INTERNAL REPORT

OAC's Computer Cluster Facilities: Basic Hardware and Software Configurations

Francesco Nasir

Report N. 34, released: 17/02/2014

Reviewers: A. Poddighe, I. Porceddu

OAC Osservatorio Astronomico di Cagliari

Contents

1 Introduction

The Astronomical Observatory of Cagliari has a good deal of computing resources amongst which an IBM[1] cluster that consists of 60 computing nodes, storage servers and front-ends which are used to login into the cluster.

Before continuing with this report it is interesting to understand what a cluster is and what it can be used for. Computer clusters may be used and implemented in many ways in order to achieve different goals. One might implement a common services cluster for web, database or storage applications: a web server cluster can assign queries to different nodes in order to load-balance requests and optimize response time. Clusters may be used for high availability or fail-over of a certain service by implementing many redundant nodes so that there is not a single point of failure and the services are still provided if some components are not available.
In addition, clusters may be implemented for computational purposes, as in this report, in order to distribute a scientific computation amongst many nodes reducing the runtime of a program: some common examples of applications are climate modeling, signal processing, astronomical and engineering simulations.
It is difficult to find an exact definition of computer cluster but some underlying characteristics are usually present [9]:

- a great number of identical machines or nodes (homogeneous cluster);
- machines connected to one or more dedicated networks;
- common resources shared amongst nodes (e.g, the home directory exported through a network file system like NFS or IBM's GPFS);
- nodes that "trust" one another so that SSH or RSH login does not require passwords;
- middleware between the operating system and the applications such as MPI in order to run code across many nodes making the whole cluster seem like one very big and powerful computer.

Let us look at some basic concepts regarding computer clusters and why they can be useful. Computer code can be executed sequentially on one CPU, or it can be executed in parallel by subdividing the program across many processing units [6]. In any case, the goal is to improve the runtime, which depends mainly on the execution time of the processing units and also on the communication time between units (i.e., the bus and network bandwidths). Considering only the former component, it is fair to assume that:

$$rt = n_i \times n_c \times t_c \tag{1}$$

where rt is runtime, n_i is the number of instructions in a program, n_c is the number of instructions executed every clock cycle and t_c it the clock cycle time. So as t_c decreased evermore, the runtime improves, this is called "frequency scaling". Nevertheless power p consumption increases as well:

$$p \propto f \tag{2}$$

where $f = 1/t_c$, so it is not possible to increase CPU frequency evermore, in addition to this, other problems such as transmission speeds, costs and limits to miniaturization are also present. To improve performance other methods, such as parallelization, must be used.
Parallelism regards both the underlying hardware support and software implementation. Parallel code is usually more complicated than sequential code, furthermore the performance of a

[1] Refer to appendix section N for meaning of various acronyms

program does not depend only on the underlying hardware (e.g., the number of CPUs) but also on the level of parallelization of the code itself: very few programs are completely parallel, a condition known as "embarrassingly parallel". Regarding this problem Admahal's and Gustafson's law indicate that the speed at which a program can be executed depends on the portion of non-parallel code in the program. As an example, the following is Admahal's law:

$$s(n_{pu}) = \frac{1}{\alpha + (1-\alpha)/n_{pu}} \tag{3}$$

where s is speed up and n_{pu} is the number of processing units (i.e., CPUs, cores) and α is the percentage of non-parallel or sequential code expressed in decimals. For a given α, there is a n_{pu} after which performance increases very slowly.

The main difficulty when implementing parallel algorithms is data dependencies: there might be some calculations performed by a subprogram, also referred to as "thread", that depend on the data output of another thread, in fact, no program can run more quickly than the longest chain of dependent calculations known as "critical path". Moreover, there are some conditions known as Bernstein's conditions that indicate when two threads are independent of each other (e.g., they do not share data between each other) and can run in parallel. If these conditions are not satisfied, flow dependency and data dependency is introduced and some parts of code will run sequentially. If two threads are not independent of each other they must communicate with each other, for example, if they share the same variable the first subprocess to access the variable must lock it in order to make it mutually exclusive, the lock is removed when the thread has finished using that variable. If both threads could access the same data simultaneously so called "race conditions" would be introduced and the program might not produce the correct output. So techniques such as semaphores, barriers and synchronization must be used when dealing with shared data. These techniques, amongst other things, are important for consistency: the result obtained by the parallel code must be the same that would be obtained if the code were run sequentially. On the other hand, too much communication overhead between threads might make the program slower than it would be if it were run sequentially, this is known as "parallel slow down". So before applying parallelization to a certain problem one must evaluate if it is worthwhile. From the above discussion we can state that parallel code classification may be based on the frequency with which subprocesses, that belong to a common application, communicate with each other.

Further classification can also be given by Flynn's taxonomy [6] where parallel applications are divided in: single instruction single data SISD, multiple instruction single data MISD, single instruction single data, single instruction multiple data SIMD and multiple instruction multiple data MIMD. SIMD is also called "data parallelization" and it indicates that the same instruction or thread is run many times on different processing units over different data sets, this type of parallelism is used often in signal processing. While in the case of MIMD different calculations or threads can be performed on different computational units over different data sets, this is a fully parallel implementation also known as task parallelization.

The hardware implementation of parallelization may be realized in many ways. On the same CPU, bit parallelization is introduced by increasing the word size of the processor (e.g., 32, 64 bit chip). Instruction parallelism or pipelining is obtained by implementing processor architectures such as RISC. In order to actually run subprograms simultaneously as in the case of data and task parallelism, many processing units are needed. In this case the main hardware issue depends on wether the memory is shared or not. Multi-core computing consist in having one processor with two or more Arithmetic Logic Unit ALU or cores (i.e., the part of the processor which performs the actual calculations), in this case registers, caches and main memory are shared. Also in the case of symmetric multiple processors SMP on one machine, main memory

is shared and communication between processors occurs through buses. In the case of distinct nodes, memory is not shared but distributed and communication occurs through a network.
A computer clusters is made up of many nodes and memory is distributed. A famous implementation is the Beowulf cluster [11] where identical commercial desktops are connected with a TCP/IP Ethernet local area network. A Beowulf cluster typically has one or more "master nodes" with two NICs, a public one for user login and a private one to communicate with the "slave nodes" on which the computations are actually performed.
Obviously on cluster nodes it is possible to have many CPUs, each of which may have many cores, so shared and distributed memory coexist and it is up to the middleware software (e.g., MPI, PVM) to take care of how memory and communication between processing units occurs. Recently GPGPUs, which are usually used for accelerated graphics, are also being used as additional processing units for scientific or engineering calculations.
An extension of cluster computing is grid computing. The former may be imagined as a cluster of clusters, communication between clusters occurs through a WAN, in some cases also through the Internet. Also in this case, middleware software is used to standardize interfaces between clusters and manage network resources (e.g., gLite, BOINC). A very famous application of grid computing is the SETI@home project.

When techniques, such as parallelization and clustered computers, are used to greatly speed up a program's runtime and performance, we are in the domain of High Performance Computing HPC or supercomputing, in this case, performances that range from Tera to even tens of Peta Floating Operations Per Second (FLOPS) can be obtained. It is fair to assume that the number of FLOPS is given roughly by:

$$flops = n_{pc} \times f \times flops_c \tag{4}$$

where n_{pc} if the number of processing units and $flops_c$ is the number of FLOPS per clock cycle. So for a single machine with 2 2.5GHz quad-core CPUs, supposing that each core manages roughly 4 floating operations per cycle, the performance would be $\simeq 0.1$ TFLOPS. The most powerful clusters in the world (e.g., IBM's Sequoia, Cray's Titan and the Chinese Tianhe2) can produce tens of PFLOPS.

This report starts by describing the underlying hardware and network topology and moves on to software and services configurations. The cluster will be described by using a bottom-up approach: illustrating firstly the hardware then the firmware, basic software configurations up to monitoring tools and the implementation of middleware. For every configuration there will be a generic description of the service in the relative section and a accurate software installation and configuration description in the corresponding appendix. Many acronyms will be used, in order to understand their meanings refer to appendix N.

2 Underlying Hardware

In fig.1, a front view of the cluster is visible. It is installed in three Emerson-Knurr racks: the first rack (left side) and the third one (third rack) contain computing nodes or slave nodes while the second rack (the middle one) contains front-end nodes, storage servers and the disk subsystem. On each rack there is a touch screen monitor through which it is possible to see front and

Figure 1: A front view of the cluster: racks, server and client nodes, switches and storage subsystem.

back rack temperatures and IP interface settings. It is not possible to change settings through the monitors.
Furthermore, it is possible to see that on the left hand side of each rack there is a black leaver and a green button, these are used to provide power to the computer systems which are installed inside the racks. In the following sections a lot more hardware detail will be provided.

To learn how to switch on and off the cluster see appendix A.

2.1 Network topology

As mentioned above and as can be seen in fig.2, the cluster is confined in one big multi-rack cabinet comprised of 3 racks, the first and the third rack contain the client nodes, also referred to as slave or computational nodes, while the second rack contains the front-end nodes, also referred to as master or login nodes. There are three private networks: the BMC network, the InfiniBand one and the GbE network. The first is used for low level hardware control: turning on or off machines, booting options and logging, the second is dedicated to storage: exporting the home directory, while the third is used for login and communication between nodes. There is also a backup GbE login network. In rack 2, it can be seen that one of the storage servers connects to the IBM DS4200 disk subsystem through fibre channel with a fibre optics cable. Logical drives are imported from the disk subsystem to the storage server and can only be seen

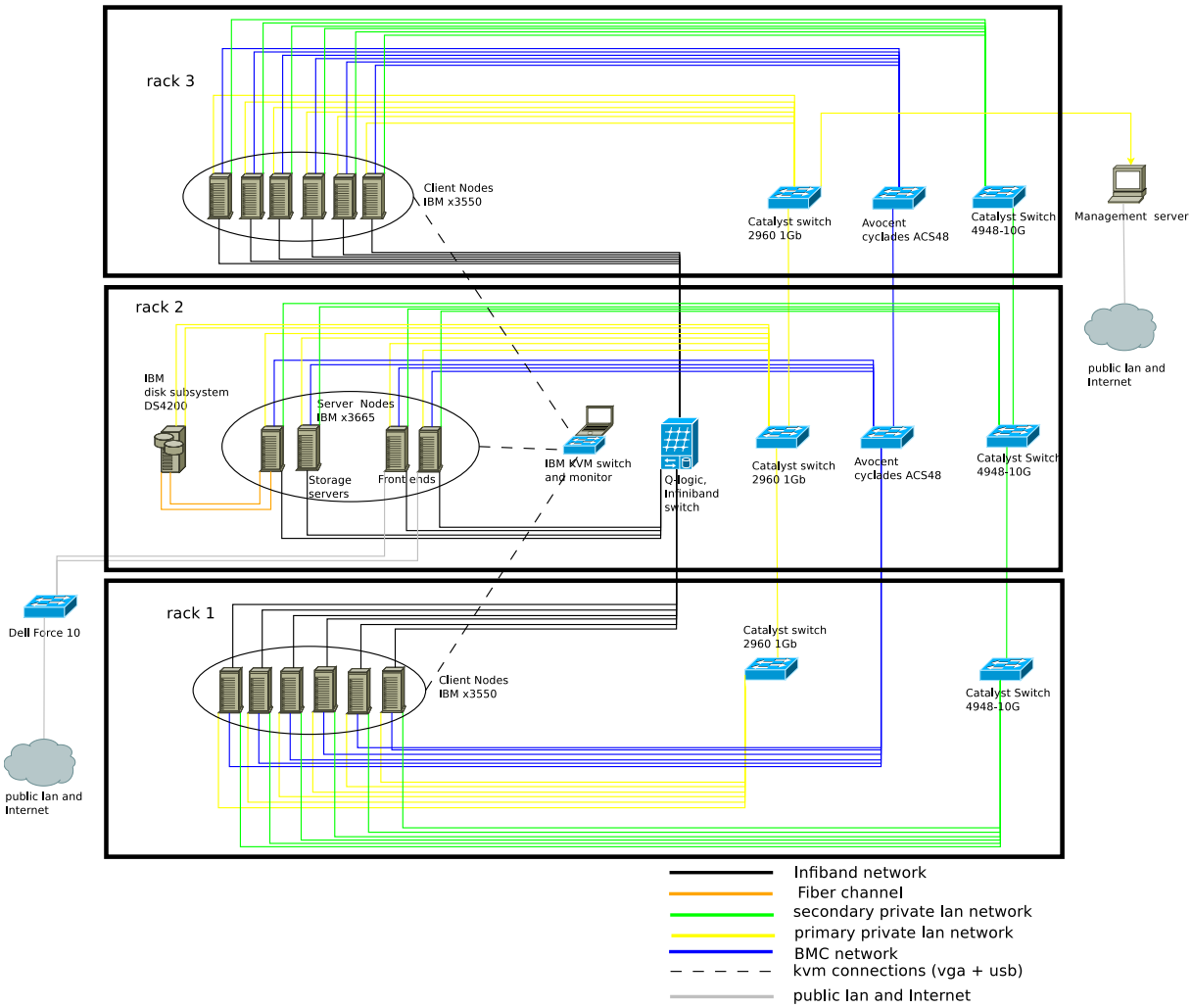


Figure 2: Network Topology of the cluster, for clarity not all client nodes are shown in rack 1 and 3.

by the latter. From the storage server, by using the general parallel file system GPFS, the disk drives are then associated to the home directory and exported through the InfiniBand network and the Qlogic switch to all the other nodes in the cluster.
The front-ends are connected to the three private networks mentioned above and also to the public local area network and the Internet. A user connects to his account on one of the front-ends and then he can launch and control his code from here, the actual computation is performed on the slave nodes.
The nodes are also connected to the keyboard video mouse KVM hardware which is located in rack two. The nodes are daisy-chained one to the other, while the eight node in the chain connects to the KVM switch, so each switch port corresponds to 8 nodes. The KVM monitor and keyboard are used to access each node locally.
Finally a management node is connected to the GbE network in order to manage the whole cluster. Services such as network booting, local repository, NAT and other services are provided by this server.

