Wikiprint Book

Title: Marc2 Trac

Subject:

Version:

Date:

Table of Contents

Marc2 Users Guide

Table of Contents

Introduction

Getting started

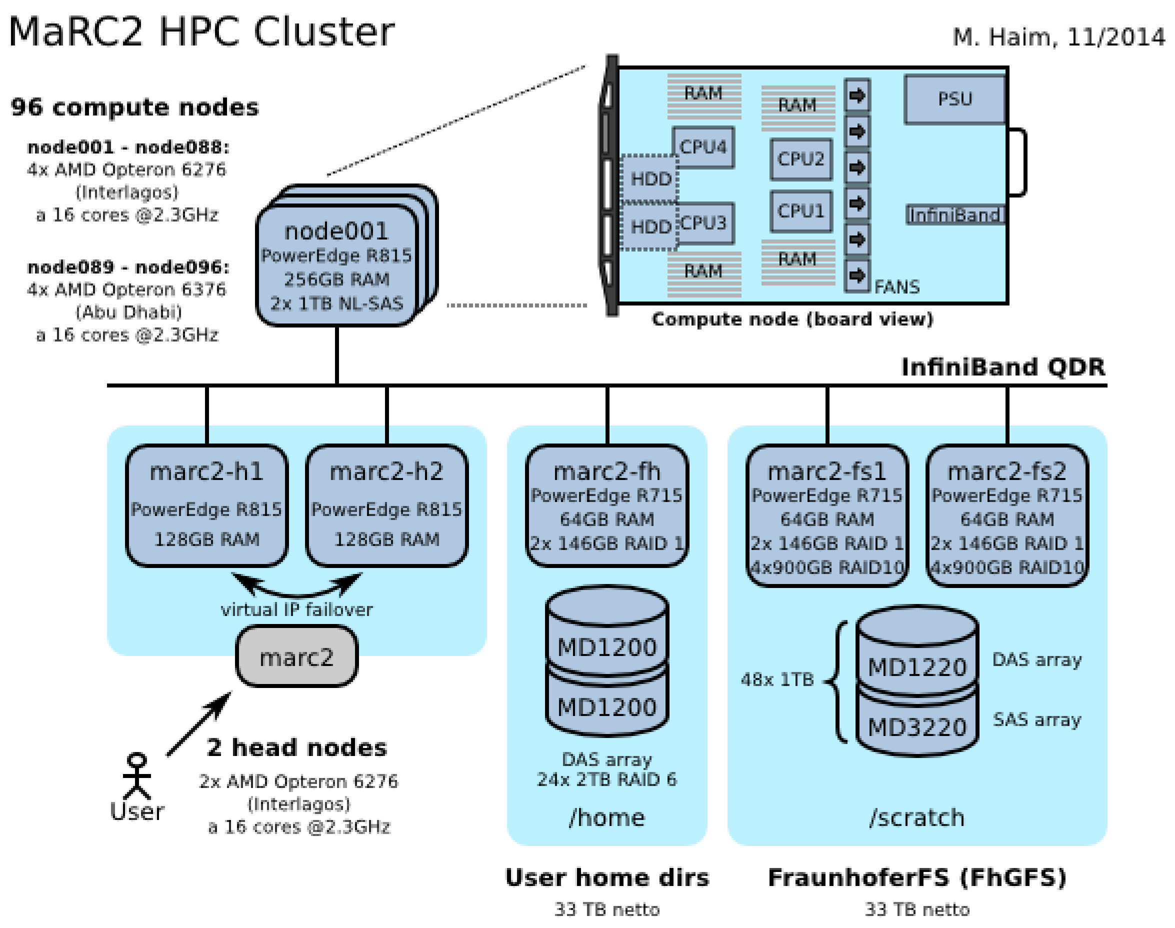
In order to get access to the MaRC2 HPC Cluster, please email the MaRC2 team at marc[at]hrz.uni-marburg.de for further assistance.

There are various ways of getting started:

- See the overview and workshop materials below
- Visit a workshop (as announced on the linux-cluster mailing list)
- Read the tutorial and other chapters from this MaRC2 User Guide (see table of contents)
- Just ask the MaRC2 team by email: marc[at]hrz.uni-marburg.de
- See the language-specific example files, located on MaRC2's file system at /home/examples

The MaRC2 HPC Cluster - overview

The following picture shows a schematic overview of the MaRC2 HPC Cluster:



(Also see the German overview at http://www.uni-marburg.de/hrz/infrastruktur/zserv/cluster)

After account activation, you may login to the head nodes via SSH and use them for medium-performance tasks, e.g. to edit your files or to compile your programs. Several pre-installed compilers and software versions can be selected through Environment Modules.

