Tardis Tutorial

October 1, 2018

1 Cluster Overview

Tardis is a cluster of 16 computing nodes and a head node. The head node is used for writing/compiling/debugging programs, file serving, and job management.

Among the 16 computing nodes, 12 are "pure" computing node named node01-node12 without GPUs and 4 are "GPU" nodes named gpu01-gpu04 each with 2 Nvidia Tesla M2075 GPUs. Except for GPUs, all nodes are almost identical in CPUs, memory and interconnections. Each node has 2 AMD Opteron 6272 CPUs with total 32 cores, 64GB ECC memory and 40Gbps high bandwidth low latency InfiniBand networks. All nodes are connected to a Mellanox 18 port InfiniBand switch.

The head node has a RAID 6 disk array with 8TB usable storage for users. Each node has 1TB storage for their local use. /home and /act directories are exported to all nodes by NFS. All nodes run CentOS 7.4 operating system.

2 Hardware You need to Know

512 computing cores on 16 compute nodes integrated by Advanced Clustering Technology. All machines are 2-way SMPs with AMD Opteron 6272 processors with 16 cores per socket totalling 32 cores per node.

6272 Opteron Processor Specifications:

- 16 cores, 16 threads
- frequency: 2.1 GHz, Turbo(2.4GHz when more than 8 cores, 3.0GHz when less than 8 cores)
- bus speed: 3.2 GHz HyperTransport links (6.4 GT/s)
- L1 cache: 8*64 KB 2-way associative shared instruction caches, 16*16 KB 4-way associative data caches
- L2 cache: 8*2 MB 16-way associative shared exclusive caches
- L3 cache:2*8 MB up to 64-way associative shared caches
- Average CPU power 80 Watt, thermal design power 115 Watt

Nodes:

- 64GB ECC protected DRAM
- On board QDR IB adaptor or Mellanox ConnectX-3 VPI adapter card, 40Gb/s QSFP, PCIe 3.0 x8 8GT/s and 40GbE
- Nvidia Tesla M2075 1.5GHz, 6GB GDDR5, 448 CUDA cores (2 GPUs per GPU node)
- 1TB temporary disk space each node
- 16 TB hard drive on head node configured in RAID 6; 8TB usable.
- 12 computing nodes: node01-node12
- 4 GPU nodes: gpu01-gpu04

- 1 login node: tardis.cs.ucr.edu

Network:
- Mellanox 18 port IB QDR switch QSFP ports
- 24 port Gigabit switch

3 Software We have

All softwares are 64-bit.
- CentOS 7.4 Linux x86_64
- gcc/gfortran 4.4.7/4.8.5/5.1.0 (C/Fortran compiler)
- MPICH-3.2.1
- OpenMPI-3.0 (Recommended)
- Slurm (batch job manager)
- Nvidia CUDA 8.0

Software in Tardis will be updated over time. Please check the up-to-date software environment on Tardis.

4 Account/Login

Tardis sits behind the department firewall, which means that only computers within UCR CS department are able to access it. If your computer is outside CS network you have to login bell/bass first. **If you want to apply for an account of Tardis, please send an email to Kai Zhao (kzhao016@ucr.edu) specifying how you affiliate with the program.**

5 Modules

There's a convenient tool called **Modules** installed on the cluster for managing different versions of software (notably compilers, MPI implementations).

If you want to switch between different compilers or MPI implementations when you prepare to work on different projects, it's quite tedious to manually rewrite **PATH**, **LD_LIBRARY_PATH** etc without a tool to help us. It's even more tedious if we want to switch back and forth to do some testing. Modules software provides a clean and graceful way to do this exactly.

For example, the system default(CentOS 7.4) GCC compiler is version 4.8.5; we might want to use an older version like 4.4.7. First, we want to see which compilers are available to us, type:

$ module avail

You will get all modules that are available in Tardis. Then if you want to load module **gcc-4.4.7**, type:

$ module load gcc-4.4.7

If you want to unload module **gcc-4.4.7**, type:

$ module unload gcc-4.4.7

The command **$ module purge** will unload all modules. **$ module list** gives you your current loaded modules. If you install an even newer version of GCC in your home directory, you can write a simple module file to use modules to manage it like above. Please consult their website for more information.

6 Before You Run

This cluster uses a batch job system called **SLURM**. The workflow is like this: you write and compile code, write a submission file and submit the job. When the batch system finds enough resources, it grants your job the resources and starts executing it. It's important to NOT to bypass the batch system such as using **mpiexec -f hostfile -np 256 executable**, doing so will confuse the batch system so that it does not know which nodes are free to assign to other jobs. Your job and others' might run on the same node which will result in degraded performance in both jobs.

