INAF-OATs Technical Report 222 - INCAS: INtensive Clustered ARM SoC - Cluster Deployment

S. Bertocco[a], D. Goz[a], L. Tornatore[a], G. Taffoni[a]

[a]*INAF-Osservatorio Astronomico di Trieste, Via G. Tiepolo 11, 34131 Trieste - Italy*

Abstract

The report describes the work done to deploy an Heterogeneous Computing Cluster using CPUs and GPUs integrated on a System on Chip architecture, in the environment of the ExaNeSt H2020 project. After deployment, this cluster will provide a testbed to validate and optimize several HPC codes developed to be run on heterogeneous hardware.

The work done includes: a technology watching phase to choose the SoC best fitting our requirements; installation of the Operative System Ubuntu 16.04 on the choosen hardware; cluster layout and network configuration; software installation (cluster management software slurm, specific software for HPC programming).

Keywords: SoC, Firefly-RK3399, Heterogeneous Cluster Technology HCT, slurm, MPI, OpenCL, ARM

1. Introduction

The ExaNeSt H2020 project aims at the design and development of an exascale ready supercomputer with a low energy consumption profile but able to support the most demanding scientific and technical applications. The project will produce a prototype based on hybrid hardware (CPUs+accelerators). Astrophysical codes are playing a fundamental role to validate this exascale platform. A preliminary work has been done to

Email addresses: sara.bertocco@inaf.it ORCID:0000-0003-2386-623X (S. Bertocco), david.goz@inaf.it ORCID:0000-0001-9808-2283 (D. Goz), luca.tornatore@inaf.it ORCID:0000-0003-1751-0130 (L. Tornatore), giuliano.taffoni@inaf.it ORCID:0000-0001-7011-4589 (G. Taffoni)

port on the heterogeneous platform a state-of-the-art N-body code (called ExaHiGPUs) [4]. The urgency to test step by step the code porting before the release of the final hardware prototype, pushed the need to have a test platform. With this goal in mind, we deployed a minimal heterogeneous computing cluster using multiple processor types (CPUs and GPUs [5]) including three nodes: a manager and two computational nodes. The manager, which is also the log-in node, consists of rescued hardware and hosts some utility software, the shared storage areas and the cluster management software slurm server. The two computational nodes, based on Firefly-RK3399 Single Board Computers, host the cluster management software slurm client, a C++ development compiler and debugger and some utility softwares. In this report we describe in section 2 the preliminary technological watching phase and the reasons to choose the Firefly-RK3399 Single Board Computer as support for the computational nodes; in section 3 the cluster layout and single nodes hardware characteristics; in section 4 the Operating System Ububtu 16.04 LTS installation on Firefly-RK3399 board; in section 5 the network configuration; in section 6 the steps to install specific development and debugging tools optimized for HPC software production; in sections 7 the SLURM cluster management software installation and configuration.

2. Technological watching and the state of the art

The main requirement is to have hybrid hardware (CPUs+accelerators) as platform to perform parallel computing and to test a state-of-the-art astrophysical N-body code (called ExaHiGPUs) [4]. Secondly, low cost, and high power efficiency performance are appreciated.

Raspberry Pi based HPC clusters deployment and test are widely enough well documented, e.g. [2] and [3]. The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The boards are low cost, well-featured, easily available and all models feature a Broadcom system on a chip (SoC) with an integrated ARM compatible central processing unit (CPU) and on-chip graphics processing unit (GPU). These are very interesting hardware: processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ and on-board memory ranges from 256 MB to 1 GB RAM. Secure Digital (SD) cards are used to store the operating system and program memory in either SDHC or MicroSDHC sizes. The boards have one to four USB ports. For

video output, HDMI and composite video are supported, with a standard 3.5 mm phono jack for audio output. Lower-level output is provided by a number of GPIO pins which support common protocols like I^2C. The B-models have an 8P8C Ethernet port and the Pi 3 and Pi Zero W have on-board Wi-Fi 802.11n and Bluetooth. Unfortunately Raspberry Pi lacks in computing power and its slow 100 Mbps networking significantly limits such clusters to parallel computational tasks that require high inter-node data communications relative to task run [3]. This makes it difficult to run computational intensive software on it.

A study of current market available technology brought us to choose Firefly-RK3399 as single board computer to install our nodes. It is a six-core 64-bit High-Performance Platform, 4GB DDR3, and on-board eMMC with 32GB storage, powered by the ARM Mali T-860 GPU. The reasons to choose it are:

- the heterogeneous hardware: dual 2.0GHz CPU clusters, one cluster houses two Cortex-A72 high-performance cores and another cluster contains four Cortex-A53 cores and a 4K capable Quad-Core ARM Mali-T864 GPU;
- the 4GB DDR3 RAM which is a minimum requirement to run our software;
- the on-board gigabit ethernet.