For high-performance execution on the compute nodes, a job must be submitted to the Batch system (Sun Grid Engine). The batch system keeps track of the cluster's ressources and makes sure that everyone will get a chance to have his/her jobs executed in good time. You may also set the Parallel environment of your job (e.g. multiple cores on one node, or distributed over multiple nodes).

Memory (RAM) among the CPUs of a compute node is shared, but may have different access times (so-called NUMA architecture). Each compute node has four CPU sockets, however each physical CPU is composed of two logical CPUs internally (with eight cores each). Furthermore, each two cores of a logical CPU share a single FPU.

Jobs which are distributed over multiple compute nodes have no shared memory and thus must be programmed to communicate via network, e.g. by using OpenMPI or the ParaStation MPI.

There is a Tutorial as well as Frequently asked questions available within this wiki.

MaRC2 workshops and training materials for download

Introductory workshops are planned to be held yearly. The upcoming and past workshops are summarized here:

HiPerCH 2 workshop (High-Performance Computing in Hessen) (22-Sep-2014 thru 24-Sep-2014):
3-day workshop at TU Darmstadt
More information soon at HKHLR, see http://www.hpc-hessen.de

Iterative linear solvers and parallelization (in German language) (24-Mar-2014 thru 28-Mar-2014):
1-week compact course, provided by High Performance Computing Center Stuttgart (HLRS)
http://www.hlrs.de/organization/sos/par/services/training/2014/ITER-S

Parallel Programming Concepts (starting 03-Feb-2014):
Free 6-week online course, hosted by Hasso Plattner Institute (HPI) Potsdam
https://openhpi.de/course/parprog2014

MaRC2 Matlab Meeting (06-Nov-2013):
http://www.staff.uni-marburg.de/~haimm/marc2-matlab-20131106.txt

MaRC2 R Meeting (06-Nov-2013):
http://www.staff.uni-marburg.de/~haimm/marc2-r-20131106.txt

MaRC2 Introductory Workshop (19-Jun-2013):
High Performance Computing in general: http://www.staff.uni-marburg.de/~gebhardt/marc2-slides.tar.bz2
Using the MaRC2 cluster: http://www.staff.uni-marburg.de/~haimm/marc2practice.pdf

MaRC2 User Meeting (23-May-2013):
(no materials)

MaRC2 Introductory Workshop (06-Mar-2012):
(no materials)

Obtaining a PDF version of this document

A PDF version of the User's Guide may be found here.

Table of Contents

Batch system

Purpose of a batch system

The batch system schedules the workload on the cluster in such a way that all users can benefit according to the site policy. It should prevent users from monopolising the resources. Typically this is done by enqueuing several jobs into a queue. The batch systems then assigns priorities to the queued jobs. These priorities can be calculated from the time a job has been waiting, resources already used during the last few days, membership in a special project, requested resources, … (see Job priorities). The queued jobs are re-ordered according to priority and are executed.

Sun Grid Engine

The installed version of Sun Grid Engine is 6.2u5.

The documentation for the Sun Grid Engine can be found at

- Oracle's web site (offline by now, use www.archive.org instead)
- Open Grid Scheduler website (open-source successor of SGE 6.2u5)
- or within the man pages (see `man sge_intro`)

If you find the time, you're encouraged to read through it to learn about the concepts.

A first example

Jobs are submitted to the system via scripts. Resources are requested by specifying certain directives.

To start, here's the classical "hello world" example:

```
#!/bin/bash

# hello_world.sh
# a minimal example

#$ -S /bin/bash
#$ -e /home/partec/sge
#$ -o /home/partec/sge
#$ -l h_rt=360

echo "Hello world from $(hostname)"

exit 0
```

This simple script tells the Sun Grid Engine what shell to run in, and where to write the stderr and stdout outputs. It does it via *directives*. These are marked by the comment sign followed by the dollar sign (#$). Here,

- -S specifies the shell to use (/bin/bash in this case),
- -e specifies the location where your stderr goes,
- -o specifies the location where your stdout goes.
- -l h_rt specifies the runtime (wallclock) limit in seconds (this is mandatory, your job won't start otherwise).