Please try NOT to run a significant program on head node for too long, for that might affect other people's login experiences. What you should use is the **Slurm Batch System** that will be introduced in the next several sections.

7 Quick Start Guide

7.1 A Sample Sequential Program

When you only want to execute your program sequentially and exclude others from accessing the same node. You should apply only one node with all cores, and run your program without any MPI environment. The following gives a "hello.c" example to help understand how it works in Tardis.

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  printf("Hello Tardis!\n");
  return 0;
}
```

Assume you use default gcc compiler 4.8.5, then you should compile your program by:

$ gcc -o hello hello.c

To submit your sequential job, you also need to write a job script (hello.sh) as follows:

```
#!/bin/bash
#SBATCH --job-name=hello
#SBATCH --output=res.txt
#SBATCH -N 1
#SBATCH -n 32
#SBATCH -t 00:10:00

./hello
```

The job script is basically a bash script. The line **SBATCH –job-name=hello** specifies the name of your job. **SBATCH –output=res.txt** specifies to which file the output should go. **SBATCH -N 1** specifies that your job is requesting 1 nodes. **SBATCH -n 32** means you will use 32 cores in that node, and **SBATCH -t 00:10:00** specifies the job should terminate within 10 minutes wall time. If it doesn't, the job system will kill it properly.

Finally, to submit your job to Slurm batch system, you should type:

$ sbatch hello.sh

7.2 A Sample MPI Program

Here's a hello world MPI program (hellompi.c), and how to compile and run it.

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
  int numtasks, rank, len, rc;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  rc = MPI_Init(&argc, &argv);
  if (rc != MPI_SUCCESS) {
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }

  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(hostname, &len);
  printf ("Number of tasks= %d My rank= %d Running on %s\n",   numtasks, rank, hostname);


  MPI_Finalize();
  return 0;
}
```

To compile it, we first load the appropriate MPI libraries by **Module** and the use the MPI compiler wrappers to compile the code:

$ module purge
$ module load openmpi-3.0/gcc-4.8.5
$ mpicc hellompi.c -o hellompi

To run the hello program, we need to submit a job via Slurm batch system. Here's an example of job script hellompi.sh:

```
#!/bin/bash
#SBATCH --job-name=hellompi
#SBATCH --output=res.txt
#SBATCH -N 2
```

```
#SBATCH -t 00:10:00

mpirun -np 64 ./hellompi
```

SBATCH -N 2 specifies that your job is requesting 2 nodes for the MPI program.

mpirun -np 64 ./hellompi

will run job with 64 processes on 2 nodes finally. To submit your job, you should type:

$ sbatch hellompi.sh

7.3 CUDA

To use GPU card and cuda-8.0 toolkit in Tardis, you should type in terminal:

$ module load cuda/cuda-8.0

After this, you will compile your cuda program with compiler nvcc. Suppose you get the executable called "**gpuexe**". To submit your gpu job to gpu node, you should apply CPU resources as well as GPU resources at the same time. The following give an example of job script:

```
#!/bin/bash
#SBATCH --job-name=gpu
#SBATCH --output=res.txt
#SBATCH --gres=gpu:1
#SBATCH -N 1
#SBATCH -n 32
#SBATCH -t 00:10:00

./gpuexe
```

The line **SBATCH −−gres=gpu:1** explicitly applies 1 gpu from the gpu node. **SBATCH -n 32** specifies the number of cores to use in the gpu node.

7.4 Manage Your Job

After you submit your job to Slurm batch system, you may want to know the job status, or kill the job, or get system information so on and so forth. There are a lot of ways to help you achieve your goals.

If you hope to check the running status, just type: **$ squeue** to get the current status of your job.

If you hope to kill your job because of some reasons, suppose your job id is "100", you should type: **$ scancel 100**.

If you want to check the node status in the system, you can type: **$ sinfo**

Finally, if you want to know more information about your job (with "100" job id), you can use: **$ sstat -j 100**.

For more information about Slurm job system, please refer to Slurm Documentation

8 FAQ

8.1 How to apply specific nodes

In some cases, you may want to apply specific nodes for your job. Assume you only need node01 and node02 these two nodes, you can simply add the following line to your job script without specifying the number of nodes you need:

```
#SBATCH -w, --nodelist=node01,node02
```

8.2 How to run program interactively?

If you want to use a single GPU node or computation node interactively, you might be interested in the interactive mode the Slurm.

Assume you want to apply one node with 32 cores and 1 gpu card resources. You should type:

$ salloc -N 1 -n 32 −−gres=gpu:1

Then Slurm will assign a node for you, and you can find it by typing **squeue**. Suppose gpu01 is assigned for your job. You can type:

$ ssh gpu01

to login gpu01 node and do some work interactively.