Using the new S-CAPAD cluster

G. Moguilny – December 17, 2014

Contents

1 Available resources

In July 2013, the S-CAPAD (or SCP) has been equipped with a new DELL cluster consisting of

- 96 CPU intensive nodes (`cnode001` to `cnode096`),
- 16 data intensive nodes with SSD disks (`dnode01` to `dnode16`),
- 4 GPU nodes (`gpu01` to `gpu04`),
- 1 SMP node with 64 cores and 128 GB RAM (`smp01`),
- 1 GPFS parallel filesystem with 524 effective TB,

all connected by an Infiniband Fat-tree network QDR 100 % non-blocking.

Data from the Cohersis Netapp filer is available from the new cluster.

All machines run CentOS 6.4 and provide, amongst others, Intel optimized scientific compilers and libraries.

The latest version of this documentation can be found on the cluster:
`/cm/shared/docsExtra/SCAPADuserENG.pdf`
(in french: `SCAPADuserFR.pdf`) and a more detailed description of the resources is available at
`http://webpublix.ipgp.fr/rech/scp/Ressources/ClusterVision.php`.

2 Access

After request and obtaining an account on the S-CAPAD web site, it is possible to access the cluster via three login servers on which the load is automatically distributed. All are accessible by the generic name `malbec`. From these 3 machines, applications can be edited, compiled and submitted to the cluster. To access the cluster from IPGP type:

`ssh [-XC]` *user*`@malbec`

(`X` to enable X11 forwarding, and `C` to compress the data transferred).

To access the cluster from outside IPGP, users need an account on the IPGP firewall obtained from the IPGP IT team (entry point: `malbec01`).

3 The new DELL cluster for the impatient

Once logged into `malbec`, you can do the folowwing to launch a first MPI/fortran test:

1. Retrieve a simple f90 code (`hellof.f90`) and the bash script to submit the job (`hellof.job`):
   `cp /cm/shared/docsExtra/Examples/hellof.* .`
   (the contents of these files are shown in the appendix A.1, page 10).
2. Load the necessary modules to use Intel fortran and its libraries, MPI and the SLURM scheduler:
   `module load intel/compiler intel-mpi/64 intel/mkl slurm`
3. Compile and build the executable:
   `mpiifort  hellof.f90 -o hellof.x`
4. Submit the job:
   `sbatch hellof.job`
5. If the job is not over, check whether it is running:
   `squeue -u $USER` (to monitor only your own jobs)
6. When finished, check the output:
   `cat hellof-*.out`
   which has to contain 8 lines (since the job asked for 2×4 tasks) near:

```
node           0 : Hello world
node           4 : Hello world
...
```

`/cm/shared/docsExtra/Examples` contains also a very simple MPI/C code.

4 Work environment: modules

The Modules system allows dynamic changes in the user environment through *modulefiles*. Each *modulefile* contains the necessary information (such as `PATH`, `MANPATH`, `LD_LIBRARY_PATH`, `LM_LICENSE_FILE`...) to use a software (or a specific version of a software).

Usage: `module [ switches ] [ subcommand ] [subcommand-args ]`

The most used switches are:

`avail`	lists all available modules on the system
`load`	loads a module
`unload`	unloads a module
`list`	lists modules currently loaded in a user's environment
`purge`	unloads all modules currently loaded
`initadd`	add module(s) to the shell's initialization file in the user's home directory (`.bashrc`)
`show`	shows environment changes that are made by loading a given module

For example, to use MPI on the cluster with Intel Fortran and the MKL optimized libraries, one can use

```
module load intel/compiler/64/15.0/1.133 intel-mpi/64/5.0.2/044 \
            intel/mkl/64/11.0/2015.1.133
```

If only one version of a module exists, it is not necessary to specify the version number, and if more than one exists, the most recent version (the one with the largest number) will be loaded. Therefore, the previous command can be simplified to

```
module load intel/compiler intel-mpi intel/mkl
```

Be careful with these modules: `mpif90` uses by default `gfortran`. To use the Intel Fortran compiler, call "`mpiifort`" or "`mpif90 -fc=ifort`".
Similarly, for C, one can call "`mpigcc`" or "`mpiicc`".

