Idris H2OLAB installation

Nadir SOUALEM

1 Introduction

1.1 Vargas Server

1.1.1 Specifications

vargas is an IBM eServer (Regatta Power6) with the following specifications:

- 3.584 Power6 cores
- 17,5 To Memory
- 67,3 Tflops
- InfiniBand Network x4 DDR

Power6 processor specification:

- Dual-core 2-way SMT
- 4,7 GHz frequency
- Cache L1 data : 64 Ko
- Cache L1 instructions : 64 Ko
- Cache L2 : 4 Mo per core
- Cache L3 : common 32 Mo

1.1.2 Working directories

`$HOME` is limited to 300 Mo , for H2OLAB you have to work in `$WORKDIR`(limited to 400 Go). To check your `$HOME` quota, use:

```
$ quota_u
```

for your `$WORKDIR`:

```
$ quota_u -w
```

You must know that your account is limited to 150 000 inodes for the team project, you can ask the Idris institute (`assist@idris.fr`) to increase your quota inodes.

To check the hours you consume:

```
$ cpt
```

You can find further informations in `http://www.idris.fr` in the technical support section.

1.2 Environment

First of all, let's replace default shell by the Bash one:
`$ chsh`
and choose */bin/bash*.
Note: Logout and login again to see the change (or use **exec** to change shells).
Next, create two configuration files *.bashrc* and *.bash_profile* in the `$HOMEDIR`:
`$ touch .bashrc .bash_profile`
Here is *.bashrc*

```
export HYDROLAB_ROOT=$WORKDIR/software/H2OLAB
export LIB_EXT=$WORKDIR/software/lib_ext
export CC=mpcc CXX=mpCC
module -s load svn
module -s load cmake
module -s load blas
module -s load lapack
```

In your *.bash_profile*:

```
# ~/.bash_profile: executed by bash(1) for login shells.
# User specific environment and startup programs, for login setup
# Functions and aliases go in ~/.bashrc
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
   . ~/.bashrc
fi
```

At last, reload the new configuration by sourcing your *.bashrc*:
`$ source .bashrc`

2 Installation

2.1 Checkout from Inria-Gforge repository

First, create two directories *h2olab,lib_ext*:

```
$ mkdir -p $HYDROLAB_ROOT/svn $LIB_EXT
```

Next, checkout trunk and external libraries from Gforge Inria source manager

```
$ cd $HYDROLAB_ROOT/svn
$ svn checkout svn+ssh://user@scm.gforge.inria.fr/svn/hydrolab/trunk .
$ cd $LIB_EXT
$ svn checkout svn+ssh://user@scm.gforge.inria.fr/svn/h2olibext/unix .
```

Note: you can remove non-regression tests directory in the view to reducing your quota inodes.

2.2 Installation

Install external libraries

```
$ cd $LIB_EXT
$ sh ./install
```

install parameters files

```
$ cd $HYDROLAB_ROOT/svn/install/linux
$ chmod +x install
$ ./install
```

3 Compilation of H2OLAB

3.1 Out-Source Building

This is the prefered mode. To use Release out-of-source build just create a directory **outside** of *svn* directory. For example to build PARADIS:

```
$ mkdir build
$ cd build
$ cmake $HYDROLAB_ROOT/svn
$ gmake -j8 PARADIS
```

To specify Debug mode configuration, just replace **cmake** command-line by `cmake -DCMAKE_BUILD_TYPE=Debug $HYDROLAB_ROOT/svn`

3.2 In-Source building

To use in-source build just go to `$HYDROLAB_ROOT/svn`:

```
$ cd $HYDROLAB_ROOT/svn
$ cmake .
$ gmake -j8 PARADIS
```

To specify Debug mode configuration, just replace **cmake** command-line by `cmake -DCMAKE_BUILD_TYPE=Debug .`

3.3 Update your code

To update your code:

```
$ cd $HYDROLAB_ROOT/svn
$ svn update *
```

Note: * is used to avoid checking out the non-regression tests repository if it was removed.

3.3.1 Out-Source updating

Remove the cache from your *build* directory:

```
$ cd build
$ rm CMakeCache.txt
$ cmake $HYDROLAB_ROOT/svn
```

3.3.2 In-Source updating

Remove the cache from your *svn* directory:

```
$ cd $HYDROLAB_ROOT/svn
$ rm CMakeCache.txt
$ cmake .
```

Note: you reload cmake if someone changes the project tree (it happens sometimes).