The true IP addresses of networks and hosts are not shown but for the purpose of understanding configurations we will suppose that:

- GbE login network : 192.168.1.0/24;

- Backup GbE login network : 192.168.2.0/24;
- BMC network : 192.168.3.0/24;
- Storage Infiniband network : 192.168.4.0/24;
- the last octet for client nodes in each network goes from 101 to 160;
- the last octet for front-end nodes in each network goes from 201 to 202;
- the last octet for storage nodes in each network goes from 203 to 204;

The switches used in this cluster configuration are the following (see fig.3):

- 3 Cisco Catalyst 2960-1G (login Ethernet network);
- 3 Cisco Catalyst 4948-10G (login Ethernet network);
- 2 Avocent Cyclades ACS48 (BMC network);
- 1 QLogic 9080 Sylverstorm Technologies (storage-InfiniBand network);
- 1 Dell Force10 (public LAN and Internet);

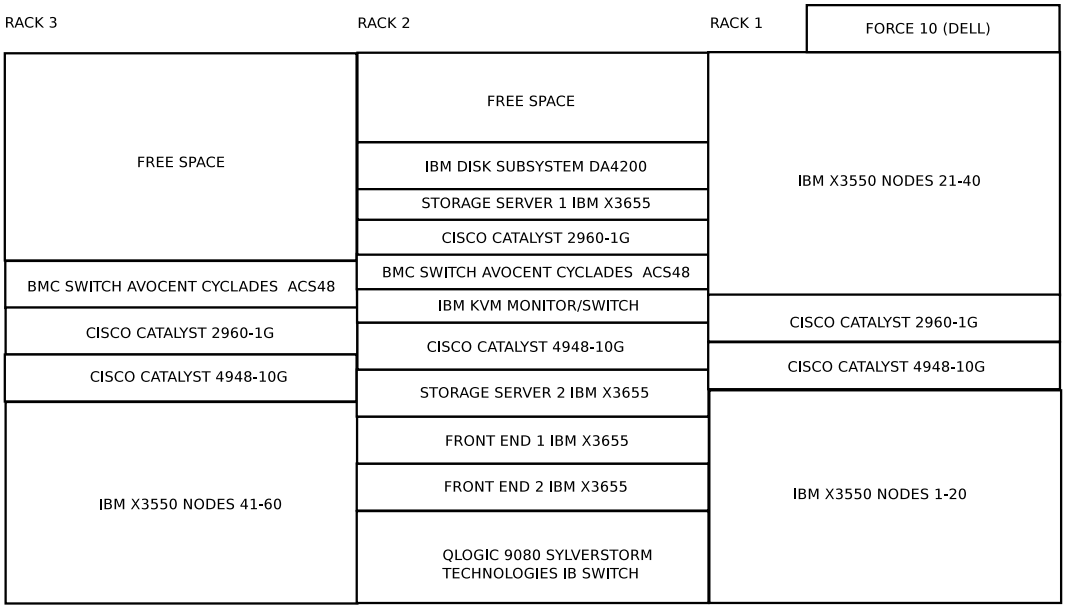


Figure 3: Schematic showing back view of the cluster and indicating collocation of nodes and switches.

Now we will go more into detail of each cluster hardware component.

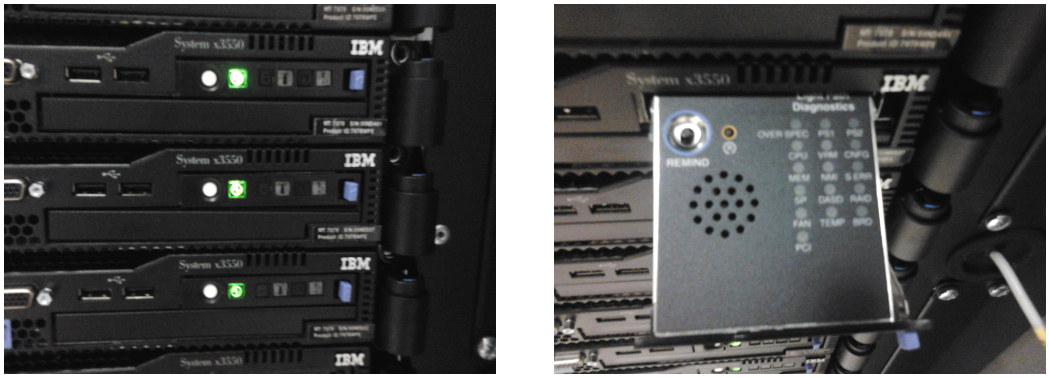
2.2 Client nodes

Hardware: the client nodes are IBM system X3550, with the following features:

- 2 64 bit 2.5GHz quadcore Intel processors;
- 4 4GB RAM modules DDR3 for a total of 16GB;
- 2 hard disks 250 GB SATA2;
- 2 1GbE Broadcom Nextreme III NICs;

- 1 10Gb Mellanox Technologies InfiniBand interface;
- 1 BMC serial interface;

Basic features: it is also worthwhile mentioning some basic hardware features and diagnostic tools present on the X3550 systems. In order to turn the system on and off one must press the white button as shown in fig. 4(a). From left to right: if the green light is off, the system is not connected to a power supply, if it blinks the system is powered but in stand by, while if it is green the operating system is up. The next light indicates hard disk access, for example it blinks during boot up. The next blue light is a system locater, it indicated with which machine we are interacting in order to distinguish it from the others. Next there is an orange light with an exclamation point, if this is on, it means that something is not working correctly.
In order to further investigate a problem it is possible to flip a small tray with the blue leaver to the right and check the exact problem: for example, a memory access problem, a power supply problem and so on will be indicated with an round orange light next to the current issue.



(a) front pannel (b) diagnostic tray

Figure 4: The IBM X3550 On/Off button and diagnostic lights and tray.

Boot sequence: the boot sequence starts up with the IBM BIOS, then control is given to the RAID controller in order to correctly find the drives present on the system, after this control is given back to the BIOS and then the operating system takes over.
The boot procedure goes through different steps, the first screen gives you the option to go to setup mode or BIOS settings (F1), to diagnostic mode (F2) and to boot mode (F12). If no button is pressed boot continues as normal and in the next screen the system loads the raid controller. At this stage, by pressing CTRL+A one can enter the RAID controller setup utility otherwise the system proceeds as normal. When the controller is loaded, it looks for raid arrays or disk drives present on the system. If it finds two arrays followed by the word "volumes" it means that the two drives have no raid configuration, they are just two stand alone HD which are interfaced by the raid controller. If RAID has been configured, one might see one raid array to which HD belongs according to various configurations (e.g., RAID1, RAID5). The system then displays the message "BIOS installed correctly" and the operating system starts loading.
When setup mode is entered, it is possible to perform all usual BIOS settings (e.g., change boot sequence, enable PIX), and in advanced settings it is also possible to configure the BMC (e.g., administrator, users and IP). The diagnostic and boot mode enable you to check logs and choose boot medium respectively.
As mentioned previously, by pressing CTRL+A during the booting of the controller kernel then it is possible to enter the RAID utility and configure RAID arrays. It is also possible to delete, create, edit new arrays.

For more details look at the IBM x3665 documentation [3].

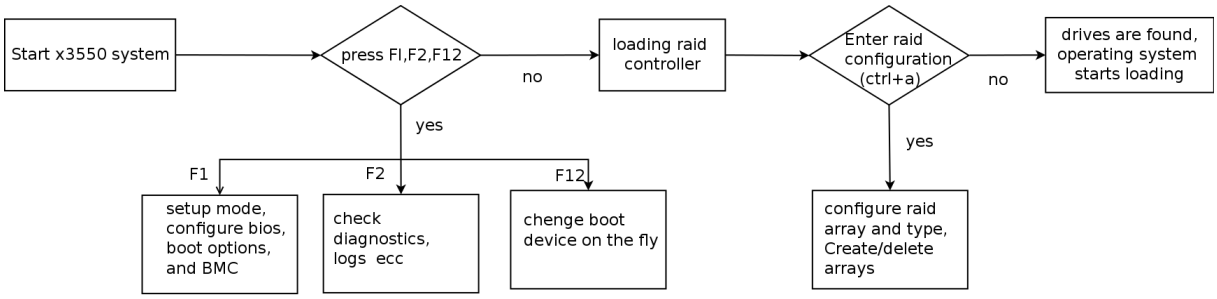


Figure 5: Flow chart that shows boot sequence and possible options of IBM system X3550.

2.3 Server nodes

Hardware: The server nodes (front-ends and storage) are IBM X3665 with the following features:

- 2 64 bit 2.5GHz quadcore Intel processors;
- 4 4GB RAM modules DDR3 for a total of 16GB;
- 2 hard disks 500 GB SATA2;
- 3 1GbE Broadcom Nextreme III NICs;
- 1 10Gb Mellanox Technologies InfiniBand interface;
- 1 BMC serial interface;
- Only for the storage server $n°1$: 1 Emulex fibre channel HBA with two ports;

Regarding the boot sequence and the basic features it is the same as the X3550 systems.

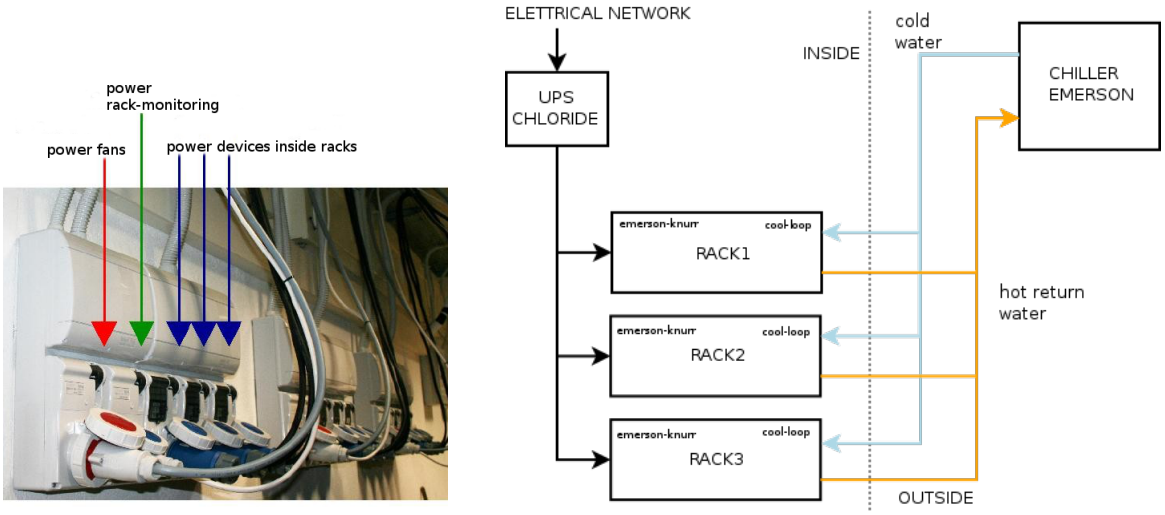
For more details look at the IBM x3665 documentation [2].

2.4 Chiller, coolLoop and racks

A cluster as the one described above can produce a lot of heat, especially during intensive CPU utilization. If the ambient, motherboard and CPU temperatures exceed recommended ones, this can cause permanent damage to the cluster nodes and/or it can represent a safety hazard.
In order for the temperatures to remain contained, the cluster is coupled to a cooling system which in turn comprises two subunits:

- the external Emerson "chiller" which pushes cooler water towards the racks and warmer water out of the racks through a system of pumps.
- The Emerson-Knurr "coolLoop" within the racks themselves. The racks are internally ventilated with fans that enable air circulation, also known as N-S ventilation. The air is cooled down by the contact with the chiller's cool water tubes.

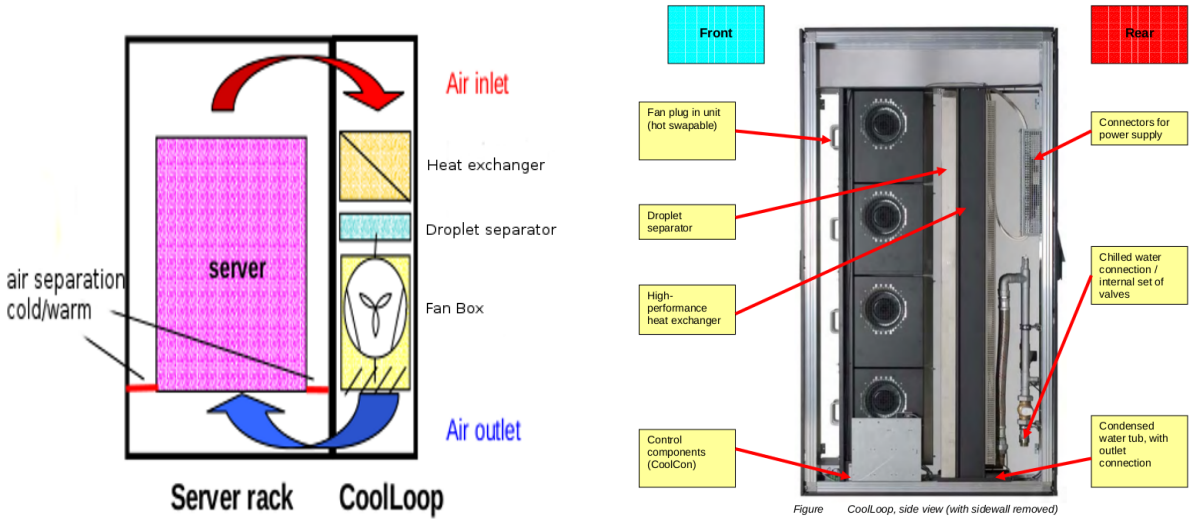
The chiller's outgoing/incoming temperatures and pressures must remain constant in order to provide correct cooling to the racks. It has been noted that the operational outgoing/incoming water temperatures are $15°C$ and $27°C$ respectively and outgoing/incoming water pressures are 3 bar and 1 bar respectively.
As can be seen in fig.6(a), the colder water flows towards the racks through a system of pumps.