Finally, the script echoes the name of the machine it is executed on and exits. Assuming you have placed this file under the name "hello.sh" in your storage, you can place it into the queue for execution (your prompt may vary):

```
lpartec@marc2-h1:~/sge>qsub ./hello.sh
Your job 115 ("hello.sh") has been submitted
```

Please note that it's not necessary to grant execution rights to the script. You can check the state of your job using the qstat command:

```
lpartec@marc2-h1:~/sge> qstat
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
   115 0.50500 hello.sh   lpartec      Eqw   03/01/2012 15:21:06                                    1
```

Our job is in an error state (E) and queued waiting (qw). To find out why, use the -explain switch of qstat. Look for the "error reason":

```
lpartec@marc2-h1:~/sge> qstat -j 115 -explain E
==============================================================
job_number:                 115
exec_file:                  job_scripts/115
submission_time:            Thu Mar  1 15:21:06 2012
owner:                      lpartec
uid:                        1000
group:                      partec
gid:                        1000
...
stderr_path_list:           NONE:NONE:/home/partec/sge
mail_list:                  lpartec@marc2-h1
notify:                     FALSE
job_name:                   hello.sh
stdout_path_list:           NONE:NONE:/home/partec/sge
jobshare:                   0
shell_list:                 NONE:/bin/bash
env_list:
script_file:                ./hello.sh
error reason    1:          03/01/2012 15:21:16 [1000:24143]: error: can't open output file "/home/partec/sge": No such fi
...
```

The problem is that the file /home/partec/sge doesn't exist. There's not much we can do about this (the correct home dir would be /localhome/lpartec/sge in this case), so the only option is to delete the job:

```
lpartec@marc2-h1:~/sge> qdel 115
lpartec has deleted job 115
```

To see if we get it right, we can override the settings in the script from the command line:

```
lpartec@marc2-h1:~/sge> qsub -e $(pwd) -o $(pwd) -l h_rt=360 ./hello.sh
Your job 116 ("hello.sh") has been submitted
```

where the current working dir has been evaluated. Our job is running now:

```
lpartec@marc2-h1:~/sge> qstat
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
   116 0.50500 hello.sh   lpartec      r     03/01/2012 15:28:16 all.q@node082.marc2                1
```

It has produced some output files:

```
lpartec@marc2-h1:~/sge> ls -l hello.sh*
-rw-r--r-- 1 lpartec partec 180 Mar  1 15:27 hello.sh
-rw-r--r-- 1 lpartec partec   0 Mar  1 15:28 hello.sh.e116
-rw-r--r-- 1 lpartec partec  19 Mar  1 15:28 hello.sh.o116
```

Indeed the output file (hello.sh.o116) contains:

```
lpartec@marc2-h1:~/sge> cat hello.sh.o116
hello from node082
```

With this knowledge, you can now fix your script to use the proper locations for the output. Be aware though that shell script evaluation like we did on the command line won't work. You have to hard-wire the exact location:

Bad:

```
#$ -e $(pwd)
#$ -o $(pwd)
```

Good:

```
#$ -e /your/output/directory
#$ -o /your/output/directory
```

Further reading: man sge_intro

The important commands

The main commands you will need deal with the creation, diagnose and control of batch jobs. They all come with well written man pages. Please consult the man pages first if you have a question.

qstat

This command tells you about the state of the cluster: The running jobs, state of nodes, … Some examples:

- Show your jobs:

```
lpartec@marc2-h1:~> qstat
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
```

```
   107 0.55500 hello.sh   lpartec      r     02/29/2012 14:34:04 all.q@node046.marc2              128
```

Use the option -u "*" to see jobs from all users.

- Select full view:

```
lpartec@marc2-h1:~> qstat -f
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
...
---------------------------------------------------------------------------------
all.q@node011.marc2            BIP   0/64/64        8.00     lx24-amd64
   107 0.55500 hello.sh   lpartec      r     02/29/2012 14:34:04    64
---------------------------------------------------------------------------------
...
```

- Diagnose job priority: qstat -pri or qstat -urg (see `man sge_priority` or Job priorities)
- Show details about your job: qstat -f -j <jobid>
- Diagnose your job: qstat -j <jobid> -explain a|A|c|E
  Please note that, due to a memory leak bug in SGE 6.2u5, gathering of scheduler information had to be disabled, thus it must be queried explicitly: qalter -w v <jobid>
- Diagnose a node: qstat -q <queue> -explain a|A|c|E

qsub

This command is used to submit the actual jobs. You can use all the options specified here also in your script by using the #$ directive comment.

- qsub -l h_rt= <runtime limit> <your job script.sh> <your job script's arguments>: Simple job submission via script
- qsub -l h_rt= <runtime limit> -b y <your binary> <your binary's arguments>: To submit a binary directly
- qsub -l h_rt= <runtime limit> -t <n1>-<n2> …: To submit a job array. n1 has to be bigger than 0.
- qsub -l h_rt= <runtime limit> -cwd <directory> …: To specify the workdir of a job
- qsub -l h_rt= <runtime limit> -q <queue-list> …: To specify a list of queues

qmod

This command can control/administer running jobs.