3. Cluster layout and hardware datasheets

Our use case was to deploy a cluster based on heterogeneous hardware (CPU+GPU) to validate the parallel implementation of the N-body code on such kind of platform. Chosen the hardware for the computation nodes, we decided to deploy a minimal cluster with only two computational nodes, since this will be enough to perform some simple test to proof the suitability of the hardware and software layout for our requirements, and, at the same time, to minimize the economical effort.

Figure 1 shows the cluster components layout and the network connections. The three computers, the master node and the two computation nodes, are connected through a switch in a local area network. The master node has two network cards, one cabled and configured to connect to the wide area network (WAN) and another one connected to the switch and configured to

link in a local area network (LAN). This way, the master node, suitably configured as described in section 5, will be also a gate between the LAN and the WAN.

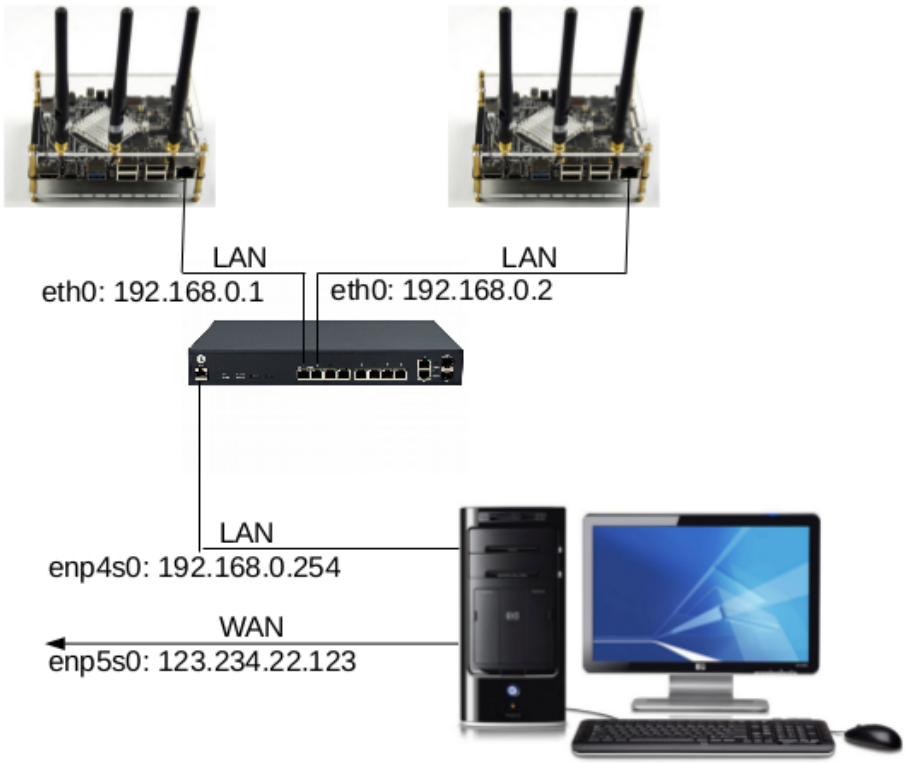

Figure 1: Cluster components, layout and networking

The tables 1 and 2 show the main technical features of the cluster components and the Operating System installed. Firefly-RK3399 is provided with a dual boot system, Android and Ubuntu. Because our software is built on Linux, we decided to install Ubuntu 16.04 LTS as Operative System in all the nodes. This way, we have all the cluster components homogeneous from the Operating System point of view.

4. Operating System Ububtu 16.04 LTS installation on Firefly-RK3399 board

The Operating System installation on Firefly-RK3399 single board computer deserves a dedicated section because the on-line available documen-

Master node datasheet	
Motherboard	Foxconn 2ABF
CPU	Intel(R) Core(TM) i3-3220 CPU @ 3.30GHz 64 bits
Ram memory	4+4GB DIMM DDR3 Synchronous 1600 MHz (0,6 ns)
Network	2xGigabit Ethernet network devices
Operating System	Ubuntu 16.04.4 LTS

Table 1: The main hardware characteristics of our master node.

Computation nodes datasheet	
Soc	Rockchip RK3399 (28nm HKMG Process)
CPU	Six-Core ARM 64-bit processor (Dual-Core Cortex-A72 and Quad-Core Cortex-A53)
GPU	ARM Mali-T860 MP4 Quad-Core GPU
Ram memory	4GB Dual-Channel DDR3
Network	1000Mbps Ethernet
Power	DC12V - 2A (per node)
Operating System	Ubuntu 16.04.2 LTS

Table 2: The main hardware characteristics of our computational nodes.

tation is very poor and the promised dual boot was not correctly working on the newly purchased hardware. Although instructions on how to boot from SD Card are available, we found easier to install Ubuntu flashing the Rockchip provided image on the on board eMMC following the steps below:

- Choose a machine to be used as installation server:
  in our case we used the master node, already equipped with Ubuntu 16.04 LTS;

- Download the upgrade tool provided by Rockchip:
  by the github site (providing the current version)

  `https://github.com/rockchip-linux/tools/tree/master/ \\`

or by the web page (providing the last stable version)

`http://en.t-firefly.com/doc/download/index/id/3.html`

We choosed the web page download: on the web page, click on the link *"Firmware Upgrade Tool"* to download the current stable version of Linux_Upgrade_Tool .zip file and install it.