To know what compiler is used with which options type:
`[mpif90|mpiifort|mpigcc|mpicc|mpiicc] -show`

Loading modules can be done once and for all by adding "`module add`" in the `.bashrc` file (or using `module initadd`).

The `modsbycat` command from the `scp` module displays a categorized list of the available modules.

5 Storage

Several storage spaces are available on the new cluster, storage for the `homes`, local storage on the compute nodes and shared storage on a GPFS parallel file system. Please note that these file systems are not backed up (but homes benefit from *snapshots*).

5.1 Netapp filesystem (homes and shared applications)

All `home` directories are hosted on a 11 TB Netapp partition, part of the Cohersis Netapp FAS3140, accessible from `/home` and limited to 4 GB per user. This storage bay is not part of the Infiniband network and is connected to the new cluster with four 1 Gb links. There is no real backup, but regular *snapshots* are performed.

The snapshot mechanism allows to do very quickly and using very little disk space, a kind of data backup. Indeed, in a snapshot (or immediate copy) the blocks of a file that have been modified since the previous snapshot are set aside and not modified, the new blocks being written elsewhere. Advantage: any user can, without administrator intervention, retrieve a file in the state it was in one of the previous snapshots.

Snapshots policy applied to this Netapp is as follows

- weekly: on Sunday at midnight, 6 kept (`weekly.0` to `weekly.5`);
- daily:
  - 1 per night, Monday through Saturday at midnight, 2 kept (`nightly.0` and `nightly.1`),
  - 1 during the day, at 13h, 2 kept (`hourly.0` and `hourly.1`).

Netapp snapshots can be retrieved (read only) from
`/home/.snapshot/`*snapshotName*`/`*user*
where *snapshotName* is of the form `[weekly|nightly|hourly].[0..5]`, the most recent being `.0`.

This Netapp partition also hosts `/cm/shared` with applications, compilers and auxiliary shared files (this space is often called `/usr/local` on other facilities), and since 09/26/2013, other Netapp partitions are also mounted on the DELL cluster.

To know the current usage and quota on the Netapp partition, use the `quota` command.

5.2 Local storage on the computing nodes

- `cnodexxx`: 2×2 TB HDD with Raid 0,
- `dnodexx`: 20×600 GB HDD with Raid 6+0, and 4×200 GB SSD configured as cache for frequently accessed data,
- `gpuxx`: 2×2 TB HDD with Raid 0.

Please note that these local file systems mounted on `/local` have to be considered as temporary scratch storage (except on `smp01`), and their contents may be destroyed at any time.

Reminder of Raid levels:

Raid 0 (or *striping*):	data spread on 2 disks (rapidity),
Raid 1:	mirroring (security),
Raid 5:	striping with distributed parity,
Raid 6:	striping with double distributed parity (here, 2 parity disks for 8 data disks),
Raid 6+0:	RAID 0 array striped across RAID 6 elements,
Raid 10 (or 1+0):	stripe of mirrors.

A more detailed description of Raid is given on Wikipedia for standard levels and nested levels.

5.3 Global shared GPFS storage

The 524 TB on the GPFS parallel file system are organized with Raid 6 (2 parity disks for 8 data disks) and distributed across 3 partitions with different operating policies:

1. `/gpfs/users`: (100 TB) with 500 GB quota per user,
2. `/gpfs/groups`: (200 TB) with 1 TB quota per Unix group,
3. `/gpfs/scratch`: (100 TB) where files older than 15 days can be destroyed (after notice).

Data belonging to a user no longer working in S-CAPAD would be deleted six months after departure without notice.