4 Execution of H2OLAB

PARADIS is in *$HYDROLAB_ROOT/H2OLAB_INSTALLATION/$CONFIGURATION/bin*, for example for Release Configuration:

```
$ cd $HYDROLAB_ROOT/H2OLAB_INSTALLATION/Release/bin
$ ls -l PARADIS
-rwx------    1 rsli003  sli        78142139 Nov 02 12:17 PARADIS
```

4.1 Interactive Execution

Interactive jobs have

- an execution time limited to 20 minutes wall-clock
- a memory size limited to **data=3.2GB**, **stack=0.5GB,0.5GB**
- a number of processors limited to 32

To run *PARADIS* with 4 processors:

```
$ mpiexec -n 4 ./prog
```

If processors are not available, you get the following message:

```
ERROR: 0031-365  LoadLeveler unable to run job, reason:
LoadL_negotiator: 2544-870 Step vargas001.1613.0 was not considered
to be run in this scheduling cycle due to its relatively
low priority or because there are not enough free resources.
```

4.2 Batch Execution using LoadLeveler software

When jobs are submitted to LoadLeveler, they are not necessarily executed in the order of submission. Instead, LoadLeveler dispatches jobs based on their priority, resource requirements and special instructions; for example, administrators can specify that long-running jobs run only on off-hours, that short-running jobs be scheduled around long-running jobs or that certain users or groups get priority. Now, we are going to submit a job to LoadLeveler queue by using the **llsubmit** command. Here is a template command file *PARADIS.ll*, you can use to submit the job to LoadLeveler:

```
# Job Name
# @ job_name = PARADIS
# Standard Output file
# @ output = $(job_name).$(jobid)
# Error Output file
# @ error =  $(job_name).$(jobid)
# Type of job: serial, parallel
# @ job_type = parallel
# Number of process
# @ total_tasks = 16
# Time ELAPSED max.
# @ wall_clock_limit = 00:10:00
# Mail Notification
# @ notify_user = username@domain
# @ notification = error
# @ queue

# To get the command
set -x
mpiexec -n 16 ./PARADIS
```

Finally, we submit the job to the queue using:

```
$ llsubmit PARADIS.ll
```

Note that you can specify stack and data limits using the following parameters in your command file:

```
# @ stack_limit=size,size
# @ data_limit=size
```

where size is specified in GB. By default: **data=3.2GB**, **stack=0.5GB,0.5GB** per node. **data+stack** must not exceed 3.7 GB with **data $<$ 3.2** Go and **stack $<$ 3.2 Go** per node.

4.3 LoadLeveler usage

4.3.1 Job status information

llq gives you the job status information, you can also use **Qstat** command:

```
$ llq -u $USER
Id                  Owner    Submitted   ST PRI Class  Running on
------------------ -------- ----------- -- --- ------ -----------
vargas043.318726.0  rsli003  11/4  07:50 R  100 c32t4  vargas095
```

Id Job identifier presented in the format: host.jobid.stepid

Owner Userid of the job submitter

Submitted Date and time of job submission

ST Current job status (state). Job status can be:

C Completed

CA Canceled

I Idle

R Running

RM Removed

The class is the number of processors as a function of the elapsed time.

```
T(h)^ ELAPSED
    |
100h+------+-------+-------|
    | c8t4 | c16t4 | c32t4 |
 20h+------+-------+-------+-------+--------+--------+--------|
    | c8t3 | c16t3 | c32t3 | c64t3 | c128t3 | c256t3 | c512t3 |
 10h+------+-------+-------+-------+--------+--------+--------|
    | c8t2 | c16t2 | c32t2 | c64t2 | c128t2 | c256t2 | c512t2 |
  1h+------+-------+-------+-------+--------+--------+--------|
    | c8t1 | c16t1 | c32t1 | c64t1 | c128t1 | c256t1 | c512t1 |
   0+------+-------+-------+-------+--------+--------+--------+->
           8      16      32        64      128      256      512
                                            Number of processors
```

4.3.2 Cancel a Job

llcancel command allows you to cancel a submitted job using associated **Id**:

```
$ llcancel vargas043.318726.0
llcancel: Cancel command has been sent to the central manager.
```