------------
 All Standard Queues Usage
 --------------------------------------------------------------------------------

 R_Wall_Time Credit used this month       :        10.6 hours

 Job Usage Detail (times in hours)
              # of       Total     Total       Total             Total         Average
 Username     jobs    CPU_Time Wall_Time N_Wall_Time    η  R_Wall_Time    α  Wait_Time
 ----------- ----- ---------- --------- ----------- ----- ---------- ----- ---------
 TOTAL           3         8.7       1.0        10.6 82.1%        10.6 82.1%       0.0

 user1           3         8.7       1.0        10.6 82.1%        10.6 82.1%       0.0

 Job Set Summary (times in hours)
                          Percentil   Percentil   Percentil
                Minimum         P_50        P_75        P_95     Maximum
              ----------  ----------  ----------  ----------  ----------
 Ncpus                1           2           2           2          32
 CPU_time           0.0         0.0         0.0         0.0         8.7
 Wall_time          0.0         0.3         0.3         0.3         0.7
 Wait_time          0.0         0.0         0.0         0.0         0.0


 --------------------------------------------------------------------------------

 R_Wall_Time Credit allocated             : 31000 hours
 R_Wall_Time Credit Valid until           : Thursday Apr 30 2016
 R_Wall_Time Credit used previous report  :     20128.2 hours
 R_Wall_Time Credit used this month       :        10.6 hours
 R_Wall_Time Credit used current report   :     20138.8 hours
 Remaining R_Wall_Time Credit             : 10861 hours
 Percentage R_Wall_Time Credit left       : 35 %
```

5. Any question ?

Do not hesitate to contact us. Please send email to it@cenaero.be