- qmod -sj <job id>: puts job into suspend ($^{\text{Z/kill -SIGSTOP)}}$
- qmod -usj <job id>: releases job from suspend (kill -SIGCONT)
- qmod -cj <jobid>: clears the error state of a job

qalter

This command can modify pending or running jobs. This is useful if you have to re-shuffle the order of your jobs. Most of the switches available with qsub are available here, so you can fix parameters you forgot or specified wrongly at submission time. Note that you can ask only for less, not more.

- qalter -a <time> <jobid>: Re-define the time a job is eligible for execution.
- qalter -cwd </some/path> <jobid>: Re-define the working dir. No effect if job is already in state "r"
- qalter -q <new queue> <jobid>: Move the job from the previously specified queue to the new queue. No effect if job is already in state "r"

There is also a workaround for getting scheduler information for your job:

- qalter -w v <jobid>: Show scheduler information

qdel

Terminates jobs.

- qdel <job id>: Deletes the job
- qdel -u <user>: Deletes jobs of user <user>

qconf

This command allows you to query properties of the batch system.

- qconf -sql: List all queues
- qconf -sel: List all execution nodes
- qconf -se <node>: Show properties of execution node <node>
- qconf -sq <queue>: Show properties of queue <queue>
- qconf -spl: List all parallel environments (see below)
- qconf -sp <pe>: Show properties of parallel environment <pe>

Batch jobs/Resources

You can find several example scripts for the Sun Grid Engine in $SGE_ROOT/examples/jobs. The topics treated cover probably most of what you will need:

- Job Arrays: Multiple jobs running under one JOB
- Job Dependencies: A chain of jobs running in a certain order.
- …

As mentioned before, the purpose of a batch system is to make resources available to user in a somewhat just way and at the same time maximise the load of the machine. Among these resources are compute time, memory, licenses, … They are specified by the -l switch. Resources come in two flaovours:

- hard: job will not start until resource is available
- soft: Sun Grid Engine will try to allocate resource, but might start without

The complete list of resources is available via *man 5 complex* and *man 5 queue_conf*.

Some of the more important resources are:

- [h|s]_rt: The real ("wallclock") time a job may run. When reaching the hard limit, the job is killed via a SIGKILL. If the soft limit is reached, your job will receive a SIGUSR1 signal. After the time specified in the queue parameter notify, it is then killed with SIGKILL.
- [h|s]_cpu: The CPU time your job (all processes) have consumed. Otherwise, same behaviour as the _rt parameter. For parallel jobs, the limit is applied to each slot.
- [h|s]_fsize: The filesize, set via *ulimit -f*
- [h|s]_data: The data seg size, set via *ulimit -d*
- [h|s]_stack: The stack size, set via *ulimit -s*
- [h|s]_core: The core file size, set via *ulimit -c*
- [h|s]_vmem: The virtual memory size, set via *ulimit -v*
- h_rss: resident set size limit. Job will be killed if exceeded.
- mem_free: Node must have at least this free memory.

See also *man ulimit*

Interactive jobs

Interactive jobs can be launched in different ways:

- qlogin: Interactive login session
- qrsh: Interactive rsh session
- qsh: Interactive X-windows session

What's the difference?

Task arrays

Task arrays allow you to submit a group of jobs under one job id. This is useful e.g. if you have a set of data which is split into chunks ("farming"). You can identify your data set by evaluation of the environment variable SGE_TASK_ID.

Parallel environments

By setting a parallel environment, you may define the number of slots (i.e. CPU cores) needed by your job, as well as the way the slots are allocated among the compute nodes (e.g. all slots on one node).

Select the parallel environment with the -pe switch, for example:

- ~~parastation: Contiguous allocation of slots (slots may be allocated on different nodes, though)~~ (this PE has been disabled)
- parastation_rr: Round robin allocation of slots
- ~~parastation_sl04 / parastation_sl08 / parastation_sl12 / parastation_sl16 / parastation_sl32 / parastation_sl64: Single-node allocation with 4 / 8 / 12 / 16 / 32 / 64 slots~~ (use parastation_smp* instead)
- smp / smp_long / parastation_smp / parastation_smp_long: Single-node allocation of slots (symmetric multiprocessing, all slots on one node)
  Hint: Use a * wildcard to allow any of those smp environments, e.g. "-pe parastation_smp* 4" to allocate 4 slots on one node, either on parastation_smp or parastation_smp_long (depending on "-l h_rt")
- Use "qconf -spl" to get a list of all available parallel environments, "qconf -sp <pe>" will show configuration details of parallel environment <pe>.