(a) Emerson-Knurr racks and cool-loop (b) Rack hardware features

Figure 6: Picture and schematics showing (a) rack power supply connectors (b) rack electrical and hydraulic supply provided by the UPS and by the Emerson chiller respectively.

The presence of this cold source enables the fans in the racks (fig.6(b)) to blow cool air from the bottom front part of the rack towards the nodes. The air that has been heated by the nodes (e.g., $35°C$) is led through the laterally arranged wall openings or through the rear door to a special air/water heat exchanger. This heat is absorbed and the air is cooled (e.g. 20-$25°C$). If dew-point is reached and water droplets are produced, they are removed by the droplet separator. The cooled air is now blown again by speed-controlled fan boxes at the front of the rack. As the colder water in the pumps exchanges heat with the hot air it warms up too and recirculates back to the chiller in order to be cooled down again.



(a) Emerson-Knurr racks and cool-loop (b) Rack hardware features

Figure 7: The Emerson-Knurr coolLoop (a) schematics and (b) features of the cool-loop inside the racks.

As mentioned above, the system works so that it autoconfigures the speed of the fans in order to keep the temperatures constant. Temperature changes depend on the computational load of the nodes and/or on how many nodes are actually turned on. The fans start at 25% of their speed and increase proportionally to the rear temperature up to 100% at $40°C$. The cooling

capacity is also controlled according to the cold water flow: to $20°C$ at the air supply side the valve controls water flow between 0% and 100%.
Some system settings can be seen in the touch screen in the front top-left corner of the racks. Here settings like fan speed, front and rear temperatures and IP address settings can be observed and one can navigate through these setting with the touch screen. It is possible to note that the front temperature (incoming cool air) is always cooler that the rear one (return air temperature). It is not possible to change settings from the touch screen. In order to change the default settings it is possible to interact with the rack's control components, namely the CoolCon web-based software. In the current configuration one must proceed as follows:

- configure a laptop to be in the network 1.1.199.0/8;
- laptop must have Internet Explorer and Java;
- connect laptop to front top left RJ-45 connector in the racks: open cabinet on which the monitor is installed, at the top there is the RJ-45 connector (see fig. 1);
- enter IP 1.1.199.2-4 (according to which rack you want to connect to) into browser
- the coolCon software will start, you may enter as normal user (user: xxxx; password: xxxx) or administrator (user: xxxx ; password: xxxx); enter as admin in order to perform changes to settings;

In order to change fan speed which is directly related to the return air temperature one must go in the service subsection under temperature/humidity section and change the return air setting. The supply air setting will change the water valve flow which is directly related to the supply air temperature.

For more details look at the Emerson-Knurr documentation [7].

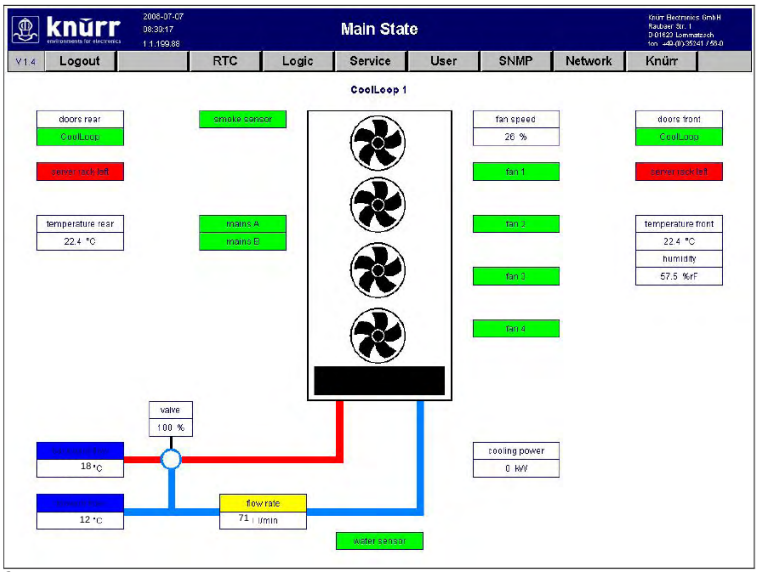


Figure 8: Emerson Knurr CoolLoop Software Interface

3 Storage

From here on, the server which manages storage and is directly attached to the disk subsystem will be referred to as the storage manager, the storage server or the host (fig.1).
As mentioned in the introduction, another important feature of a computing cluster is the possibility for the nodes to share resources (e.g. the /home directory). This is important because each node should be able to read/write to the same storage area as if it were local. This can be accomplished by using network file systems over the LAN or over a dedicated network like in the case of a SAN. In the first case, the network, usually Ethernet technology, is used both for communication between nodes and for storage access. In the second case, a separate storage network is implemented and technologies such as fibre channel and InfiniBand are used.
One could also define scratch areas on each computational node and use them in order to write the results of the computations, but this method will create differences between the nodes: some data will be stored on one node and other data on another node. Therefore it is harder to manage a cluster this way and it is more complicated for the user to remember where to retrieve the outputted data. It is better to have a common storage area (e.g., the /home directory) and save all the files and the computational results in an appropriate subdirectory.
Anyhow, the starting point is to configure a storage subsystem with many hard disks which are aggregated in RAID arrays to form logical drives. The disk subsystem is then connected to a storage manager through a high bandwidth connection like fibre channel. The storage manager will view the logical drives as if they were locally attached SCSI drives.
At this point it is possible to associate a file system to the logical drives and export them through a network file system such as NFS. When using GPFS it possible to aggregate more than one logical drive under a common file system, making the whole available space look as if it were a unique network shared disk which is then associated to the /home directory and exported to all the other nodes by using a GbE networks or preferably an InfiniBand network.
The practical steps are the following:

- The storage manager, sometimes referred to as the host, must be attached to the storage subsystem with two fibre channel connections, for the actual data transfer and a Ethernet connection for the management. Two fibre channel connections are needed for fail-over and flow-control.

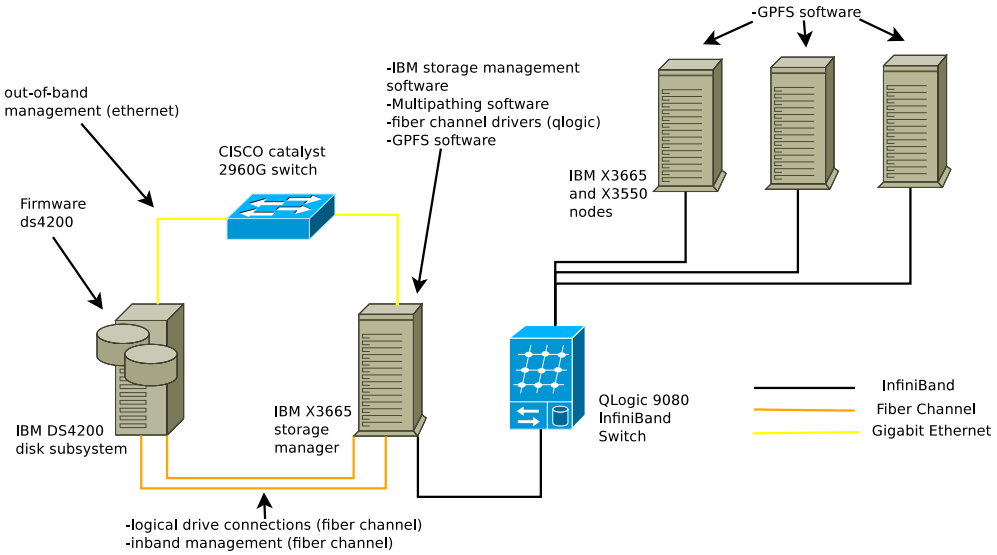
- The storage manager must have fibre channel HBA and corresponding drivers installed (e.g., QLogic's "qlxxxx") so that it can "see" the disks on the storage subsystem as if they were SCSI disks attached locally.

- The IBM System Manager Client software must also be installed on the storage manager.

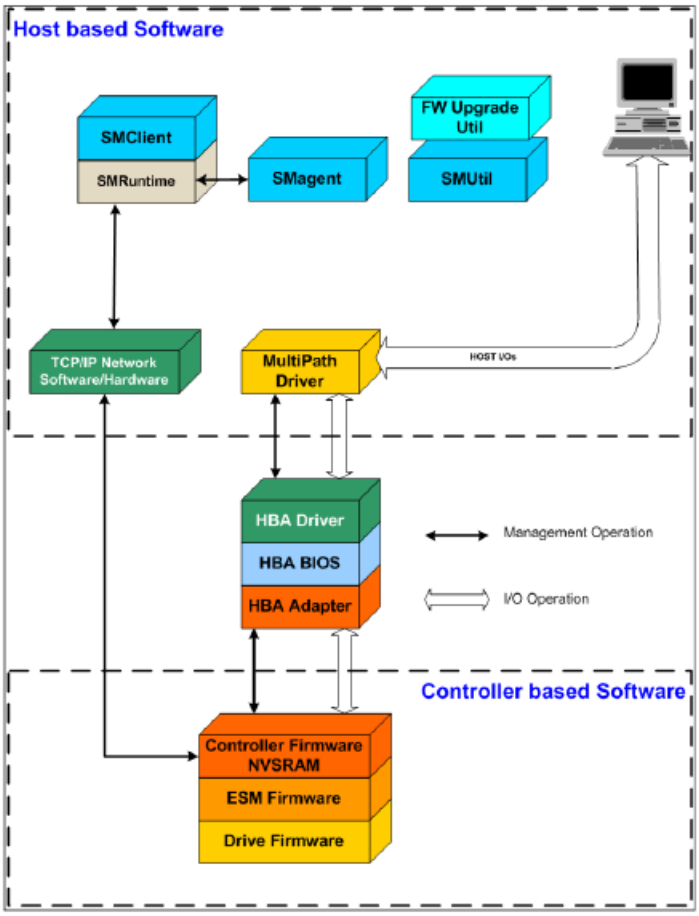
- Multipathing software (e.g., device-mapper-multipathing) must be installed on the storage server. This software is used in order for the operating system to understand that it is not attached to two sets of identical logical disks, instead the two fibre channel wires are connected to the same logical disks, therefor fail-over and fault control may be implemented.

- The storage manager configures logical drives on the disk subsystem through the System Manager GUI and subsequently the logical drives will be seen by the server itself.

- Now it is possible to install GPFS on the storage server and on all the other nodes and create a partition which comprises all the logical disks and associate the partition to the /home directory which is then exported by GPFS to all the other nodes through high speed InfiniBand connections.



(a) Diagram of the storage's physical configuration



(b) Diagram of the storage's software configuration, the storage manager is referred to as the host and the disk subsystem is referred to as the controller

Figure 9: Diagrams of the storage (a) physical configuration (b) software configuration.

3.1 Fibre channel and InfiniBand

Let us start by mentioning some of the technologies which are often used when implementing a SAN or a dedicated storage network and which were used in this report. First of, all fibre chan-

nel is the “de facto standard” in storage networking. It is described by a five layer stack which takes into account the more sophisticated upper level protocols for data control and mapping down to the physical layer protocols and standards for physical transmission. fibre channel is very popular because it can carry any type of data (video, audio, data) over long distances ($\approx$ 10km) and at very high speeds (2Gbps, 4Gbps or 8Gbps). It has low overhead, more over it is very popular for transporting upper level protocols such as the SCSI and the IP one. In fig.10 it can be seen how the SCSI protocol runs over the FC protocol stack. Optical cables comprise two connectors on each end which lead to two tightly linked thin cables, data flows serially in one direction or the other. Common connector types are LC and SC and physical transmission may occur in singlemode or multimode depending on how many optical frequencies are used. Servers must have some sort of HBA card which is attached to the motherboard through PCI express. The optical cable is then attached to the storage device and the HBA driver must be installed on the server. This is a point to point topology and because of the close relationship between fibre channel and SCSI it is as if the remote storage system is attached locally to the server through a SCSI connection.
fibre channel can also support an arbitrated topology, where many hosts are connected between each other forming a ring. More importantly, a fibre switched topology may implemented by using FC switches.
Just as Ethernet NICs have a burned in MAC address, fibre channel HBA have a 64 bit WWID or WWNN expressed in hexadecimal. Nevertheless hosts in a fibre channel switched environment do not communicate through the WWID but they send a FLOGI request to the fibre switch in order to get a logical 24 bit source ID S_ID. The S_ID comprises a domain, an area and a port portion. The domain is similar to the VLAN concept in Ethernet, the area is the switch port to which the host is attached to and the port is an ID that identifies the host port. Once the S_ID has been received, the host will send a PLOGI to the switch in order to map his new dynamically assigned address to his WWNN. This service provided by the switch is called fabric name service.
It is also possible to implement zoning on fibre channel switches (similar to VLAN creation on Ethernet switches) in order to create subdomains for fibre channel hosts on the same switch. Zoning can be based on host WWNN (soft zones) or on switch ports (hard zones).
Often more than one fibre channel cable is used to reach one storage device in order to account for fail-over and load-balancing. In this case multipathing software must be installed on the host, the multipathing driver will run on top of the SCSI layer so that the operating system may detect only one logical drive (fig. 10).
In this report fibre channel was used to attach the storage server to the disk subsystem making the storage server similar to a storage gateway for all the other nodes.