- Download the Ubuntu image provided by Rockchip:
  go to the web page

  `http://en.t-firefly.com/doc/download/index/id/3.html`

  and click on the link *"Ubuntu 16.04 image"* to download the current stable image.

- Enter the Firefly RX3399 board in recovery mode:

  - connect the board with the installation server through a usb type-C cable;
  - swich-on the board keeping pushed the recovery button shown in figure 2.

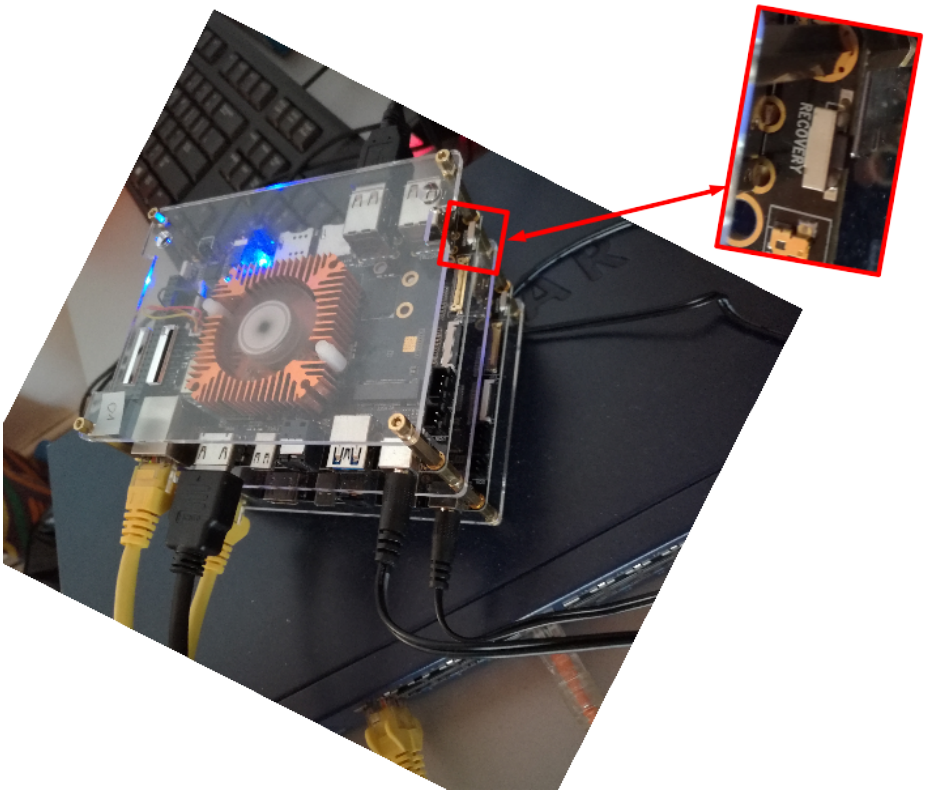

Figure 2: Zoom of the "recovery button" placement in the board

- Flash the Ubuntu image using the command:

```
sudo ./upgrade_tool uf ../Firefly-RK3399-ubuntu16.04-20180416112819.img
```

At this point the OS is installed and up and the default user to access the system is:

```
username: firefly
password: firefly
```

To ensure better performances of the computation nodes, we switched off the X Window Manager. To ensure this situation will be maintained on the next boot, we used the command:

```
sudo systemctl set-default multi-user.target
```

If needed, to return to default booting into X, it can be used:

```
sudo systemctl set-default graphical.target
```

To see the current default target:

```
sudo systemctl get-default
```

5. Network configuration

The network layout is displayed in figure 1. It is a private network connecting the master node and the computational nodes. The master node has two network cards, one connected to the LAN in order to link it to the computational nodes and another one connected to the wide area network, through which the cluster can be reached by outside the LAN.

The master public host-name and IP addresses are configured in the OATs-INAF institutional DNS and DHCP.

In the LAN there is neither a DNS nor a DHCP. Having only three nodes, the host names and IP addresses of all the cluster nodes are configured in $\backslash etc \backslash hostname$ and $\backslash etc \backslash hosts$ respectively.

The master node is set up as a router: the 2-network interface cards (NIC) are enp4s8 (LAN) and enp5s8 (WAN). NAT (Network Address Translation) is enabled on the WAN-NIC enp5s8. Details are in section 5.2. The cluster computation nodes can reach external (WAN) hosts by name because a DNS host is configured 5.3.

5.1. Host names, domain and cluster host name resolution

An example of the nodes network configuration files is below. Note that **a domain *cluster.local* is defined for the LAN**.