Parallel environments also provide the environment variable NSLOTS, which can be used for the -np option of the mpiexec command:

```
#!/bin/bash

#$ -S /bin/bash
#$ -e /localhome/lpartec/src/mpi
#$ -o /localhome/lpartec/src/mpi
#$ -cwd /localhome/lpartec/src/mpi
#$ -pe parastation_rr 128

# load the proper module
. /etc/profile.d/modules.sh
module load parastation/mpi2-gcc-5.0.27-1

# run the mpiexec
mpiexec -np ${NSLOTS} ./hello
```

Environment variables

Your job script sets the following environment variables (among others)

SGE_O_WORKDIR	The directory from which you submitted, or, if applicable, specified with the -cwd switch
TMPDIR	The location of a scratch dir provided to you. Disappears after job finished
SGE_TASK_ID	The id of a task in your task array.

Queues

Jobs scheduled for execution are assigned to one of the following queues, depending on the h_rt and parallel environment (and maybe queue list) you chose:

```
serial_test   (max. h_rt=3600,   i.e.  1 hour)
serial        (max. h_rt=259200, i.e.  3 days)
serial_long   (max. h_rt=864000, i.e. 10 days)
parallel_test (max. h_rt=3600,   i.e.  1 hour)
parallel      (max. h_rt=259200, i.e.  3 days)
parallel_long (max. h_rt=864000, i.e. 10 days)
```

Old and new MaRC2 nodes:

The MaRC2 Cluster has been expanded in Sep / Oct 2014 by 8 additional compute nodes (node089-096). Therefor, we have defined host groups for the different hardware generations:

```
@nodes_g1 (= node001 ... 088)
@nodes_g2 (= node089 ... 096)
```

In order to execute e.g. a serial job only on the second generation nodes, you may submit your job as follows:

```
qsub -q serial@@nodes_g2 -l h_rt=100 hello-world
```

Software update (Status 23-Oct-2014)

There will be a software update on the compute nodes in the near future. We plan to update the compute nodes step by step without cluster downtime. Again, we hope that the new software generation acts as a drop-in-replacement.

Similarly there will be two host groups:

```
@nodes_s1 (old software, now node001 ... 096)
@nodes_s2 (new software, currently empty)
```

Nodes will move from @nodes_s1 to @nodes_s2 one after the other. We will let you know when the upgrade precedure begins. We will start with a few test nodes.

When the software update has finished, the hostgroups @nodes_s1 and @nodes_s2 will be discarded (@nodes_g1 and @nodes_g2 are going to stay).

Job priorities

Submitted jobs are ordered and executed (i.e. assigned to queues) by job priority. The job priority usually depends on several distinct values (see `man sge_priority`).

Current status (05-Sep-2014):

On MaRC2, the job priority only depends on allocated slots and waiting time (both the higher the better).

In detail, the priority is calculated as follows:

```
priority = 1*normalized_posix_priority + 0.1*normalized_urgency + 0.01*normalized_tickets
urgency = resource_requirement_contribution + waiting_time_contribution + deadline_contribution
resource_requirement_contribution = 1000*allocated_slots
waiting_time_contribution = 0.740740*seconds_since_submit
```

whereas normalized_posix_priority, normalized_tickets and deadline_contribution are usually constant.

Hint: Call `qstat -pri` or `qstat -urg` to see your jobs' distinct priority or urgency values.
Use `qconf -ssconf` and `qconf -sc` to show scheduler and complex configuration.

Table of Contents

Using the Module Environment

The compilation and execution environment installed on Marc2 is defined using *modules* command. For a list of available modules, use the *avail* sub-command:

```
$ module avail

---------------------------------- /usr/share/Modules/modulefiles ----------------------------------
dot         module-cvs  module-info modules     null        use.own

------------------------------------- /usr/share/ModulesLocal --------------------------------------
acml/gfortran-5.1.0(default)           parastation/mpi2-gcc-mt-5.0.27-1
acml/ifort-5.1.0                       parastation/mpi2-intel-5.0.27-1
acml/pgi-5.1.0                         parastation/mpi2-intel-mt-5.0.27-1
gcc/4.4.6                              parastation/mpi2-pgi-5.0.27-1
gcc/4.6.2(default)                     parastation/mpi2-pgi-mt-5.0.27-1
parastation/mpi2-gcc-5.0.27-1(default) pgi/12.2(default)
```