Infiniband was also used here to distribute the /home partition from the storage server to the other nodes. InfiniBand is a new technology and similarly to the TCP/IP stack it is characterized by a five level stack that goes from upper layer application services to the physical layer. The InfiniBand adapters are known as host channel adapters HCA and target channel adapters TCA. The HCA is directly attached to the host’s system controller and therefore to memory and it bypasses the PCI bus, although often PCI-X is used too. On the other hand TCA is usually placed on the storage device or disk subsystem. An important concept of InfiniBand is that devices on an InfiniBand Network communicate as if their main memories were directly attached, this concept is called remote direct memory access RDMA.
The IB connectors are made up by channels, for example a 4X IB connection means that it has 4 channels both in the outbound and inbound direction, each of which has a 2.5 Gbps bit rate. In addition to this, HCA can offload the processing from the main host processor, therefor latency is reduced.
In fig. 10, one can see how the SCSI protocol can be mapped to FC through the FCP mapping

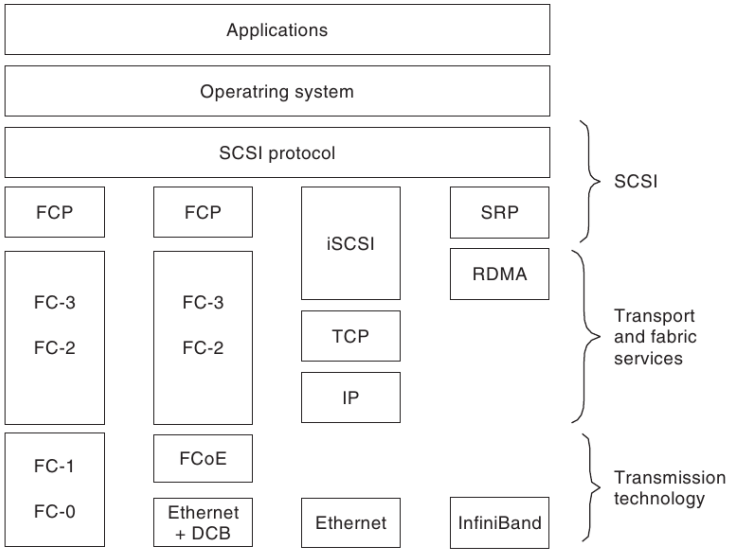


Figure 10: Layered diagram of communication protocols.

protocol or to IB through the SRP mapping protocol, furthermore communication protocols can be intermixed such as in the case of FCoE where FC can be carried over an Ethernet physical connection or vicecersa. In this report, IB was used at physical level, in the sense that an IP over IB protocol was used in order to carry TPC/IP information over an IB physical network, an IB switch was also used.

For more information on these topics see [12],[4].

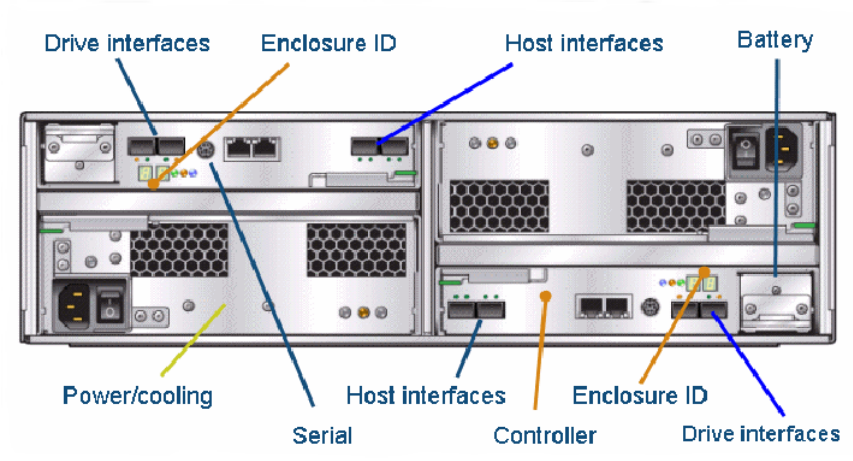
Table 1: Comparison between three main storage communication technologies.

	Ethernet	fibre Channel	InfiniBand
adaptors	NIC	HBA	HCA/TCA
connection distance	100 m	10 km	20 m
physical connection	fibre/copper	fibre/copper	fibre/copper
connection speed	1,10 Gbps	2,4,8 Gbps	2.5(1X),10(4X),30(12X) Gbps

3.2 IBM storage subsystem and the Storage Server

The disk subsystem DS4200 comprises 16 750GB SATA2 disks which are in turn managed by the subsystem controller so that RAID arrays and logical drives may be created. The disk subsystem has dual controller and power supply for load balancing and fail-over. As can be seen in

(a) front view



(b) back view schematics

Figure 11: The IBM disk subsystem DS4200 (a) front view (b) back view.

fig.11(b) there are two sets of controllers and power supplies. In each controller there are two fibre channel host ports in order to connect the storage server to the disk subsystem and two Ethernet ports in order for the storage server or some other server to manage the disk subsystem. There are also two fibre channel drive ports in order to extend the disk subsystem capacity attaching it to an expansion unit. First of all we must connect the IBM disk subsystem DS4200 to the storage server with two fibre channel connectors, the storage server has one HBA with two ports and the appropriate drivers installed accordingly to the hardware manufacture (e.g., "QLogic's qlxxxx"). The DS4200 must be connected to the storage server with two Ethernet cables for out-of-band management.
The next step is to install the "IBM system storage DS storage manager 10" software on the storage manager, in particular the SMClient component must be installed for out-of-band management (Ethernet), in the case of in-band management also SMAgent component must be installed. Once this is done, one can open the graphical interface in order to manage the subsystem with the Ethernet connection (fig. 9(b)). There are two basic windows: the Enterprise Management Window EMW to discover and select all possible IBM disk subsystems available and the Subsystem Management Window SMW used to manage a specific disk subsystem.
Once you have automatically discovered or manually added the disk subsystem (the default IP

address for the subsystem is usually 192.168.128.101-102) then you can start configuring arrays, logical drives and determine which hosts are allowed access to the logical drives. On the SMW you have basically 6 environments: summary, logical, physical, mappings and support; their functionalities are self explanatory.
It is time to create a logical drive and assign it to a host or a group of hosts. One must go to the "Logical" tab to view the unused raw capacity on the subsystem, the logical drive icon can be selected and configured through the wizard. First of all one must create a RAID array by choosing the RAID level and the number of physical drives that will be included. Once the RAID array has been created you can "carve" out of the array logical drives, to do so, select the array and label the logical drive. Now the LUN corresponding to the new logical drive must be mapped to the host or server that will be able to access that logical drive.
In the case of this report, the 16 750GB drives were grouped in RAID arrays of 4+1 disks each using RAID level 5 which supports the failure of one disk. Consequently a logical drive of roughly the same raw capacity was carved out of each RAID array, therefore three logical drives are present. One disk in the DS4200 is kept as hot swappable in the case that any other disk fails and this accounts for the 16 disks present in the DS4200 subsystem.
Some further considerations include setting a password in order to manage the subsystem on the SMW. In addition to this it is important to set controller ownership: determine which controller is the primary controller and which is the back-up one.

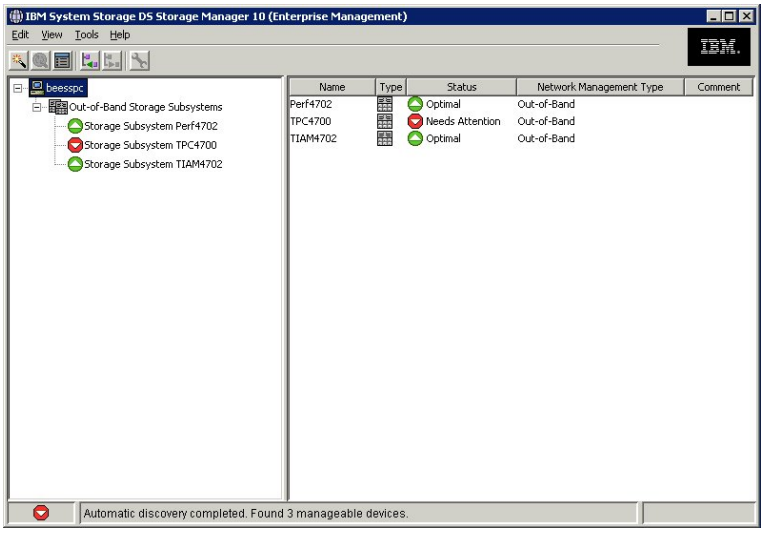


Figure 12: Storage Management Software.

Finally it is interesting to note that on the EMW it is possible to monitor the status of any discovered subsystem. If the subsystem is in healthy state then the status notification is green an no intervention is needed, if the subsystem needs attention then there will be an orange or red light according to the level of danger. In the latter case, something is wrong and it must be solved by using "Recovery Guru", which is the recovery wizard.

On the storage or metadata server we must install the drivers related to the fibre channel HBA, the Storage Manager 10 software components, and the multipathing drivers. Regarding the latter, if such driver is not installed and configured, the storage server will see two sets of drives of the same capacity. For example if we have configured on the disk subsystem 3 logical drives the operating system on the storage server will detect 6 logical drives because it has two fibre channel connections to the same subsystem. In order for the storage server's operating system to access the same LUNs via multiple paths, a multipathing driver is needed. This is useful also to implement load-balancing and fail-over. The multipathing layer sits above the FC

protocol, and determines whether or not the devices discovered on the target, represent separate devices or whether they are just separate paths to the same device. On Redhat based system the components used for this type of application are usually:

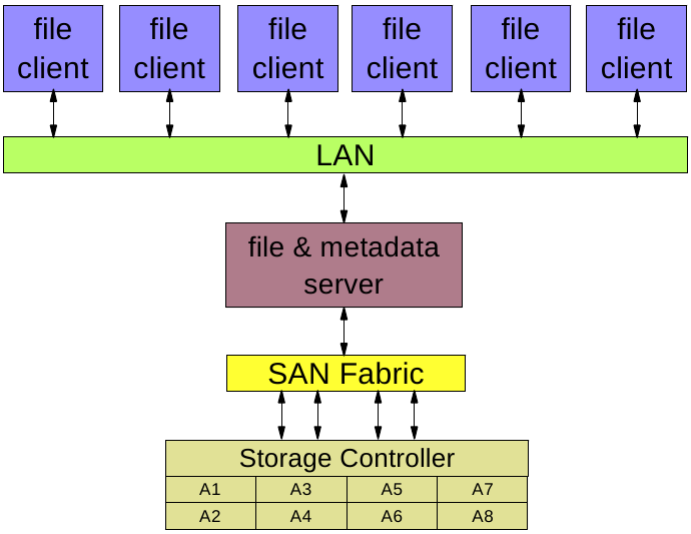
- device-mapper is the kernelspace device mapper DM which implements multipathing;
- multipath-tool is the userspace package.

To determine which SCSI devices/paths correspond to the same LUN, the DM initiates a SCSI Inquiry. The inquiry response, among other things, carries the LUN serial number. Regardless of the number paths a LUN is associated with, the serial number for the LUN will always be the same. This is how multipathing SW determines which and how many paths are associated with each LUN. Usually on Redhat based systems the package "device-mapper-multipath" comprises user and kernel modules.
If you are using a storage subsystem that is automatically detected, no further configuration of the multipath-tools is required. Otherwise you need to create /etc/multipath.conf and add an appropriate device entry for your storage subsystem.
Once the system is restarted and the multipath daemon "multipathd" is running you will see on the storage server three logical drives usually indicated with the dm-x nomenclature.

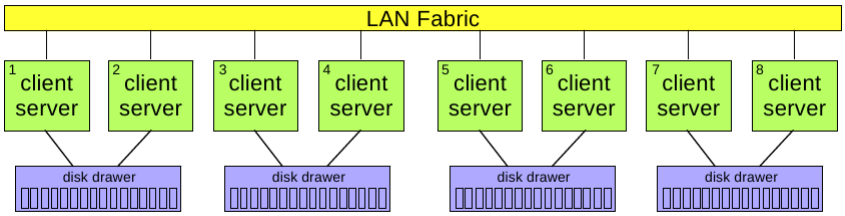
For more information see [10] and [4].

3.3 General Parallel File System

The storage server can now see the logical drives (e.g. dm-0, dm-1) as if they were locally attached. It is time to use a network file system, in this case GPFS, to aggregate the logical drives in order to form one big network shared disk under a common file system, which is then associated to a particular partition, usually the /home directory, and is then exported to all the other nodes.
The General Parallel File System GPFS is named this way for the following reasons: it is general in the sense that is supports a wide range of applications and configurations. It is parallel because data and metadata flows between all nodes and all disks in parallel. It is a file system because it maps and determines how data is written and retrieved from a storage media. As mentioned above, the objective is to export a common network shared disk to other nodes on the same network. In Unix/Linux systems NFS is often used for this task, nevertheless GPFS offers some added features, above all the fact that each node that belongs to the GPFS cluster may be a storage client and server simultaneously and the network shared disk may be distributed across many nodes so that no bottlenecks are created. Secondly file access is parallel and it is performed through GPFS multithreading capability: each block is accessed by a different thread. The GPFS daemon "mmfsd" runs symmetrically on each node. With GPFS it is also possible to aggregate many logical drives under one file system, in other words one unique storage partition may be created out of many logical drives.
In actuality, we implemented GPFS with respect to a client/server architecture similar to the one shown in fig. 13(a), but instead of a Ethernet LAN we have a dedicated storage InifiniBand network. The architecture in fig.13(b), is expensive to implement because each node must have a HBA, it is also harder to manage. Nevertheless, even under a client/server architecture, one can still use GPFS' managing utilities, which are designed for a clustered environment, and also its multithreading and parallel functionalities.
In GPFS one must define the management and quorum node, the rest are client nodes. Usually the management node is the one on which the GPFS cluster is defined and implemented, while the quorum node is usually the one directly attached to the storage subsystem. These two roles may coincide, and there may be redundant management nodes. It is important to note that for a GPFS cluster to work the management nodes must be up and running and at least half+1 of



(a) NFS client/server architecture



(b) GPFS SAN architecture

Figure 13: Comparison between different network file system typical architectures/topologies.

the quorum nodes must be up and running. Furthermore it is important to remember that one node may belong to only one GPFS cluster at a time.
For more information on GPFS theory see [1], for information on how to configure GPFS see appendix B.