- Node name set up:

```
# sudo vi /etc/hostname
node01
```

- Node names resolution in the LAN:

```
# sudo vi /etc/hosts
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
```

```
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.0.1 node01 node01.cluster.local
192.168.0.2 node02 node02.cluster.local
192.168.0.254 masternode masternode.cluster.local
```

5.2. Iptables and NAT configuration on master host

The master host has an Ubuntu 16.04 LTS OS and it has been transformed into a router establishing basic NAT (Network Address Translation) and activating port forwarding. Data of the problem:

```
enp5s8 = Network adapter connected to the Internet (WAN)
enp4s8 = Network adapter connected to the internal network (LAN)
123.234.22.0 = Subnet for enp5s8
192.168.0.0 = Subnet for enp4s8
123.234.22.123 = IP address of enp5s8
192.168.0.254 = IP address of enp4s8
```

NAT configuration:

1. enable IP forwarding:
   edit */etc/sysctl.conf* and uncomment:

   ```
   # net.ipv4.ip forward=1
   ```

   so that it reads as follows:

   ```
   net.ipv4.ip forward=1
   ```

2. enable the IP masquerading creating the following iptables rules:

   ```
   $ cat iptables.nat.txt
   *nat
   :PREROUTING ACCEPT [6:1229]
   :FORWARD ACCEPT [0:0]
   :OUTPUT ACCEPT [1:1980]
   :POSTROUTING ACCEPT [0:0]
   -A PREROUTING -j LOG --log-prefix "PREROUTING" --log-level 7
   -A POSTROUTING -o enp5s0 -j MASQUERADE
   -A POSTROUTING -j LOG --log-prefix "POSTROUTING" --log-level 7
   -A OUTPUT -j LOG --log-prefix "NAT_OUTPUT" --log-level 7
   -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
   ```

```
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -s 192.168.0.0/21 -j ACCEPT
-A INPUT -s 123.234.22.0/21 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i enp5s0 -o enp4s0 -m state \\
           --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i enp4s0 -o enp5s0 -j ACCEPT
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

3. Enable these iptables rules:

   ```
   iptables-restore < iptables.nat.txt
   /etc/init.d/networking restart
   ```

4. Until now, the iptables rules are stored in volatile memory. To make them persistent:

   ```
   sudo /etc/init.d/iptables-persistent save
   ```

   To reload the iptables rules:

   ```
   sudo /etc/init.d/iptables-persistent reload
   ```

5.3. DNS set on nodes

The cluster computation nodes are able to contact by name whichever computer in the internet (i.e. outside the LAN) because in each of them a DNS is configured. Because the computation nodes are working with X Window Manager disabled (run level 3), we needed to use a textual tool for the network configuration . We chosen *nmcli*. Steps to set the DNS:

- to obtain active connection name:

  ```
  nmcli connection show --active
  ```

- to list available connections and choose the appropriate

  ```
  nmcli connection edit
  ```

  and double tab

- remove eventually existent DNS and set the right DNS (primary and secondary):

```
nmcli> remove ipv4.dns
nmcli> set ipv4.ignore-auto-dns yes
nmcli> set ipv4.dns 123.234.22.XX 123.234.24.X
nmcli> save
nmcli> quit
```

- restart the network connection:

```
nmcli connection down our_connection_name
nmcli connection up our_connection_name
```

6. Installation of development and debugging tools optimized for HPC software production

The software development environment has been tailored for HPC software production installing the last version of the GNU C++ compiler, recompiling some libraries where needed, and compiling and installing valgrind for debugging and profiling, PMIx and OpenMPI for MPI.

The software has been uploaded from the GNU repository

`ftp://ftp.gnu.org/gnu`

The performed steps are enumerated below:

- install autoconf libtool
- building chain:

```
[ 1 ] gmp-6.1.2
            - ./configure --prefix=/usr && \\
                    make -j4 && \\
                    make check && sudo make install

[ 2 ] mpfr-4.0.1
      - ./configure --prefix=/usr &&
                    make -j4 && sudo make install

[ 3 ] mpc-1.1
```

```
      - ./configure --prefix=/usr && \\
                    make -j4 && sudo make install

[ 4 ] isl-0.18
      - ./configure --prefix=/usr && \\
                    make -j4 && sudo make install

[ 5 ] valgrind

[ 6 ] gcc-7.3.0
      - >>> important:
            CPLUS_INCLUDE_PATH must be undefined before make
                export CPLUS_INCLUDE_PATH=

      - mkdir gcc_build && cd gcc_build \\
            - ../%gcc_src_dir/configure --prefix=/usr \\
            --enable-lto --disable-multilib --enable-threads \\
            --enable-languages=c,c++,lto \\
            --enable-valgrind-annotations --program-suffix=-7 && \\
            make -j4 && sudo make install

      - use update-alternatives to manage the multiple versions
        of the software we have newly installes:
            sudo update-alternatives --install /usr/bin/gcc gcc \\
             /usr/bin/gcc-5 50 --slave /usr/bin/g++ g++     \\
             /usr/bin/g++-5 --slave /usr/bin/gcc-ranlib     \\
             gcc-ranlib /usr/bin/gcc-ranlib-5
            sudo update-alternatives --install /usr/bin/gcc gcc  \\
             /usr/bin/gcc-7 100 --slave /usr/bin/g++ g++     \\
             /usr/bin/g++-7 --slave /usr/bin/gcc-ranlib      \\
             gcc-ranlib /usr/bin/gcc-ranlib-7