To show all currently loaded modules, use *list*:

```
$ module list
Currently Loaded Modulefiles:
 1) acml/gfortran-5.1.0             3) parastation/mpi2-gcc-5.0.27-1
 2) gcc/4.6.2
```

To load a specific module, use *load*:

```
$ module load parastation/mpi2-gcc-5.0.27-1
```

For more details, see *man module*. Other important commands:

1. **`module avail`**: shows a list of available modules
2. **`module list`**: lists the currently loaded modules
3. **`module unload <module>`**: removes a module
4. **`module purge`**: removes all loaded modules

Note: at interactive login a default set of modulefiles is loaded. In contrast, a batch job is started in a clean minimal environment where even the module command is not available. In a job script you can load the shell functions that implement the module command by either the line

```
. /etc/profile.d/modules.sh
```

or, alternatively, you can declare the executing shell to be a *login shell* :

```
#!/bin/bash -l
```

Tutorial

This section contains some simple examples for you to familiarise yourself with the cluster. The relevant files can be found under */direct/Software/Training*.

Hello World

In this simple example you will learn how to write a script which can be submitted into the batch system. You can then submit it and get some output back.

Copy this script into your account:

```
#!/bin/bash

# hello.sh
# a simple script
# PN Thu Feb 23 15:58:23 CET 2012

#$ -S /bin/bash
#$ -e <put the directory the script resides in here, e.g. /home/<your user name>/myfirstsge>
#$ -o <like -e>

echo "hello from $(hostname)"
```

Name this script *hello.sh* (or whatever name you like to give it).

Submit the script:

```
qsub ./hello.sh
```

Check it's status:

```
qstat
```

You might see something like

```
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
  487 0.00000  hello.sh   lpartec      qw     03/12/2012 09:15:24                                   1
```

Meaning that your job has been assigned the ID 487, has no running priority yet (it's 0), the name of your job is `hello.sh`, it's running under the user id of `lpartec`, it's waiting (`qw`), was submitted at that time, and uses 1 slot. After some time it will get a higher priority (0.50500 or so) and will change into running (`r`) and a queue instance will be assigned. At some point it will have disappeared from qstat (if not, check the FAQ section) and you will find two files called *hello.e.<jobid>* and *hello.o.<jobid>*. One is the std error, the other the std output of your job. Inside, it says something like

```
hello from nodeXXX
```

You could add the command **env** to your script and find out how your environment differs from interactive use.

Now, modify the "echo" line of your script to look like this:

```
echo "hello from task ${SGE_TASK_ID} on $(hostname)"
```

To run your job again as a task array, use the **`-t`** switch:

```
qsub -t 10-19 ./hello.sh
```

This feature is useful when working on a set of data, you could e.g. index your data files using the *$SGE_TASK_ID* variable.

ACML

The AMD Core Math Library is a compilation of mathematical functions specially tuned for the AMD processor. You can find more info on the ACML home page. A PDF document containing compilation and linking examples is at */opt/acml<version>/Doc/acml.pdf*.

Example programs are found in the */opt/acml<version>/<compiler>/examples* directory.

Under */direct/Software/Training/acml*, you will find the C++ source code to do an LU-decomposition of a matrix (*dgetrf.cc*). You can use the supplied Makefile to generate a binary and to check what flags are necessary.

The program will read in a problem size from the command line, print out the ACML version information and calculate the GFlops from the number of operations and the time spent in the necessary routine:

```
lpartec@marc2-h1:~/training/acml> ./dgetrf 200
ACML (AMD Core Math Library) version 5.1.0  (Mon Dec 12 01:58:29 CST 2011)
Copyright AMD,NAG 2011
Build system: Linux 2.6.32.28-fam15h-default x86_64 denarius
Built using Fortran compiler: GNU Fortran (GCC) 4.6.0
  with flags:  -ffixed-line-length-132 -Wall -W -Wno-unused -Wno-uninitialized -fPIC -fno-second-underscore -fimplicit-non
and C compiler: gcc (GCC) 4.6.0
  with flags:  -Wall -W -Wno-unused-parameter -Wstrict-prototypes -Wwrite-strings -D_GNU_SOURCE -D_ISOC99_SOURCE -fPIC  -m
Problem size: 200, Number of runs: 100, time spent in dgetrf: 0.35 s, Number of operations: 5.36e+08, GFlop/s: 1.53143
```

The supplied *dgetrf.sh* shell script contains two directives:

1. **`#$ -cwd`**: It tells SGE to change to the directory the job was submitted from. Thus, you can specify your binary as **`./dgetrf`** instead of it's absolute path. By the way, the ~ won't expand to your home directory.
2. **`#$ -j y`**: Join the standard input and standard output. Without it, the acml version information will go to the .o file, the timing info will go to the .e file.