4 Basic configuration of cluster

When the hardware configurations have been dealt with, the cluster is inserted in racks which provide sufficient cooling, the network connections have been performed, BIOS has been configured, a common storage area has been defined, then it is possible to go to the next level. The operating systems must be installed on the nodes and basic services must be provided.

4.1 Setting up a management server

It is useful to have a dedicated management server that deals with all the services that will be provided to the client and server nodes. The management node in our case is connected to the cluster via the 1 GbE LAN as can be seen in fig. 2.
The management node has the same operating system as the nodes: Scientific Linux 6.2 (https://www.scientificlinux.org/), it has two 1 GbE cards: one for the cluster private LAN and one attached to the public LAN and the Internet.
One of the most important services, that the management server must provide, is the possibility of installing and configuring the nodes' operating system remotely, via network, and configuring it automatically via kickstart. To enable the former service, we must first install a DHCP, a TFTP, a NFS server on the management node. It is important to gain all the MAC addresses of the cluster nodes in order to implement static DHCP between MAC addresses and IPs that we want to assign. In addition to this we must create a directory on the NFS exported directory that contains the contents of a specific operating system ISO image, go to appendix C for details.
Finally, when the operating systems have been installed, the management server can provide many useful services to the nodes, such as a local package repository, a DNS server, a gateway to the Internet and so on.

4.2 Network booting: pxe-dhcp-tftp-nfs

Manually installing an operating system on many systems is difficult and time consuming, besides it is hard to configure and install the OS in the exact same way on every node. An automatic procedure must be used: firstly the nodes must be able to install the OS remotely from the management server, secondly there must be a way for the installation and configuration procedure to occur automatically.
In order to network boot the nodes the first step is to configure their BIOS so that Preboot Execution Environment PXE booting is possible, in other words, one or more Ethernet cards with PXE functionality present on the node, must have the ability of searching for the bootloader and other boot mediums on a server on the network. On the management node a DHCP, TFTP and a NFS server must be configured, in alternative to the NFS server, one can use also FTP or HTTP. For security and higher control, static DHCP is implemented: the nodes' MAC addresses are statically associated to the node's IP addresses.
After the node is started, the control passes from the BIOS to the PXE enabled NIC. The first thing that PXE does is search for a DHCP server in order to acquire an IP addresses and use the network itself (fig. 15). The DHCP must also be able to indicate where the bootloader (pxelinux.0) is found. Usually the bootloader is under a specific TFTP server directory on the management server itself. Associated to the bootloader, there is a bootloader file, which is usually named "default", it may also be named with the IP address in hexadecimal assigned by the DHCP to the node. This file tells the system where to find the kernel and the ramdisk image, which are also stored on the TFTP server. The bootloader file may also specify the static IP address that must be assigned to the image or it may tell the kernel to perform a new search for a DHCP server in order to get a dynamically assigned IP address. The bootloader file can also tell the kernel if and where to load the complete operating system and whether the installation is automatic or manual.

Supposing the kernel image has been loaded successfully in the node's ram and it has also obtained an IP address (usually the same as the one used for PXE booting), now the kernel image looks for the NFS exported folder where the ISO image of the wanted distribution has been extracted. The kernel knows where to look due to the information written in the bootloader file. At this point it starts loading the complete operating system and if no automatic method is specified in the bootloader file then it needs human interaction else it performs the OS installation by itself and it may also perform pre and post installation configurations such as editing configuration files, adding users and shutting on or off services. In fig.15, the network booting

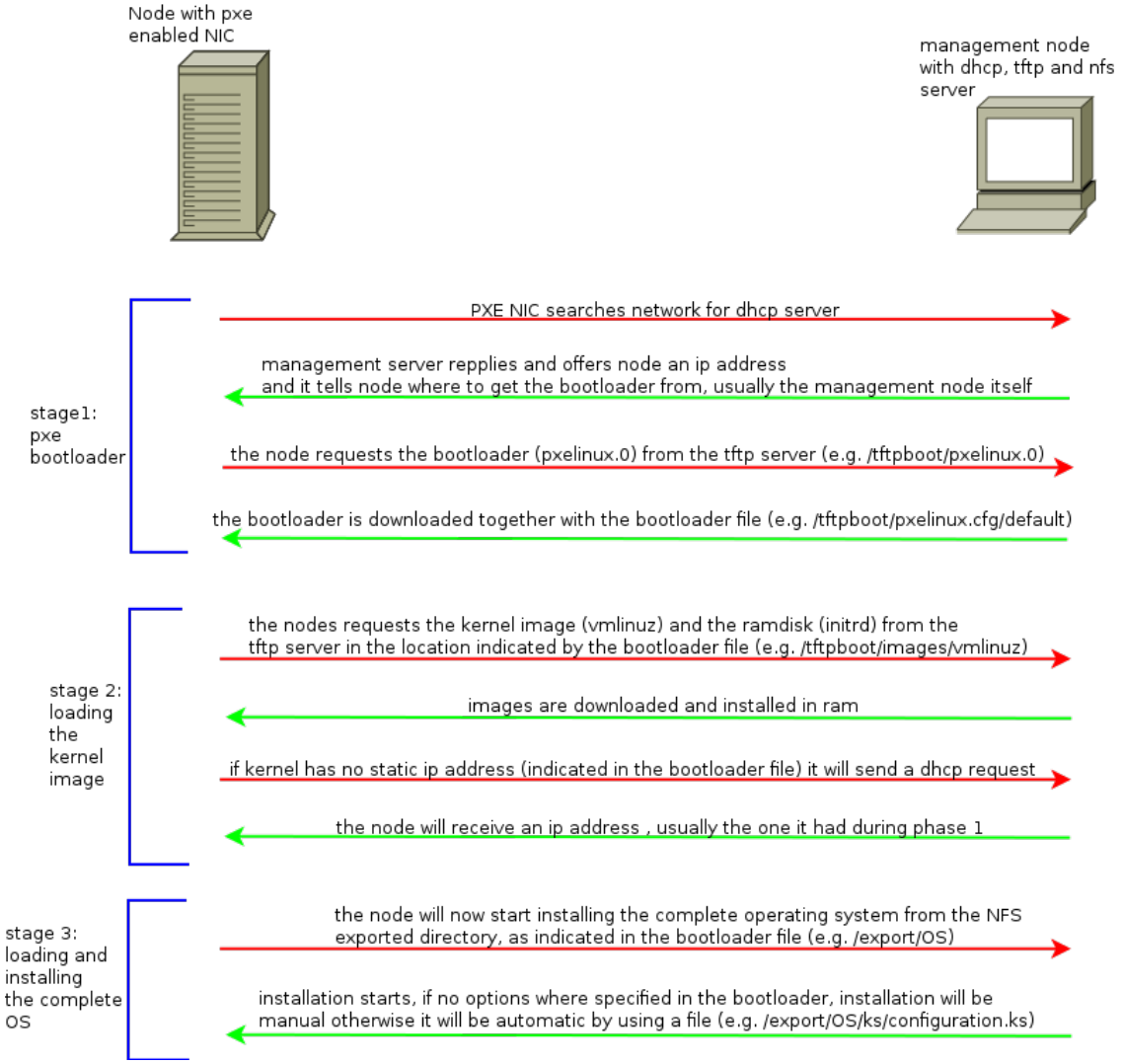


Figure 14: Network booting procedure using PXE enabled NIC on client node and DHCP, TFTP and NFS services on the management node.

procedure is shown, the last stage depends on the options specified in the bootloader file: we must decide whether to install the OS manually or automatically. For a cluster environment the latter option is mandatory in order to speed up cluster configuration greatly. For further details go to the appendix D.

4.3 Automatic installation with kickstart

There are many methods used for automatically installing an operating system, the one chosen here is the Kickstart method. This simply means that the installation is performed automatically

following the directives written in a kickstart script file. As mentioned before, the kernel knows that it will use this method and where to look for the file due to the information written in the bootloader file in the TFTP server.
Usually Red-Hat based operating systems such as Scientific Linux and CentOS have a utility called Anaconda which upon installation creates a kickstart file in which the current installation instructions are written, this file is saved in the /root directory as "anaconda-ks.cfg". One may use this file as a baseline and edit it for more complex automatic installations.
The basic instructions in the kickstart file regard: boot media, language, keyboard layout, network parameters, root password, timezone, partitions, packages that must be installed and so on. A fine grained installation is therefor possible; in summary, the first section of the kickstart file regards basic configuration, the second part regards partitions, the third regards packages that will be installed. In this part it is possible to install package suites or single packages.
At the end of the kickstart file one can edit a post installation section which takes place after the basic installation. In the post sections the kickstart file is just like a script file. It is possible to indicate what services to start and stop, set the default run level, configure the repository files. One can also configure the network interfaces: both Ethernet and also other technologies such as InfiniBand. You may mount a directory containing configuration files from a NFS server and copy them to the appropriate locations on the node. It is possible to install software from the mounted directory too, such as packages which are not readily available from the ISO distribution.
In other words, anything that can be done with a bash script may be also performed in the post section of a kickstart file. In appendix E there will be an example of a custom kickstart file. It is important to state that the kickstart configuration for the client nodes will probably be

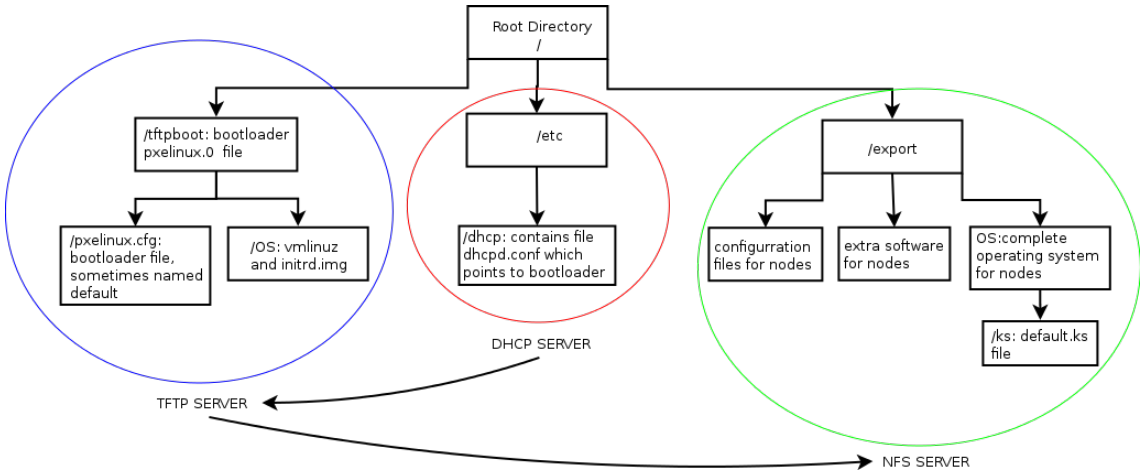


Figure 15: A diagram showing where folders and files of the various services are kept, including the kickstart file. The figure also shows how various service point to one another during network booting and installation of an operating system.

different from the storage and front-end nodes, the operating system on these nodes can even be installed manually or in alternative a different kickstart file can be used for client, storage and front-end nodes.

4.4 Secure shell host and key exchange

So now the operating systems are installed and configured on all the client nodes in exactly the same manner, the only thing that distinguishes a node from another is their IP addresses and their host names.
As stated in section 1, a key element when setting up a computer cluster is the fact that all the

nodes “trust each other” so that when protocols such as SSH or RSH are used to login from one node to another, or they are used by other services such as MPI or GPFS, passwords are not required. In this report the SSH application layer protocol was used.
Before we continue let us take a step back and talk about how the SSH protocol works, in particular let us address the steps taken by SSH to enable a secure tunnel between two hosts. When a user on a SSH client initially contacts a SSH server, the server will authenticate itself by sending its public key, moreover his digital certificate, which is usually located in the /etc/ssh directory, to the client. The user on the client will be prompted with a warning message to accept it only if he is sure that the server is who he says to be and he will store it permanently under his home in the ./ssh directory in the file “known_hosts”.
The next step occurs in the background: the client and the server initiate a Diffie-Helman DH procedure in order to safely produce the same symmetric shared key between each other that will be used for encryption. Data integrity will also be guaranteed by applying hash tags to the data itself and then encrypting everything.
Next, the client will be challenged by the server in order to authenticate itself. Basically the user on the client must input his password and the server checks that this password corresponds to the one stored locally for that user. More specifically, the client hashes the password using a hashing algorithm (e.g. MD5 or SHA) and sends it to the server who confronts it with the hash of the same password stored locally. If the two hashes match, then the user on the client is authenticated on the server.
A SSH tunnel is now up between the client and the server and the two can communicate by using an encrypted flux of data.
An alternative way to enable the user on the client to authenticate on the server may be signature-based. In this case the user on the client generates a public-private key pair, usually placed in the /home/user/.ssh directory. Now he will append his public key in a file named “authorized_key" in the /home/user/.ssh directory on the server he wishes to connect to. When the client starts an SSH session, the server will challenge the client by encrypting some random data with the client’s public key, he will send the encrypted random data to the client, who will decrypt it with his private key, now the client will create a hash of this random data like in the password-based case and send it to the server, the server will hash the random data and check that the two hashes match, if this is so than user authentication is completed successfully.
In the case of a computer cluster which is inserted in a private LAN, nodes are both SSH servers and clients simultaneously because processes on each node must be able to access other nodes and exchange data with them. In addition to this we must use a signature-based user authentication in order to remove the need of manually inserting passwords. Firstly all servers must be authenticated by all clients so that no warning prompt is issued upon login, this can be done automatically by writing a script or even manually. Secondly it is possible to produce one private and public key-pair and use it for every node by copying it in all the /home/user/.ssh folder. Now each node will be able to access the other without being prompted neither for server authentication nor for passwords (client authentication).
One weak point in the SSH login sequence is the initial step, where the client accepts that the signature sent by the server is authentic. In our case we are using a private network which is isolated from the Internet so this is not a big issue but on a public network one would need the aid of a third party certification authority in order to safely complete this step. In fig. 16, we can see the event sequence of a SSH connection both in the password case and in the signature-based passwordless case.
In summary, server authentication on the client is obtained by initially accepting the server’s public key. The authentication of the user on the server is obtained by inserting a password when prompted or by copying the user generated client’s public key on the server. Assymetric key-pairs are usually used for authentication, while symmetric keys are usually used for encryption.