Attention:

Regarding GPFS, `df` produces a non standard output:

```
[moguilny@malbec01 ~]$ df -h /gpfs/users
Filesystem            Size  Used Avail Use% Mounted on
/dev/gpfsstorage      100T   43T   58T  43% /gpfs

[moguilny@malbec01 ~]$ df -h /gpfs/groups
Filesystem            Size  Used Avail Use% Mounted on
/dev/gpfsstorage      200T   16T  185T   8% /gpfs

[moguilny@malbec01 ~]$ df -h /gpfs/scratch
Filesystem            Size  Used Avail Use% Mounted on
/dev/gpfsstorage      100T   48T   53T  48% /gpfs
```

The filesystem shown is `/dev/gpfsstorage` while the quota shown is for a `fileset` (`users`, `groups` or `scratch`).

With the `gpfs` module loaded, quotas (limitations, current usage) are displayed with:

- `mmlsquota --block-size auto` (for a user),
- `mmlsquota --block-size auto -g` *group* (for a Unix group).

To display disk usages on all partitions with quota type:
`myquotas` (from `scp` module).

6 Job management with SLURM

Job execution must be performed on the compute nodes (cpu, gpu or data), and never on the login nodes (malbec)! Login nodes should be used only for editing, compiling codes and submitting jobs (on the most appropriate partitions) with SLURM commands.

Each compute node in the DELL cluster has 2 sockets (processors) with 8 cores each; so each node has 16 cores (or CPUs in the SLURM documentation).

6.1 Job partitions (or queues)

Partitions are defined as follows

`cpunormal`	default partition, uses `cnode[001-092]`, MaxTime=48h and MaxNodes=32;
`cpushort`	uses `cnode[001-092]`, MaxTime=120mn and MaxNodes=8, with higher priority than `cpunormal`;
`cpushortprio`	uses `cnode[093-096]`, MaxTime=60mn and MaxNodes=4 ;
`cpuprio`	no limitation, only "on demand"; (MaxTime and MaxNodes will be then configured by the administrator);
`gpu`	uses `gpuxx` nodes;
`data`	uses `dnodexx` nodes with SSD, MaxTime=7d;
`datalowprio`	uses `dnodexx` and `cnodexxx` nodes, running jobs will be suspended or killed as soon as a job submitted on a `cpu*` partition needs the cnodes;
`killable`	reserved for real-time earthquake processing on `smp01`; a job sent to this partition can be stopped and re-queued at any time if necessary, without notice.

Please note, the parameters given above may change to adapt to the cluster usage. To know the current configuration, use the `scontrol` command.

Partitions configuration: `scontrol show partitions`, or
One specific partition configuration: `scontrol show partitions` *PartitionName*.

An overview of partitions and nodes is displayed with `sinfo` where the partition ending with "*" is the one where the jobs are sent by default.

6.2 Job submission

The main submission options are:

Long version	Short	
`--job-name=`*jobname*		
`--nodes=<`*minnodes*`[-`*maxnodes*`]>`	`-N`	
`--ntasks=`*ntasks*	`-n`	number of tasks to run
`--ntasks-per-core=`*ntasks*		
`--ntasks-per-node=`*ntasks*		
`--pty`		open a pseudo terminal (for interactive mode)
`--mem=`*x*		memory per node (en MB)
`--mem-per-cpu=`*x*		minimum memory per CPU (core)
`--exclusive`		the job can not share nodes with other running jobs (>< `--share`)
"partition="*PartitionName*	`-p`	partition name where the job has to run
`--nodelist=`*NodeNameList*	`-w`	name of the nodes to use
`--output=`*FileName*	`-o`	name of the standard outputfile
`--dependency=<`*dependency_list*`>`	`-d`	see below

Interactive mode: `srun` *[options] executable [arguments]*

Example of submission of an interactive job on 1 core:

```
srun -n1 -c1 --pty  myjob
```

`matlab launch`[1]:

```
srun -n1 --ntasks-per-node=1 -p data matlab -display $DISPLAY -desktop
```

Example of an interactive job that requires opening an X window on the data partition:

```
srun -J interactive --nodes=1 --ntasks-per-node=1 -p data xterm
```

Batch: `sbatch` *[options] script [arguments]*

`sbatch` takes the name of a script as argument.
The SLURM options are described in lines starting with `#SBATCH`.
The modules which are loaded when submitting the job will be known by used compute nodes.