Don't forget to supply the problem size when submitting:

```
qsub ./dgetrf.sh <problem size>
```

MPI

MPI (Message Passing Interface) is a way of parallel programming. ParaStation MPI is an effective implementation.

A minimal program is found at */direct/Software/Training/mpi*. To compile the source code, you can use the mpi compiler wrapper:

```
mpicc -o hello hello.c
```

The program prints out its MPI rank and the host it is running on.

To protect the frontend from overload, parallel programs won't run on it, so you have to submit it into the cluster via SGE. There's a script file, *hello.sh*. When submitted, you must specify the parallel environment it should run in and the number of ranks, e.g.

```
qsub -pe parastation_rr 16 ./hello.sh
```

SGE will reserve the required number of slots and provides the environment variable *$NSLOTS* which can be used in the *-np* argument:

```
...
mpiexec -np ${NSLOTS} ./hello
...
```

ParaStation? will ignore any —*machinefile* etc. switches in batch operations.

In case you see

```
ips_proto_connect: Couldn't connect to nodeXXX
```

in your output, disable the PSM layer by setting

```
export PSP_PSM=0
```

in your script file.

High Performance Linpack

The HPL is a benchmark used to rank SuperComputers? in the top 500 list. It solves a system of linear equations in a distributed fashion.

The source code is available at netlib.org. It is also available at */direct/Software/Training/hpl/hpl-2.0.tar.gz*. To get started, unpack it:

```
gzip -dc hpl-2.0.tar.gz | tar xvf -
```

In the resulting directory, you can find some example Makefiles in the *setup* directory. Copy the *Make.Linux_ATHLON_FBLAS* into the hpl top directory and rename it to something specifying the MPI and performance lib used, e.g. Make.ps_acml. In there, you need to change the following settings:

1. **`ARCH`**: The name of your "architecture" (ps_acml) in our example
2. **`TOPdir`**: The location of your hpl top directory
3. **`MPdir`**: The top directory of your MPI implementation, e.g. */opt/parastation/mpi2*
4. ***`MPlib`**: The libraries needed for MPI. **`mpif77 -show`** tells you the details*
5. ***`LAdir`**: The top directory of your performance lib, e.g.* /opt/acml5.1.0/gfortran64
6. ***`LAlib`**: The librararies for your performance lib, e.g.* $(LAdir)/lib/libacml.a
7. ***`CC`**: Your C compiler, ususally `gcc`*
8. ***`LINKER`**: Your linker, `gfortran`*

Once your Make.<ARCH> *is set up, type*

```
make arch=<ARCH>
```

and you will get a binary *xhpl* in the directory *./bin/<arch>* plus a default steering file using 4 processes. Write a wrapper script:

```
#!/bin/bash

...
#$ -pe parastation_rr 4
...

mpiexec -np ${NSLOTS} ./xhpl
```

Once you got it running, you can tune your HPL.dat file to measure the parallel performance of your system.

Table of Contents

ParaStation MPI

To use ParaStation MPI, the appropriate module must be loaded:

- parastation/mpi2-gcc-5.0.27-1: uses GCC compiler and libraries,
- parastation/mpi2-intel-5.0.27-1: uses Intel compiler and libraries
- parastation/mpi2-pgi-5.0.27-1: uses PGI compiler and libraries

See module environment for details. To compile applications, use *mpicc*, *mpif77* or *mpif90* available by using $PATH. The required GCC compiler may be specified using the appropriate module.

To run an application use *mpiexec* also available by using $PATH. Or use the absolute path */opt/parastation/bin/mpiexec*. Information on nodes to be used is typically provided by the batch queuing system and automatically selected by the ParaStation runtime environment.

See batch queuing system for an example how to use *mpiexec*. See mpiexec man page and ParaStation User's Guide for more information.

Environment Variables for use with QLogic IB

To enable the full performance of the installed QLogic HCAs, ParaStation uses the PSM layer provided by the QLogic OFED stack. In order to do so, the shared library libpsm_infinipath (and libinfinipath) are dynamically linked to the ParaStation MPI communication layer at run time.

Some ParaStation environment variables controlling this behavior, see ps_environment for more details:

- *PSP_PSM*: this variable defines the priority of using the PSM library. Setting it to 2 or higher elevates it over the default priority of 1 and therefore favors this communication channel against all others. Setting it to 1 or unsetting it at all, gives the default behavior: use PSM for inter-node communication and shared memory for intra-node communication. Setting it to 0 prevents using PSM.