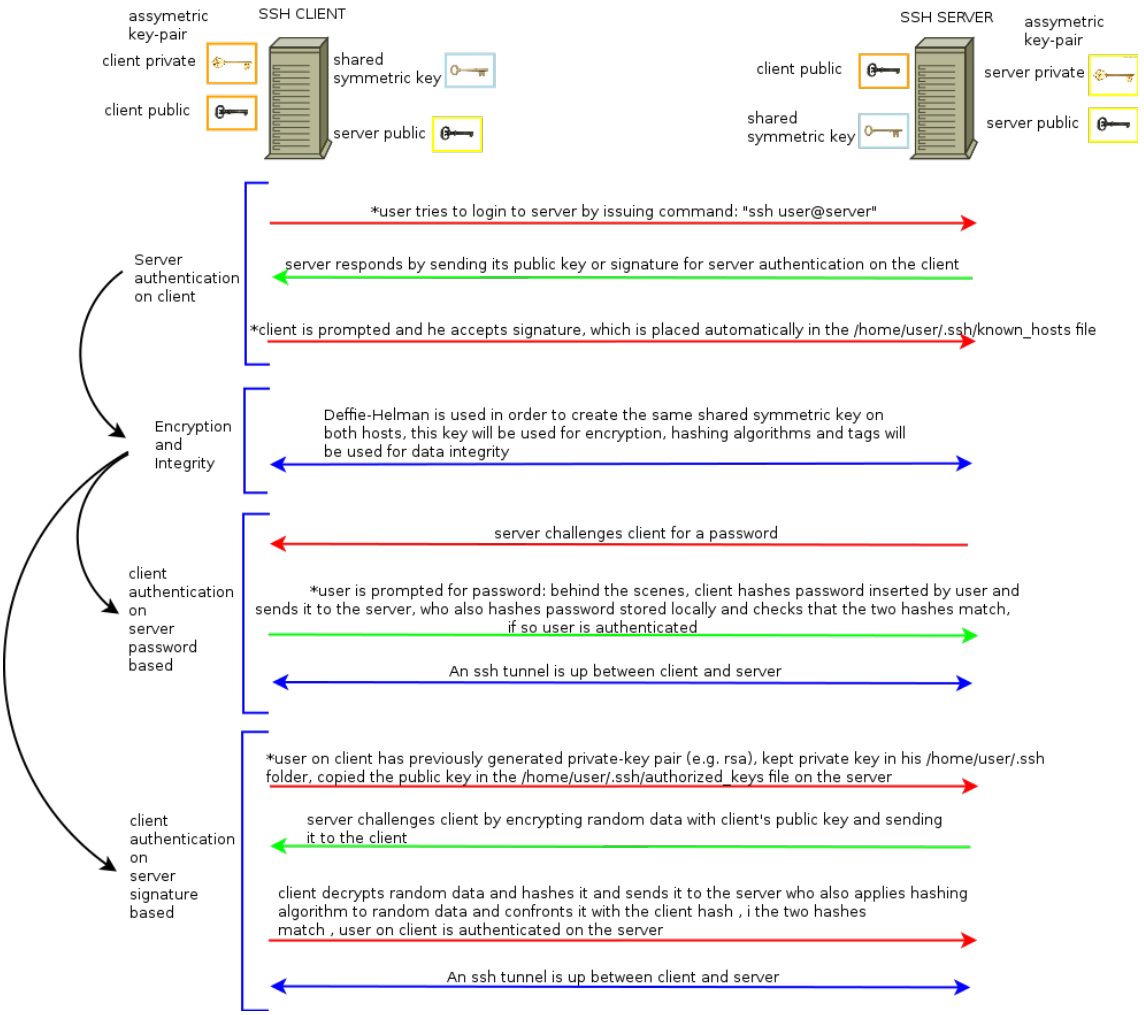


Figure 16: Event sequence of a SSH connection, the actions indicated with a preceding "" mean that human interaction is necessary otherwise actions are performed in the background by the applications.*

At this stage the cluster configuration is nearly complete: we have a great number of homogeneous nodes interconnected by a LAN and a dedicated storage network. Shared resources (i.e., the /home directory) are provided through the GPFS application and the nodes now "trust each other" through SSH key exchange. Before proceeding, let us explore some additional useful services that the management node can provide to the cluster nodes.

4.5 Local mirror

In section 4.3, we have described the automatic installation and post configuration of the operating systems on the cluster's nodes. At this stage, most of the necessary software will be installed on each node, but there will always be some software or application that will have to be installed at a later stage.
In order to accomplish this task it is possible to install software on the cluster's nodes directly from a Linux repository on the Internet. In this case the management node must serve as a gateway and a NAT server for the nodes (see section 4.8).
Alternatively it is more practical to create a local repository on the management node that mirrors periodically an official Linux repository on the Internet. This is useful because if too many nodes that simultaneously download from the Internet might saturate the Internet connection bandwidth, furthermore if the connection is slow or not available it will not be possible to install new software on the nodes. If a local repository is available, the nodes will get their updates or new software directly from the management node. To set up a local mirror we must create script on the management node that downloads the Linux packages from a official Linux mirror. The data transfer can be performed with rsync. In addition we must enable an FTP or HTTP server on the management node so that nodes can download packages from here. On the cluster nodes we must insert the local repository IP address in the appropriate repository file.
In summary these are the following steps:

- create script on management node that downloads packages from official Linux repository, rsync or ftp may be used;
- the script will be inserted in a cron file in order to be run periodically so that it is up to date;
- create an ftp or http server on the management node so that it can export packages to the cluster nodes;
- the nodes' list of package repository must be changed so that it point only to the local repository on the management node.

4.6 Domain name server

On a network, computers communicate and locate each other by using IP addresses, nevertheless for a user it is simpler to assign to each node a hostname that corresponds to their IP address. When we refer to a node through its hostname, either to connect to it or for other purposes there must be a file or a service that converts the hostname to the IP address.
Name resolution between the hostname ("namespace") and the corresponding IP address ("address space") on a Linux system is obtained mainly in two ways: either by modifying the /etc/hosts file or by using a Domain Name Service DNS server. In the first case each node must have the same /etc/hosts file in which every cluster hostname is resolved with respect to their IP address. This is simple to do, but if nodes are added or removed or if hostnames are changed on the cluster then one must modify every single file on each host. Configuring a DNS on the management node is more scalable. In this case, one must modify only the DNS' file.
Usually a domain name is represented by a string of characters separated by dots. Differently from IP addresses where the network portion is on the left side and the host portion is on the right, in domain names the more generic information is on the right, this represents the domain under which your host or server may be found, while the actual hostname is on the left. For example, in the fully qualified domain name: "myhost.mydomain.com", ".com" represents the top level generic domain while "mydomain" is the sub-domain under which the hosts on the specific network are located and "myhost" is the actual hostname.

No single DNS server maintains a database with all the hostnames in the world, instead a hierarchical model is used: the name space is divided in zones under which one or more domain and even sub-domains are managed by one authority, one company or institute with a related authoritative DNS server. By authoritative DNS it is meant that the server has the actual translations for a certain hostname and can reply to a DNS request. Alternatively, the DNS server delegates requests to an other DNS server that might know the answer.
Therefore name space is divided logically in top level domains under which many hosts or subdomains may be present, which may belong to different zones managed by different DNS servers.
The DNS resolution process is a client-server service: the client sends a DNS request to a specific server (UDP port 53 is usually used), if the DNS server is authoritative for that domain name it will send a reply else it will delegate the request to another DNS server. This process is usually recursive, the search is delegated to a top level domain server and is handed down to lower level domains servers until the answer is found. Sometimes DNS servers cache the answer so that this process must not be repeated every time.
In our case the DNS server maintains information only on the cluster's node names, and uses a private domain that is not available on the public network, if a node requests other additional hostnames then the request is delegated to a secondary DNS server. This is consistent with the fact, that nodes are in a private network and are inserted in a private domain which cannot be seen on the Internet but is used only locally. See appendix for more detail.

4.7 User authentication with ldap

Another useful task, is that of creating a unique database for authentication. Instead of creating the same user on many nodes it is far more convenient to create one centralized database in which to insert user and password information and telling the cluster nodes to point to this server for user authentication. A solution for many years has been the Network Information System NIS, lately the LDAP authentication method is becoming more and more common.

4.8 Access to the Internet using NAT

As can be seen in fig.2 the cluster client nodes are not on the public network, but in a private one, which by definition is isolated from the public network and especially from the Internet. Nevertheless, the client nodes may need to access the Internet in order to download software which is not readily available on the local repository they may need to download data directly from some external server. In this case, it is possible to use a node with 2 Ethernet cards, one attached to the private network and the other connected to the public one in order to function as a Network Address Translation NAT server for the nodes inside the private network. In our case the management node can perform this task. Furthermore, NAT or in our case Port Address Translation (PAT) means that the client nodes use the management node's public address associated to a specific source port in order to go onto the Internet. The management node will automatically cache associations between the node's private IP to the management node's public IP and specific dynamically determined source port. This way many nodes can use the same public IP, obviously the more nodes use the same public IP address to go onto the Internet the more the total available bandwidth is reduced. On a Linux system we can implement a PAT server by using iptables, see appendix I for details.

5 Monitoring and management tools

5.1 Intelligent Platform Management Interface

The Intelligent Platform Management Interface IPMI architecture comprises a suite of protocols and hardware specifications that permit the management of remote computer systems without the need for any operating system to be present or running ("out-of-band management") and independently from the system's CPU and BIOS. It has been developed by many vendors amongst which: IBM, CISCO, Intel, Dell, Hewlett-Packard, NEC corporation are the main contributors. Usually the heart of the system is the Baseboard Management Controller BMC which is integrated into the motherboard and controls low level functions independently of the operating system [5]. The main tasks that are possible with IPMI are:

- Monitoring: measuring fan speed, voltages, power supply status and CPU temperatures and other sensor values;
- Recovery: turning on and off the remote computer system, changing BIOS settings and performing many other actions;
- Logging: keeping records of critical sensor values and critical events;
- Inventory: knowing what hardware sensors and components are connected to the computer system.

It is possible also to define an alerting mechanism by using Simple Network Management Protocol SNMP platform event traps.
In other words the IPMI sits under the operating system and enables the system manager to perform actions that would not be possible with in-band management like SSH and other remote connections. In order for IPMI to function, the remote computer must be connected to the power supply even if not powered on and there must be some communication media between the machine that must be controlled and the system manager computer, the latest implementation of IPMI is version 2.
In fig.3 the IPMI architecture can be seen. The BMC micro-controller is at the center of the

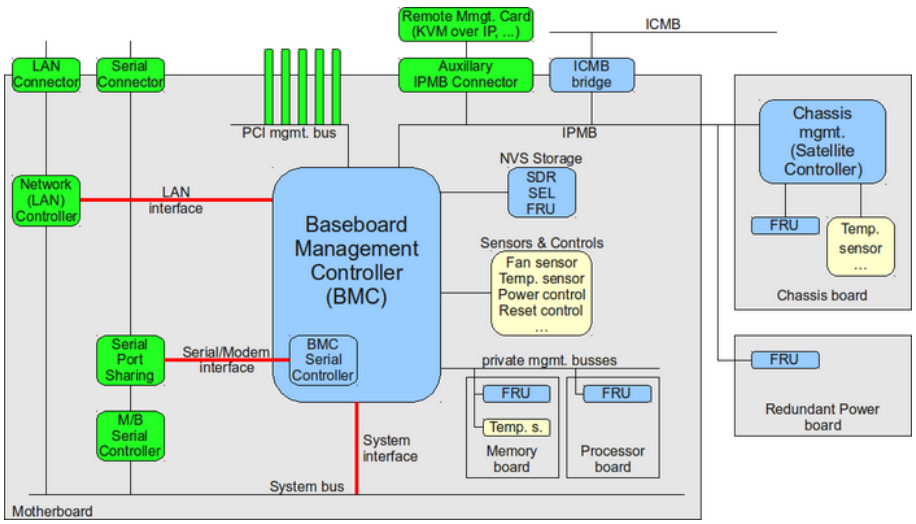


Figure 17: IPMI and BMC infrastructure.

infrastructure, locally it can be accessed through the system interface. The BMC communicates with satellite controllers through the Intelligent Platform Management Bus IPMB, which is an I2C based serial bus through which communications between controllers inside the same chassis

are possible, a later innovation of this is the SMBus. While Intelligent Chassis Management Bus ICMB permits the communication between controllers in different chassis.
In order for IPMI to be really useful it is necessary that IPMI commands can be sent locally through the operating system ("in-band") but especially remotely by using:

- a dedicated NIC just for IPMI purposes (out-of-band);
- the computer system NIC can be shared for communications (side-band);
- serial connections (out-of-band)
- serial over LAN (SOL) (out-of band)

The in-band feature means that the operating system is up: we can control send IPMI messaged to the BMC through the operating system or we can control the computer remotely by using tools such as SSH. While out-of-band means that the computer may be controlled even if its operating system is down. Side-bands means that we use the motherboard's NIC to control the BMC and not a dedicate card. Anyhow, when using a LAN interface, commands are sent to the BMC through remote control management protocol RCMP, which uses UDP port 623, this is also called IPMI-over-LAN as the IPMI control set was initially thought for serial communications. SOL is an important feature of IPMI 2, in this case, the information which is forwarded to the motherboard's serial port is then redirected to a LAN connector and a switched network may be used.
Information is stored into the System Event Log SEL and in the Sensor Data Record SDR. In the former, logs are sent regarding critical events or failures, while in the latter there is information regarding the type and number of sensors. Information may be stored also in Field Replaceable Units FRU.
Interaction with the BMC communication interfaces may be obtained by using channels from 0 to 15, each of which must be configured appropriately. In particular a user with administrator privileges and a password must be defined when using a channel for remote access. Usually the zero channel is used by the IPMB while the following 10 channels may be used for LAN communications. So in order to access the BMC remotely one must choose a channel (e.g. channel 2), define a user with administrator privileges, a password and other optional features such as MD5 encryption. For that channel the IP address, netmask and gateway must also be configured.
Sessions may also be configured in order to obtain authentication or process several IPMI streams on a single channel.
The software implementation of IPMI 2, has been accomplished by many open source products such as OpenIPMI, freeipmi and ipmitool, the details are dealt with in appendix J, anyhow, one must install the software and then he can interact with the BMC of his own computer through command line and he may configure it. Off course, the BMC may be configured also through BIOS. If many computers have been configured appropriately then it is possible to manage a computer remotely via the software tools described above, installed on the system manager's computer.