By default, the output filename is `slurm-`*jobid*`.out`. If this name is changed (`--output` option) keep the *jobid* in the file name (see `%j` utilization in hello.f page 10) or in the output file itself (see `$SLURM_JOB_ID` utilization in the MPI script below).

MPI applications can be launched either with `mpirun` (red lines in the following example), or with `srun` (green lines commented out).

MPI script example

```
#!/bin/sh
#SBATCH -J HPL-1node
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=16
#SBATCH --ntasks=64
#SBATCH --partition cpunormal

echo "Job $SLURM_JOB_ID"
echo "Running on: $SLURM_NODELIST"
cd $SLURM_SUBMIT_DIR
module purge
module load slurm

module load intel-mpi/64
# -tmi select specific network fabric for inter-node communication

mpirun -tmi ./xhpl


#SLURM_LIB=$( echo $LD_LIBRARY_PATH | sed -e 's/.*:\(\/.*\/slurm\/.*\/lib64\).*/\1/' )
#export I_MPI_PMI_LIBRARY=$SLURM_LIB/libpmi.so
#echo "Setting I_MPI_PMI_LIBRARY=$I_MPI_PMI_LIBRARY"

#srun ./xhpl
```

For more information on MPI:

- type `mpirun -h` (after having loaded the appropriate modules),
- see `IntelMPIlibRefManual.pdf` in `/cm/shared/docsExtra`.

[1] see details on available `matlab` versions in `/cm/shared/docsExtra/README.matlab`.

OpenMP script example

```
#!/bin/sh
#SBATCH -J OPENMP-1node
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=11
#SBATCH -p cpunormal

# cc -fopenmp omp_hello.c -o omp_hello.x
echo "Running on: $SLURM_NODELIST"
cd /home/moguilny/Tests/Base
export OMP_NUM_THREADS=$SLURM_NTASKS_PER_NODE

./omp_hello.x
```

Hybrid MPI/OpenMP script example

```
#!/bin/sh
#SBATCH -J HYBRID-MPI-OPENMP
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=3 # number of MPI processes per node
#SBATCH --cpus-per-task=4   # number of OpenMP threads per MPI process
#SBATCH --ntasks=6          # number of MPI processes
#SBATCH -p cpunormal

echo "Running on: $SLURM_NODELIST"
export TOTAL_NTASKS=$(($SLURM_NNODES*$SLURM_NTASKS_PER_NODE))
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

mpirun -tmi -n $TOTAL_NTASKS ./hybrid-hello
```

To use the GPU cards, load at least the `cuda50/toolkit` module and add in the SLURM script:

`#SBATCH --gres=gpu:`*N*

where *N* is the number of GPU cards to use per node (up to 2 in the current configuration of the DELL cluster).

To run jobs in sequence, see the `--dependency` option. Use it to avoid having more than one big job ($\geq$ 100 cores) running at a given time and therefore prevent others from working! For example, one can ensure that when a series of jobs with the same jobname is submitted, no more than one will run at given moment:

```
#SBATCH --job-name=only_one
#SBATCH --dependency=singleton
```

With these options, when a job with `only_one` as jobname ends, one of the others with the same jobname will be able to start.

Other examples of SLURM scripts (`.job` files) can be found in `/cm/shared/docsExtra/Examples` and `/home/cmsupport`.