By default, PSP_PSM is set to 2 and exported to all processes to prevent some multi-node timing issues appearing with multi-node jobs using PSM and shared memory. Jobs running within a single node may set the variable to 0 to re-enable shared memory communication.

Some variables for usage with PSM:

- *PSM_SHAREDCONTEXTS_MAX*: see documentation. From Andrew Russel (Intel):

  "PSM_SHAREDCONTEXTS_MAX can be helpful. It defines how many hardware contexts the job will consume. You have 16 hardware contexts. The free contexts can be shared 1, 2, 3 or 4 ways. For PSM_SHAREDCONTEXTS_MAX to be useful, you have to set it on the PREVIOUS job. Eg: if I start 16 processes on a 64-core node, that job will use all 16 contexts, so you can't start any more jobs on that node. If I start 16 processes on a 64-core node with PSM_SHAREDCONTEXTS_MAX=8, that job will use 8 contexts, leaving 8 free. You could start another 32 processes (8 contexts shared 4 ways) on that node."

- *PSM_RANKS_PER_CONTEXT*: (undocumented) From Andrew Russel (Intel):

  "If you always set "$PSM_RANKS_PER_CONTEXT=4", then it will always do 4-way sharing, and so you should not grab all the contexts when partially filling a node. (This removes the need to do the logic I went through to calculate the correct value of PSM_SHAREDCONTEXTS_MAX, further down in this thread). Note that this only works well when your ppn is divisible by 4."

- *MPI_LOCALRANKID*: (undocumented) required for PSM_SHAREDCONTEXTS_MAX or PSM_RANKS_PER_CONTEXT to work. Automatically set by mpiexec.
- *MPI_LOCALNRANKS*: (undocumented) required for PSM_SHAREDCONTEXTS_MAX or PSM_RANKS_PER_CONTEXT to work. Automatically set by mpiexec.
- *PSM_SHAREDCONTEXTS*: see documentation. Enable/disable context sharing. Enabled by default.
- *PSM_DEVICES*: see documentation. Communication paths used by PSM.

Some of these PSM-specific variables are described in UserGuide_IB_OFED_HostSoftware153_IB0054606-01B.pdf

Note: variables other than PSP_* or PSI_* have to be exported to the processes either using *mpiexec —env …* or adding there names to PSI_EXPORTS, eg.

```
# export PSM_RANKS_PER_CONTEXT=4
# export PSI_EXPORTS="$PSI_EXPORTS,PSM_RANKS_PER_CONTEXT"
# mpiexec ...
```

See ps_environment and mpiexec for more details.

Table of Contents

Frequently asked questions

How large can my task array be?

Your task array can be of reasonably large size. However, you can run only 88x64=5632 tasks simultaneously.

My job is listed in state "Eqw". What should I do?

Use the *-explain* switch:

```
qstat -j <your job id> -explain E
```

qstat says that gathering scheduler information is disabled. How do I get the information I need?

Please use the following command to get scheduler information for your job:

```
qalter -w v <your job id>
```

Jobs complain about missing libmpich

When I use the compiler wrapper, *ldd* of the produced binary says everything is fine. On the node within batch it complains about missing libraries, e.g. *libmpich.so*.

Resolution: use the appropriate module(s), see example job script.

Java quits due to lack of memory

The Java Virtual Machine usually needs more RAM than configured with the -Xmx option, thus you need to provide a higher h_vmem value for qsub.

Using a simple "hello world" program and a short wall clock time (h_rt), you may experimentally figure out the amount of memory needed for a given -Xmx value. Remember to also set -Xms to the same value as -Xmx, as otherwise the Java VM will not consume all the asserted memory at startup. If this job exits without error, your h_vmem value is big enough.

Here is an example startup script (to be submitted by qsub):

```
#$ -S /bin/bash
#$ -e /path/to/stderr-output-file
#$ -o /path/to/stdout-output-file
#$ -l h_rt=10
#$ -l h_vmem=9G
java -Xms5120m -Xmx5120m -cp /path/to/HelloWorld.jar HelloWorld
```

I have accidentally modified/deleted some files. Is there a backup?

The /home filesystem is backed up each night by ITSM. In order to have your files restored, please write an email to **marc[at]hrz.uni-marburg.de** and tell us the directory and date of the files to be restored.

What is the NUMA architecture on MaRC2?

For an introduction to NUMA, see Wikipedia article: http://en.wikipedia.org/wiki/Non-uniform_memory_access

The following picture describes the NUMA architecture on a MaRC2 compute node (click to enlarge):