5.2 Ganglia

Ganglia is an open source monitoring tool used mainly to gain CPU, memory and network load informations and statistics. It has low overhead and uses technologies such as External Data Representation XDR with Uniform Datagram Protocol UDP messages for data transfer, eXtensible Markup Language XML for data poling and representation and Round Robin Database RRD tool for data storage and management. Ganglia comprises of two daemons: gmetad and gmond, which are mainly used to aggregate and send node information respectively. The first daemon is usually installed on the server node while the second on the client nodes.

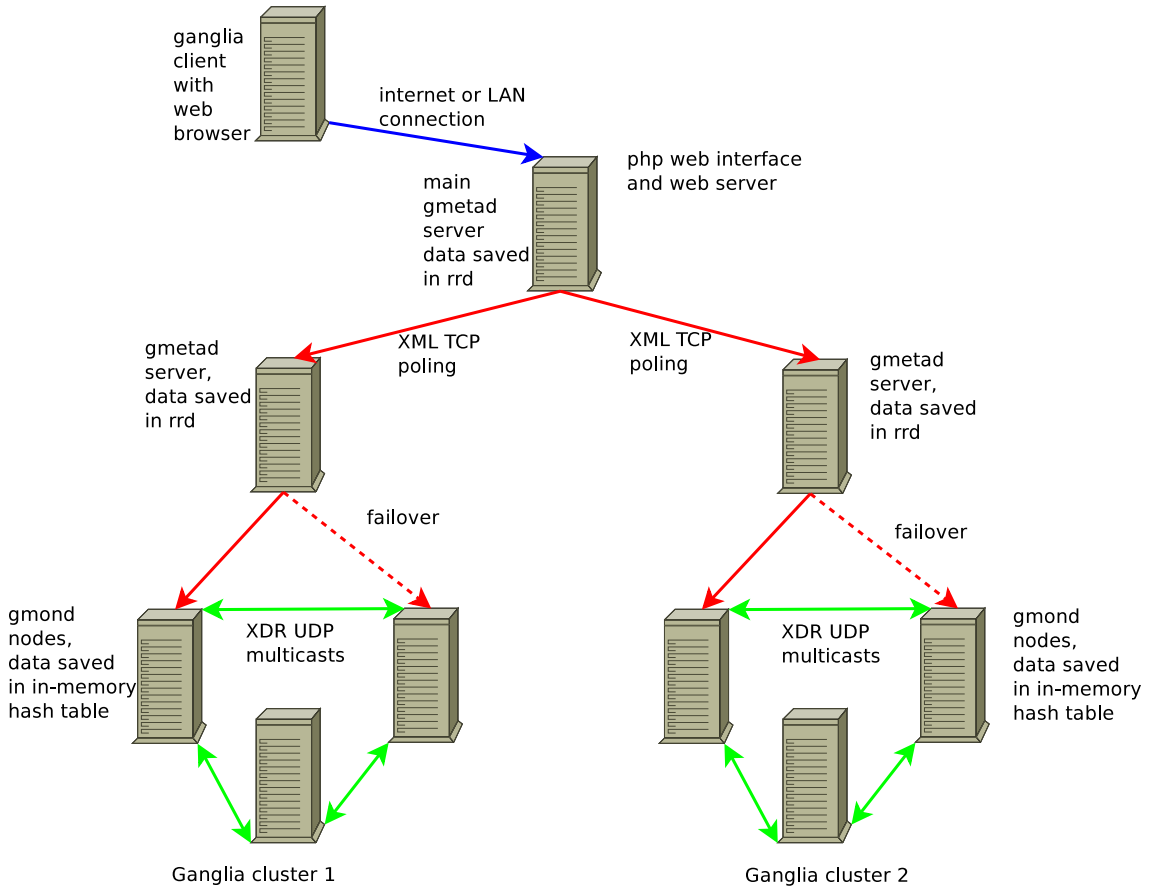


Figure 18: A complex ganglia architecture with two ganglia sub-clusters. The green arrows indicate communication between gmond nodes through XDR UDP messages while the red lines inidcate gmetad poling through XML TCP messages.

On each client node, the gmond daemon has performs the following tasks: it monitors changes on the host, it announces them in unicast or multicast mode using XDR, it listens to changes in the cluster nodes through unicast or multicast mode, it responds to XML requests of the cluster state that are sent from a gmetad server. Therefore data transmission can occur in 2 ways: XDR using UDP (default port 8649) between gmond daemons or XML using TCP (default port 8951) between gmond and gmetad daemons.
On the server node, the gmetad node has the task of poling gmond clients or even other gmetad servers through XML requests in order to gain cluster or clusters information. Data is then usually stored locally in a RRD and can then be accessed by a php engine, served through a web server (e.g. apache2), then the web-interface can be observed locally or remotely through a web browser.
Ganglia uses a multicast-based listen/announce protocol in order to automatically discover nodes added or removed to the cluster without any predetermined knowledge on the cluster membership or topology. When a node does not send any information for over a fixed amount of time it is considered automatically down and is then removed from the ganglia cluster. Conversely, when a node is turned on and it has the gmond daemon correctly configured, it is added automatically to the ganglia cluster and all its information appears on the web interface.
In the IBM cluster we used ganglia in the following way: the gmetad daemon was installed on the management node, together with the RRD tool. The php web interface and a web server were also installed. In addition to this the management node is also a gmond client with the possibility of listening to other gmond XDR announcements. The client nodes send their in-

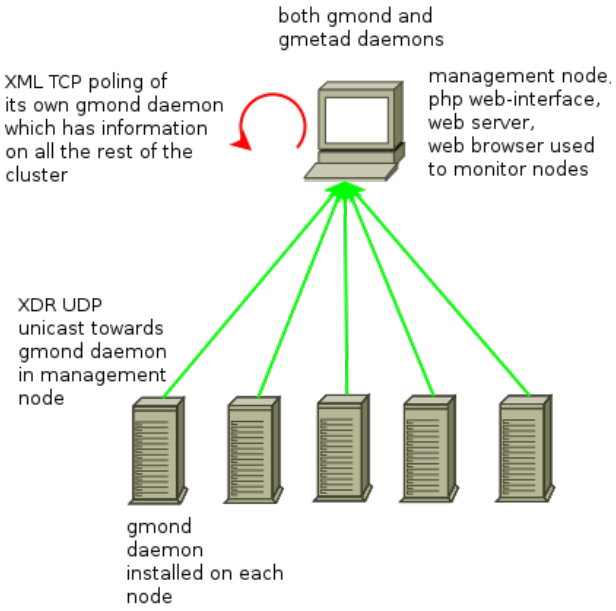


Figure 19: Ganglia architecture of cluster described in this report. All nodes send unicast XDRs to the management gmond daemon, the gmetad daemon in the management node poles itself in order to gain information on cluster nodes and stores data in RRD. The php engine gets data from the RRD and displays it through a web-server. By opening a browser on the management node it is possible to view node statistics.

formation not in multicast mode but in unicast only to the management node gmond daemon. The gmetad daemon on the management nodes poles the it own gmond daemon for cluster information which is then stored in a RRD and is then accessed by the php web interface. For more details on how to configure ganglia go to appendix K.

5.3 pconsole and shmux

In order to configure many nodes simultaneously (e.g., install packages, alter configuration files), tools such as pconsole and shmux are useful. In the case of pconsole, it is possible to open many xterminals on the management node. Each terminal corresponds to a different node. By typing commands on a console terminal, these commands are echoed to all the other terminals and therefore to all the nodes. In order to use pconsole you must first perform ssh key exchange and authentication between the management node and all the other nodes (section 4.4). The configuration details of pconsole may be seen in appendix L.

Another useful tool is the shmux "shell multiplier", a command line tool through which it is possible to perform the same command on many nodes at the same time. Also in this case key exchange and authentication between management node and the other nodes must be performed before hand. A simple example would be that of shutting down all nodes at the same time, in this case one may create a file with a list of node IPs or hostnames (if the file /etc/hosts or the DNS server contains all node resolution) and it can be fed into the shmux command followed by the instruction that must be performed on the nodes listed in the file. In the next example we first create a file "used_hosts" with all the hostnames of the nodes that we want to control than we apply it to shmux in order to shut down all the nodes:

- shmux -c "halt" - < used_hosts;

it is possible to insert any command between brackets in order to interact with many nodes simultaneously.

6 Tests using MPI

Once we have defined a hierarchical structure, installed the operating system on our client and server nodes, connected the nodes together with dedicated networks, and we are sure that the nodes can communicate between each other without passwords, it is possible to implement a middleware that binds the nodes together in order to run parallel programs across a them. Parallel code can also be implemented without any middleware. For example, in the case of SIMD one can launch identical code on several nodes through ssh sessions which process different data sets, nevertheless, the difficulty in parallelization is that the nodes must coordinate and synchronize to handle common data structures, share memory and execute subprocesses on different CPUs. All this can be done by middleware software, while implementation details are hidden from the user, who must only launch the program on a single node as if it were a sequential program. The interface takes care of making the cluster seem as one big computer to the user. Some middleware examples are Parallel Virtual Machine (PVM) or Message Passing Interface (MPI), the latter has been used in this report.

The MPI is a effort to define such interface (syntax, libraries) in order to enable communication in a shared-memory or distributed memory multiprocessor environment. It is portable across many platform and programming languages and is independent on the communication technology (e.g., ethernet, infiniband) between nodes. MPI commonly supports c, c++ and fortran programming languages but python and java support is also available.

There are many implementations of MPI (e.g., OpenMPI, LAM/MPI, MPICH2). For testing purposes, MPICH2 was chosen here. The installation and configuration of MPICH2 is described in the appendix M.

Usually programs are launched on the master node (e.g., the cluster's front-end on which login is accomplished), while the actual execution of the program subprocesses usually occurs on the slave nodes (e.g., some cluster's computing nodes). Nevertheless the MPICH2 daemon (mpd) must run both on master and slave nodes.

It is possible to do some testing using the simple programs which are available in the documentation package. Moreover some good test programs can be obtained from the website: https://computing.llnl.gov/tutorials/mpi/exercise.html. If one wants to test embarrassingly parallel program (i.e., 100% parallel) one can try the mpi_prime c or fortran program. For example, the front-end is used as the master node and 4 computing nodes (a total of 32 PUs in our case) are used as slave nodes. This program searches for prime numbers in a rage of 1 to $25 \cdot 10^7$ (tab. 2). In this case there is no need for subprocesses to communicate, they can just share the range over which to look for prime numbers and then report the result to the master node. As the number of CPUs doubles the runtimes decreases more or less linearly.
This is not always true, it depends on the portion of sequential code embedded in the program and often it may happen that after a certain number of CPUs performance does not improve at all as stated by Admahal's law, in some cases it might even get worst, this is called parallel slow down.
In some cases, especially in the case of very simple toy programs, the time spent sending messages to distinct nodes may be higher than the actual computing time, in this case no gain is obtained in parallelizing a program.

	runtime [s]		
n° of processors	mpi_prime	VecSum	quad_mpi
2	185	0.000425	0.3
4	92	0.000360	0.1
8	46	0.000111	0.05
16	23	0.00053	0.04
32	11	0.000022	0.09

Table 2: Runtime for 3 different programs: mpi_prime, VecSum and quad_mpi. As the number of processing units that are used increase the runtime varies. The first program computes prime numbers in a range of numbers, the second computes the sum of elements in a vector, the third computes an integral using the quadrature rule.

Appendix

In the appendix, we will show basic configuration steps to get your service up and running, the symbol [...] in configuration files indicates that the default configuration or parts of the configuration which are trivial have been omitted. Furthermore, IP addresses will be indicated generically as “xxx.xxx.xxx.xxx".