Resource allocation: `salloc` *[options] [arguments]*

`salloc` allows to obtain a SLURM job allocation (a set of nodes), execute a command and then release the allocation when the command is finished. This can be used to open an interactive session on a compute node.

```
[moguilny@malbec03 ~]$ salloc -n 1
salloc: Pending job allocation 181763
salloc: Granted job allocation 181763
[moguilny@malbec03 ~]$ hostname
malbec03
[moguilny@malbec03 ~]$ srun --pty /bin/sh
sh-4.1$ hostname
cnode001
sh-4.1$ exit
exit
[moguilny@malbec03 ~]$ exit
exit
salloc: Relinquishing job allocation 181763
salloc: Job allocation 181763 has been revoked.
[moguilny@malbec03 ~]$
```

The command given as argument of `srun` will be executed on the 1st (and here the only one) node allocated to the job.

6.3 Other monitoring and control commands

`squeue`	lists running and waiting jobs
`scontrol show jobs`	shows details on running jobs
`scontrol show nodes`	shows configuration/state of the cluster nodes
`sacct`	displays resources used by the jobs
`scancel`	kills a job
`sview`	gives a graphical view of the jobs, partitions...

The output of many commands is configurable. For example, to have `squeue` display, among other things, the number of cores used by jobs type:

```
squeue -o "%8u %t %12P %.7i %16R %.4C %.12M %j"
```

For a more readable output of the resources used by jobs type:

```
sacct -X -S 2013-08-10 --format "JobID,User,Partition,Ncpus,Start,CPUTime"
```

Be careful, killing a Unix process attached to a job does not necessarily destroy the SLURM job (which must be destroyed with `scancel`). When using `kill`, check that the cores are no longer allocated with `sview` for example.

6.4 More SLURM documentation

Detailed Syntax:	`man` *slurm-cmd*
Briefly:	*slurm-cmd* `--help`
Very briefly:	*slurm-cmd* `--usage`
And, on the Web:	`http://slurm.schedmd.com/`

The `SlurmV2.2.pdf` file in `/cm/shared/docsExtra`, is also interesting but doesn't cover the current version (2.5.7) installed on the cluster.

7 General documentation and other points

The cluster is administered with BRIGHT CLUSTER MANAGER. The user documentation is available at
`http://support.brightcomputing.com/manuals/6.1/user-manual.pdf`
which contains, in addition to a general description of the clusters,

- a section about modules (2.3),
- a chapter (3) and an appendix (A) about MPI,
- a chapter about SLURM (5),
- a chapter about GPUs (8),
- a chapter about the User Portal (10) providing a global view of the cluster and jobs running. This portal can be accessed from the IPGP at `https://syrah`.

Other documents and examples can be found in `/cm/shared/docsExtra` and its subdirectories.

Other useful commands:

- password change: `passwd` command from `malbec`;
- visualisation tools: `evince`, `display`, `xmgrace`, `sac`;
- versionning: `subversion`, `git`, `mercurial`.

Finally, users are not supposed to misuse the resources they have access to. Users whose applications require many resources (number of cores, duration of a run or number of runs) and are fully operational, are encouraged to apply for supplementary computing hours on the National Computing Centers (CINES, IDRIS CCGT) using the DARI procedure.

For any question or comment, send an email to scapad@ipgp.fr.

A Appendix

A.1 Impatients example files

hellof.f90

```
    program hellof
    include 'mpif.h'
    integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)

    call MPI_INIT(ierror)
    call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
    call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
    print*, 'node', rank, ': Hello world'
    call MPI_FINALIZE(ierror)
    end
```

hellof.job

```
#!/bin/bash
#SBATCH --output hellof-%j.out
#SBATCH --output hellof.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4

mpirun -tmi ./hellof.x
```

A.2 List of acronyms

GPFS	(IBM) General Parallel File System
GPU	Graphics Processing Unit
MKL	(Intel) Math Kernel Library
MPI	Message Passing Interface
OpenMP	Open Multi-Processing
SLURM	Simple Linux Utility for Resource Management
SMP	Symmetric MultiProcessing (shared memory)