[ 7 ] recompile all previous lib

[ 8 ] libevent from libevent.org
      - ./configure --prefix=/usr

[ 8 ] PMIx from https://pmix.org  <-- needs libevent
```

```
      - pmi-1.2.5 ./configure --prefix=/usr/
      - mpi-2.1.1 ./configure --prefix=/usr

[ 8 ] openmpi 3.1  <-- with slurm need pmi-1 + mpi-2

      - ./configure --prefix=/usr --enable-branch-probabilities \\
                --enable-mpi-cxx --disable-mpi-fortran      \\
                --enable-cxx-exceptions --with-valgrind     \\
                --with-slurm --with-pmi

[ 9 ] recompile valgrind
            - ./configure --prefix=/usr --with-mpicc=mpicc

[10 ] gsl-2.4
            - ./configure --prefix=/usr

[11 ] fftw-3.3.7

      - ./configure --prefix=/usr --enable-generic-simd128   \\
                --enable-fma --disable-fortran --enable-mpi

[12 ] pfft-1.0.8

[13 ] fftw-2.1.5

      - ./configure --prefix=/usr/ --enable-threads \\
                --enable-mpi --disable-fortran
```

7. SLURM cluster management software installation and configuration steps

SLURM (Simple Linux Utility for Resource Management) is a free and open-source job scheduler for Linux and Unix-like kernels, used by many of the world's supercomputers and computer clusters. It provides three key functions: it allocates access to resources (compute nodes) to users for some duration of time so they can perform work; it provides a framework for starting, executing, and monitoring work (typically a parallel job such as MPI) on a set of allocated nodes; it arbitrates contention for resources by

managing a queue of pending jobs.

We choosed SLURM because it is an open source code, so it does not require a license fee and, in case of need, it can be customized. The customization can be done adding code without changing the entire source code, thanks to its modular architecture. Moreover, SLURM is suitable for high computing systems because it can handle thousands of jobs at one time and it is widely used to run parallel MPI jobs, which is our use case.

Below it is a description of steps and commands we performed to install the SLURM cluster.

7.1. Clock synchronization

Clock on all cluster machines must be syncronized because out-of-sync time can cause errors, data corruption, and other hard to debug issues.

Ubuntu 16.04 has time synchronization built in and activated by default using systemd's timesyncd service. For first the time on all nodes can be find and verified with the most basic command "date". The output is on the form:

```
Tue Jul 31 17:57:24 CEST 2018
```

If the time zone is not aligned, it can be changed in this way:

- list the available time zones:

  ```
  timedatectl list-timezones
  ```

- change the zone, for example to Rome time:

  ```
  sudo timedatectl set-timezone Europe/Rome
  ```

The status of timesyncd and so the time and date synchronization, can be checked running "timedatectl" with no arguments. It is not need to use "sudo" in this case:

```
timedatectl
```

If the answer contains a row like:

```
NTP enabled: yes
NTP synchronized: yes
```

then timesyncd is active and time and date must be synchronized, else, if timesyncd is not enabled, it can be turn on with t“imedatectl”:

```
sudo timedatectl set-ntp on
```

At the end it can be run “timedatectl” to confirm the network time status. It may take a minute for the actual sync to happen.

7.2. NFS mounted shared storage areas

The correct run of the HPC cluster requires to share users home directories and a common storage area where to put software to be run and jobs input and output. We chosen to share these spaces as NFS mounted directories [1]. The space will be allocated on the master host and mounted on the computation nodes. Performed steps below.

1. Needed packages installation:
   - on master host (used as NFS server):

     ```
     sudo apt-get install nfs-kernel-server
     ```
   - on computational nodes (NFS clients):

     ```
     sudo apt-get install nfs-common
     ```
   - start the NFS service:

     ```
     sudo systemctl start nfs-kernel-server
     ```
2. Creation of directories to share, both on master and computation nodes, and assignment of right permissions and ownership. A sharing group must be created on all nodes and all submitting users must belongs to it. We created a group with the name of the project: “exanest”. Steps:

   ```
   mkdir \data   # user's common storage area
   mkdir \u          # users home
   sudo groupadd -g 1333 exanest
   chown root:exanest \data
   ```

[1] The shared directories are not a problem in a so small cluster, but they could being a problem growing the size of the cluster or also depending on jobs data needs because of network traffic. In these cases, a separate storage node could be considered, which would distribute the NFS load. If the jobs run horribly when they span nodes (e.g. MPI jobs) it may need a faster, lower latency interconnect like Infiniband or 10Gb Ethernet. In our case this solution is not workable because the firfly-RK3399 1Gb network connection

```
chown root:exanest \u
chmod 775 \data \u
```

3. Server configuration to export the two directories to all the nodes in the private network:
   - edit the file *\etc\exports*:

     ```
     /data    192.168.0.*(rw,sync,no_subtree_check)
     /u       192.168.0.*(rw,sync,no_root_squash,no_subtree_check)
     ```

   - export the directories running the command "exportfs -ra"
4. Computation nodes configuration to mount the shared directories:
   - *\etc\fstab* modification to ensure the shared directories automount at boot time:

     ```
     $ cat /etc/fstab
     192.168.0.254:/data   /data  nfs \\
             auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0
     192.168.0.254:/u   /u  nfs \\
             auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0
     ```

   - mount the shared directories running the command "mount -a"

7.3. Synchronized users and groups creation

The SLURM operational users UIDs and groups GIDs must be synchronized across the cluster. They are "slurm" with "slurm" group and "munge" with group "munge". To obtain this result, these users could be created on the master and then on all nodes maintaining coherent UIDs and GIDs. **This must be done prior to install RPMs** which, if these users don't exist, would create them with random UID/GID pairs.

Example for the users/groups creation for slurm and munge:

```
sudo groupadd -g 134 munge
sudo useradd  -m -c "MUNGE Uid '127' " -d /var/lib/munge \\
              -u 127 -g munge  -s /sbin/nologin munge
sudo groupadd -g 64030 slurm
sudo useradd  -m -c "Slurm workload manager" -d /var/lib/slurm \\
              -u 51030 -g slurm  -s /bin/bash slurm
```

The submission users must be created both in the OS of each cluster node and in a SLURM group of users. A submission user "rossimario", for example,

can be created as member of the already created group "exanest" in the Operating System of all the nodes:

```
sudo useradd  -m -c "Submitter user" -d /u/rossimario \\
              -u 51031 -g exanest  -s /bin/bash slurm
```

The same user must be created in the SLURM cluster in the same group. This SLURM specific configuration is described in section 7.8.

7.4. Install and test munge on all nodes

MUNGE[1] is an authentication service for creating and validating credentials. It is designed to be highly scalable for use in an HPC cluster environment. It allows a process to authenticate the UID and GID of another local or remote process within a group of hosts having common users and groups. These hosts form a security realm that is defined by a shared cryptographic key. Clients within this security realm can create and validate credentials without the use of root privileges, reserved ports, or platform-specific methods.

We installed MUNGE for authentication. Steps:

1. Installation of needed packages

   ```
   $ sudo apt-get install libmunge-dev libmunge2 munge
   ```

2. All nodes in the cluster must have the same munge.key. MUNGE key distribution means to copy the munge.key file from master to nodes and to set the correct ownership and permissions.

   ```
   $ sudo scp /etc/munge/munge.key firefly@node01:/etc/munge/
   sudo chown munge: /etc/munge/munge.key
   sudo chmod 400 /etc/munge/munge.key
   ```

3. The right permissions of MUNGE interested files:

   ```
   /etc/munge/                0700
   /var/lib/munge/        0711
   /var/log/munge/        0700
   /var/run/munge/        0755
   ```

4. To enable and start the Munge daemon before to start the SLURM daemons

   ```
   $ sudo systemctl enable munge
   $ sudo systemctl start munge
   ```

Verify that MUNGE is correctly running doing:

```
munge -n | unmunge | grep STATUS
```

Expected output:

```
STATUS:           Success (0)
```

7.5. Installation and configuration of the accounting service

Slurm can be configured to collect accounting information for every job and job step executed. Accounting records can be written to a simple text file or a database. Information is available about both currently executing jobs and already terminated jobs. There are three distinct plugin types associated with resource accounting and they can be configured in *"slurm.conf"*.

7.5.1. Mysql database installation and configuration for record accounting

Installation, enable service at boot time and start of the service:

```
$ apt-get install mariadb-server
$ systemctl enable mysql
$ systemctl start mysql
```

To secure the database installation, run the following command and answer to the posed questions:

```
sudo mysql_secure_installation
```

Accounting database configuration:

```
$ mysql -u root
mysql > create database slurm_acct_db;
mysql > create user 'slurm'@'localhost';
mysql > CREATE USER 'slurm'@'node01.oats.inaf.it' IDENTIFIED BY 'XXX';
mysql > CREATE USER 'slurm'@'node02.oats.inaf.it' IDENTIFIED BY 'XXX';
mysql > set password for 'slurm'@'localhost' = password('XXX');
mysql > grant usage on *.* to 'slurm'@'localhost';
mysql > GRANT ALL ON slurm_acct_db.* TO 'slurm'@'node01.oats.inaf.it';
mysql > GRANT ALL ON slurm_acct_db.* TO 'slurm'@'node02.oats.inaf.it';
mysql > grant all privileges on slurm_acct_db.* to 'slurm'@'localhost';
mysql > flush privileges;
mysql > exit
```

7.5.2. slurmdbd accounting service installation and configuration on master node

Installation, enable service at boot time and start of the service:

```
$ sudo apt-get install slurmdbd
$ sudo systemctl enable slurmdbd
$ sudo systemctl start slurmdbd
```

slurmdb configuration file example:

```
$ cat /etc/slurm-llnl/slurmdbd.conf
AuthType=auth/munge
#AuthInfo=/var/run/munge/munge.socket.2
DbdHost=localhost
DebugLevel=4
#DbdBackupHost=localhost
StorageHost=localhost
StorageLoc=slurm_acct_db
StorageUser=slurm
StoragePass=XXX
StorageType=accounting_storage/mysql
#StorageUser=slurm
LogFile=/var/log/slurm-llnl/slurmdbd.log
PidFile=/var/run/slurm-llnl/slurmdbd.pid
SlurmUser=slurm
#ArchiveEvents=yes
#ArchiveJobs=yes
#ArchiveResv=yes
#ArchiveSteps=no
#ArchiveSuspend=no
\begin{Verbatim}
```

7.6. Install and configure SLURM on the master node

SLURM is included in the Ubuntu distribution, so the needed packages can be installed without configuring auxiliary packages.

1. Needed packages:

```
slurm-client SLURM client side commands
slurm-wlm  Simple Linux Utility for Resource Management
slurm-wlm-basic-plugins   SLURM basic plugins
slurmctld  SLURM central management daemon
```

Installation:

```
$ sudo apt-get install slurmctld
$ sudo apt-get install slurm-wlm slurm-wlm-basic-plugins
$ sudo apt-get install slurm-client
```

2. Creation of the SLURM spool and log directories and set of suitable ownerships and permissions:

```
$ mkdir /var/spool/slurm
$ chown -R slurm:slurm /var/spool/slurm
$ mkdir /var/log/slurm
$ chown -R slurm:slurm /var/log/slurm
$ mkdir /var/run/slurm
$ chown slurm:slurm /var/run/slurm
```

3. Creation of the SLURM configuration file. The file can be created customizing the default included in the distribution or exploiting the configurator form provided in the distribution, in our case

   `/usr/share/doc/slurmctld/slurm-wlm-configurator.easy.html`

   or exploiting the on-line available form:

   `https://slurm.schedmd.com/configurator.easy.html`

   an example of the file, with very few customization, but working, is in Appendix A

4. Creation of the startup script:

```
$ cat /lib/systemd/system/slurmctld.service
[Unit]
Description=Slurm controller daemon
After=network.target
ConditionPathExists=/etc/slurm/slurm.conf
[Service]
Type=forking
EnvironmentFile=/etc/default/slurmctld
ExecStart=/usr/sbin/slurmctld $SLURMCTLD_OPTIONS
```

```
PIDFile=/var/run/slurm-llnl/slurmctld.pid
[Install]
WantedBy=multi-user.target
```

Attention must be payed in **being coherent in the "PIDFile" variable content in "\lib\systemd\system\slurmctld.service" file and "SlurmctldPidFile" variable content in "\etc\slurm.conf"** file because, if they are not the same, the startup script does not work, giving a not clear error.

5. Enable the start of the service at boot time and start it:

```
$ sudo systemctl enable slurmctld
$ sudo systemctl enable slurmctld
```

7.7. Install SLURM and associated components on a compute node

1. Needed packages:

```
slurm-client SLURM client side commands
slurm-wlm  Simple Linux Utility for Resource Management
slurm-wlm-basic-plugins   SLURM basic plugins
slurmd  SLURM compute node daemon
\begin{enumerate}
```

Installation:

```
$ sudo apt-get install slurm-client
$ sudo apt-get install slurm-wlm slurm-wlm-basic-plugins
$ sudo apt-get install slurmd
```

2. Creation of the SLURM spool and log directories and set of suitable ownerships and permissions:

```
$ mkdir /var/spool/slurm
$ chown -R slurm:slurm /var/spool/slurm
$ mkdir /var/log/slurm
$ chown -R slurm:slurm /var/log/slurm
$ mkdir /var/run/slurm
$ chown slurm:slurm /var/run/slurm
```

3. The SLURM configuration file **"\etc\slurm.conf" must be exactly the same in all the nodes of the cluster**. It can be distributed scripting an "scp" operation or NFS mounting the directory containing

it. This second option can be useful for all the SLURM installation directory, so the software and configuration items are the same for all the nodes and there are not synchronizations issues. This way, however, could raise up the load and the network traffic. In a bigger cluster, a stand alone NFS server could be a partial solution of this problem.

4. Creation of the startup script:

```
$ cat /lib/systemd/system/slurmd.service
[Unit]
Description=Slurm node daemon
After=network.target
ConditionPathExists=/etc/slurm-llnl/slurm.conf
[Service]
Type=forking
EnvironmentFile=/etc/default/slurmd
ExecStart=/usr/sbin/slurmd $SLURMD_OPTIONS
PIDFile=/var/run/slurm-llnl/slurmd.pid
[Install]
WantedBy=multi-user.target
```

   Attention must be payed in **being coherent in the content of "PIDFile" variable content in "/lib/systemd/system/slurmd.service" file and "SlurmdPidFile" variable content in "/etc/slurm.conf"** because if they are not the same, the startup script does not work, giving a not clear error.

5. Enable the start of the service at boot time and start it:

```
$ sudo systemctl enable slurmd
$ sudo systemctl enable slurmd
```

7.8. Create initial slurm cluster, account, and user in the accounting database

At this point we are ready to create our cluster (the name is already configured in *"slurm.conf"* as *"ClusterName=mycluster"* registering it in the accounting system:

```
sacctmgr add cluster mycluster
```

Then we add a SLURM account, corresponding to the already existing OS group "exanest":

```
sudo sacctmgr add account exanest
```

and, at the end, we add our user, able to submit to SLURM, and belonging to the newly created group:

```
sudo sacctmgr add user rossimario defaultaccount=exanest
```

At this point, we can verify the configuration with the following commands:

```
sudo sacctmgr show cluster
sudo sacctmgr show user
```

Appendix A. slurm.conf file example

```
$ cat /etc/slurm-llnl/slurm.conf
ClusterName=mycluster
ControlMachine=masternode
ControlAddr=192.168.0.254
BackupController=masternode
BackupAddr=192.168.0.254
#
SlurmUser=slurm
#SlurmdUser=root
SlurmctldPort=6817
SlurmdPort=6818
AuthType=auth/munge
#JobCredentialPrivateKey=
#JobCredentialPublicCertificate=
StateSaveLocation=/var/spool/slurm/ctld
SlurmdSpoolDir=/var/spool/slurm/d
SwitchType=switch/none
#MpiDefault=none
MpiDefault=openmpi
SlurmctldPidFile=/var/run/slurm-llnl/slurmctld.pid
SlurmdPidFile=/var/run/slurm-llnl/slurmd.pid
ProctrackType=proctrack/pgid
#PluginDir=
#FirstJobId=
ReturnToService=0
#MaxJobCount=
#PlugStackConfig=
```

```
#PropagatePrioProcess=
#PropagateResourceLimits=
#PropagateResourceLimitsExcept=
#Prolog=
#Epilog=
#SrunProlog=
#SrunEpilog=
#TaskProlog=
#TaskEpilog=
#Using memory cgroups to restrict jobs to allocated memory
#resources requires setting kernel parameters
#
#$ vi /etc/default/grub
#GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
#$ update-grub
#$ reboot
#TaskPlugin=task/cgroup
#TrackWCKey=no
#TreeWidth=50
#TmpFS=
#UsePAM=
#
# TIMERS
SlurmctldTimeout=300
SlurmdTimeout=300
InactiveLimit=0
MinJobAge=300
KillWait=30
Waittime=0
#
# SCHEDULING
SchedulerType=sched/backfill
#SchedulerAuth=
#SelectType=select/linear
#FastSchedule=1
#PriorityType=priority/multifactor
#PriorityDecayHalfLife=14-0
#PriorityUsageResetPeriod=14-0
```

```
#PriorityWeightFairshare=100000
#PriorityWeightAge=1000
#PriorityWeightPartition=10000
#PriorityWeightJobSize=1000
#PriorityMaxAge=1-0
#
# LOGGING
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm/slurmd.log
# Job completion information is not maintained:
JobCompType=jobcomp/none
#JobCompLoc=
#
# ACCOUNTING
#JobAcctGatherType=jobacct_gather/linux
#JobAcctGatherFrequency=30
#
AccountingStorageType=accounting_storage/mysql
AccountingStorageHost=localhost
#AccountingStorageLoc=
AccountingStoragePass=XXXXXXXXX
AccountingStorageUser=slurm
#
# COMPUTE NODES
NodeName=node01 NodeAddr=192.168.0.1 CPUs=6 RealMemory=3811 \\
Sockets=2 CoresPerSocket=3 ThreadsPerCore=1 State=UNKNOWN
NodeName=node02 NodeAddr=192.168.0.2 CPUs=6 RealMemory=3811 \\
Sockets=2 CoresPerSocket=3 ThreadsPerCore=1 State=UNKNOWN
PartitionName=debug Nodes=ALL Default=YES MaxTime=UNLIMITED State=UP
```

References

[1] Munge. URL https://dun.github.io/munge/. [Last online accessed 31-July-2018].

[2] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkila, X. Wang, K. Hamily, and S. Bugoloni. Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment. *ArXiv e-prints*, September 2017.

[3] V. A. Cicirello. Design, Configuration, Implementation, and Performance of a Simple 32 Core Raspberry Pi Cluster. *ArXiv e-prints*, August 2017.

[4] D. Goz, L. Tornatore, S. Bertocco, and G. Taffoni. Inaf-oats technical report 223: Direct *N-body* code designed for heterogeneous platforms. doi: 10.20371/INAF/PUB/2018_00002. URL `https://www.ict.inaf.it/index.php/31-doi/94-2018-2`. [Last online accessed 27-July-2018].

[5] Sparsh Mittal and Jeffrey S. Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4):69:1–69:35, July 2015. ISSN 0360-0300. doi: 10.1145/2788396. URL `http://doi.acm.org/10.1145/2788396`.