A Turn on the Cluster

Turn it on

- Go behind the racks and turn up the 5 leavers corresponding to each rack (see fig.6a):
  - one big red plug (380 V for rack ventilation)
  - one small blue plug (220 V for rack monitoring)
  - three blue plugs (220 V for internal computer system power supply)

- Go to front of cluster, inside each rack on the left black leaver must be switched in the up direction, consequently press green button, after two minutes computers should turn on (see fig. 1).
- Turn on storage servers then front-ends (inside rack two) afterwards turn on computational nodes starting from 1 to 60 (inside rack one and three). Single nodes can be turned on manually simply by pressing white button on each node or remotely by using ipmitool utilities.
- Use gpfs utilities to mount GPFS file system and therefor /home directory on each node (e.g., mmstartup -a on front-end node).

Turn it off

- On front-end, use GPFS utilities to unmount GPFS file system and therefor /home directory (e.g., mmshutdown -a).
- Shut down operating system on each computational node, you can create a script in order to automatize this task.
- Turn off each node manually by pressing white button or remotely by using ipmitool utilities.
- On the front of the cluster, inside each rack on the left, black leaver must be switched in the down direction (see fig. 1).
- Go behind the racks and turn down the 5 leavers corresponding to each rack (see fig.6a).

B Configuring GPFS

B.1 installation

Operating System: Scientific Linux 6.2 x86_64

Main Software: gpfs.base-3.1.0-1.x86_64.rpm, gpfs.gpl-3.1.0-1-noarch.rpm
gpfs.docs-3-1.0-1.noarch.rpm , gpfs.msg.en_US-3.1.0-1.noarch.rpm

Update Software: gpfs.base-3.4.0-18.x86_64.update.rpm, gpfs.docs-3.4.0-18.noarch.rpm,

gpfs.gpl-3.4.0-18.noarch.rpm, gpfs.msg.en_US-3.4.0-18.noarch.rpm

Additional Software: kernel-devel, kernel-headers, gcc-c++,
compat-libstdc++-33.i386, compat-libstdc++-33.x86_64,
imake.i386, imake.x86_64, ksh, kcp, scp, ssh

- install on every node the main software packages placed under common directory (it is not free, it must be acquired from IBM):
  rpm -ivh /path/gpfs_software_folder/gpfs.base-3.1.x.x.x86_64.rpm
  rpm -ivh /path/gpfs_software_folder/gpfs.docs-3.1.x.x.noarch.rpm
  rpm -ivh /path/gpfs_software_folder/gpfs.gpl-3.1.x.x.noarch.rpm
  rpm -ivh /path/gpfs_software_folder/gpfs.msg-3.1.x.x.noarch.rpm

- update software placed under common directory (updates can be obtained from http://www-933.ibm.com/support/fixcentral/ for free):
  rpm -Uvh /path/gpfs_software_folder/gpfs.base-3.4.x.x.x86_64.rpm
  rpm -Uvh /path/gpfs_software_folder/gpfs.docs-3.4.x.x.noarch.rpm
  rpm -Uvh /path/gpfs_software_folder/gpfs.gpl-3.4.x.x.noarch.rpm
  rpm -Uvh /path/gpfs_software_folder/gpfs.msg-3.4.x.x.noarch.rpm

- it is likely that these steps will not work
  unless you perform the following tasks in /usr/lpp/mmfs/src/config:
  - export SHARKCLONEROOT=/usr/lpp/mmfs/src
  - cp env.mrc.proto env.mrc.sample
  - edit site.mrc by inserting or substituting following tags:
  #define GPFS_LINUX
  #define GPFS_ARCH_X86
  LINUX_DISTRIBUTION = REDHAT_AS_LINUX
  #define LINUX_DISTRIBUTION_LEVEL 90
  #define LINUX_KERNEL_VERSION 2063299 (e.g. for kernel version 2.6.32-358)
  KERNEL_HEADER_DIR = /lib/modules/‘uname -r‘/build/include
  KERNEL_BUILD_DIR = /lib/modules/‘uname -r‘/build
  - edit the file /etc/redhat-release:
  “Red Hat Enterprise Linux Server release 6.2 (Tikanga)”

- It is important to note a problem that might occur on scientific linux distributions:
  the soft link /lib/modules/‘uname -r‘/build is used to build kernel modules and it points to /usr/src/kernel/2.6.x. In some versions of linux a build will not work unless a directory /lib/modules/‘uname -r‘/build is actually created and the whole content of the /usr/src/kernel/2.6.x is copied in this directory.

- build the portability layer (kernel modules):
  in /usr/lpp/mmfs/src you must perform:
  make Autoconfig
  make World
  make InstallImages make rpm

- once the build is complete you will find in the /usr/lpp/mmfs/bin directory, modules such as:
  mmfslinux, mmfs26, tracedev, dumpconv, lxtrace.

- make rpm will package the binary modules so that you do not have to repeat this step on other nodes but you can install them with the rpm utility.

B.2 configuration: client-server architecture

The storage server is attached to the disk subsystem through fiber channel and it can see three logical disks (dm0, dm1, dm2) as if they were local. The objective is to unite the three disks under a unique GPFS file system, mount it in the /home directory and export the /home directory to all cluster nodes.

- make sure that every node trusts each other as explained in appendix F, make sure that all hostnames are resolved to their IP address by using local /etc/hosts file or a DNS server
- create a directory (e.g. /root/gpfs/configure), under which to place two important files:node.desc and disk.desc. The first indicates the nodes that will partecipate to the GPFS cluster, the second describes the disks that will participate to the GPFS file system.
- e.g nodes.desc:
  gpfs2.test-gpfs
  gpfs3.test-gpfs
  gpfs4.test-gpfs
  [...]
- e.g disk.desc:
  /dev/dm-0:storage-server::dataAndMetadata:1:disk01_array01
  /dev/dm-1:storage-server::dataAndMetadata:1:disk02_array02
  /dev/dm-2:storage-server::dataAndMetadata:1:disk03_array03
- create gpfs cluster:
  mmcrcluster -C gpfs-cluster -N nodes.desc -p gpfs1.test-gpfs:quorum -r /usr/bin/ssh -R /usr/bin/scp
- you can add nodes at a later stage:
  mmaddnode gpfs10.test-gpfs
- look at the new GPFS cluster:
  mmlscluster
- turn the gpfs daemon on each node:
  mmstartup -a
- create network shared disk:
  mmcrnsd -F disk.desc
- create the file system:
  mmcrfs gpfs-fs -F disks.desc -A yes -T /home -m 1 -M 2 -r 1 -R 2 -Q yes -B 256K -n 10

B.3 delete gpfs cluster

- mmumount /dev/gpfs-fs -a
- mmdelfs /dev/gspfs-fs
- mmdelnsd disk01_array01

- mmdelnsd disk02_array02
- mmdelnsd disk03_array03
- mmshutdown -a
- mmdelnode -a

B.4 remove GPFS software

- rpm -e /path/gpfs_software_folder/gpfs.base-x.x.x.x86_64
- rpm -e /path/gpfs_software_folder/gpfs.gpl-x.x.x.noarch
- rpm -e /path/gpfs_software_folder/gpfs.docs-x.x.x.noarch
- rpm -e /path/gpfs_software_folder/gpfs.msg-x.x.x.x86_64
- rm -rf /var/mmfs
- rm -rf /var/adm/ras
- rm -rf /usr/lpp
- rm -rf /tmp/*

B.5 configure GPFS quotas

Under the gpfs defined partions there will be quota files regarding user, group and file quotas, in order to edit and customise quotas use the following utilities:

- mmlsquota -a
- mmrepquota -a
- mmedquota -u username
- change soft and hard limits
- mmcheckquota
- mmquotaon
- mmquotaoff

C Extract ISO image and copy on system's hard drive

In order to extract the content of an iso image on a dvd or downloaded from the Internet on a specific folder in the systems hard drive. This is useful for network booting from your server.

- mkdir /export/OS
- mount -o loop -t iso9660 <image>.iso /mnt
- cp -r /mnt/<image>.iso /export/ISO

Often the operating system will reside on two dvds, the second of which contains extra packages. In order to be coherent usually the extra packages should be copied in the packages directory that is found in the first dvd.

D Configure PXE booting and network installation

On the server node you must configure services such as DHCP, TFTP and NFS server while on the client node you must enable booting from the NIC assuming that the network card supports PXE. The latter task may be performed through BIOS or remotely by using IPMI software tools (appendix J).

D.1 configure DHCP server

Operating System: Scientific Linux 6.2 x86_64
Software: dhcp.x86_64

The dhcp server is used in order to assign to the PXE enabled NIC an IP address upon boot

- install dhcp server daemon: "yum install dhcp.x86_64"

- edit configuration file in /etc/dhcp/dhcpd.conf:
  ddns-update style none;
  ignore client-updates;
  deny unknown-clients;
  not authoritative;
  default-lease-time 600;
  max-lease-time 7200;

  option domain-name "mydomain";
  option domain-name-servers xxx.xxx.xxx.xxx;
  option routers xxx.xxx.xxx.xxx;

  subnet xxx.xxx.xxx.xxx netmask xxx.xxx.xxx.xxx {

  next-server xxx.xxx.xxx.xxx;
  filename pxelinux.0;

  host hostname{hardware ethernet xx:xx:xx:xx:xx:xx;fixed address xxx.xxx.xxx.xxx;}

  }

- besides the basic options, it is important to note the "next-server" and the "filename" option where we specify from which tftp server we will get the bootloader and in which location under the root tftp directory.

D.2 configure TFTP server for network booting

Operating System: Scientific Linux 6.2 x86_64
Software: syslinux-tftpboot.x86_64 (bootloader and modules in /tftpboot for network booting), xinetd.x86_64 or tftp-server.x86_64 (server), tftp.x86_64 (client for testing)

- First of all download syslinux-tftpboot.x86_64, you will find a /tftpboot directory with many files inside, the only one you need is pxelinux.0 which must be left here, create 2

subdirectories: pxelinux.cfg and OS, in the first directory we must put the bootloader file while in the second we will put the kernel image and the initial ramdisk.

- the bootloader file can be named “default" and it will be used by all the nodes or it can be named with the IP address in hexadecimal that the DHCP server will assign to the node, in this case the file is valid only for that node and will be used only for that node or it can be named with the IP’s network portion expressed in hexadecimal, in this case the file will be valid for all the nodes in that network.
  An example of the bootloader file in /tftpboot/pxelinux.cfg/default:

  default MYOS

  # manual installation
  label MYOS
  kernel OS/vmlinuz
  append initrd = OS/initrd.img method nfs:xxx.xxx.xxx.xxx:/export/OS/ devfs=nomount

  # automatic installation
  label MYOS1
  kernel OS/vmlinuz
  append initrd = OS/initrd.img ksdevice=eth0 ip=dhcp
  ks=nfs:xxx.xxx.xxx.xxx:/export/OS/ks/default.ks

- In the default file we can choose how to boot depending on the boot tag, in the first case “manual installation”, the kernel and the ramdisk is loaded from the /tftpboot/OS directory and the rest of the OS is loaded from the NFS server /export/OS directory.
  In the automatic configuration all is the same, except we tell the system which interface to use and also we tell the system to automatically get a IP address through DHCP. In addition the kickstart configuration file, named default.ks, is indicated.

- intall the tftp server: yum xinetd.x86_64

- go to file /etc/xinetd.d/tftp:
  service tftp {
  bind = xxx.xxx.xxx.xxx #your tftp server’s IP
  [...]
  server_args = -s /tftpboot #tftp root directory
  disable = no #enable the xinetd’s tftp service
  [...]
  }

- start the xinetd daemon: service xinetd start

- install the tftp client: yum install tftp.x86_64

- test the tftp server locally :
  tftp xxx.xxx.xxx.xxx
  get somefile

D.3 configure NFS server

Operating System: Scientific Linux 6.2 x86_64
Software: nfs-utils.x86_64

The Network File System daemon is used in this case in order to export the complete operating system to the system that is performing network installation. It is also used in post configuration in order to copy configuration files and software to the cluster node.

- create directory mkdir /export and extract operating system ISO image here
- install nfs: yum install nfs-utils.x86_64
- go to /etc/exports:
  /export xxx.xxx.xxx.0/24(rw, sync, no_root_squash)
- start server: service nfs start

E Configuration of kickstart

You must specify that you will use a kickstart configuration file for installation of the OS in the bootloader file (appendix D.2). You must also indicate location of the file.
In our case the kickstart file is located in the NFS exported directory, in particular in /export/OS/ks/default.ks, the following lines illustrate an excerpt of the default.ks file that was used:

```
# basic configuration parameters install
nfs –server=xxx.xxx.xxx.xxx –dir=/export/OS
lang en_US.UTF-8
keyboard it
network –onboot yes –device eth0 –bootproto dhcp –noipv6
network –onboot no –device eth1 –noipv4 –noipv6
rootpw –iscrypted place_crypted_password_here
reboot
firewall –enabled
authconfig –enableshadow –passalgo=sha512
selinux –enabled
timezone –utc Europe/Rome
bootloader –location=mbr –driveorder=sda,sdb –append=" rhgb crashkernel=auto quiet"

# partitions zerombr
clearpart –all
part / –fstype=ext4 –size=50000 –ondisk=sda –asprimary
part swap –size=20000 –ondisk=sda –asprimary
part /scratch1 –fstype=ext4 –size=100 –grow –ondisk=sda –asprimary
part /scratch2 –fstype=ext4 –size=100 –grow –ondisk=sdb –asprimary

# package installation repo –name="Scientific Linux" –baseurl=nfs:xxx.xxx.xxx.xxx:/export/OS
–cost=100
```

```
%packages
@base
@client-mgmt-tools
@core
@debugging
[...]
OpenIPMI
lm_sensors
rsh
ksh

# post installation script

# turn off services
/etc/init.d/NetworkManager stop
/sbin/chkconfig NetworkManager off
# change run level
/bin/sed -i s/id:5:initdefault:/id:3:initdefault:/g /etc/inittab
# add a repository
/bin/echo "
[epel]
name=extended packages enterprise linux
baseurl=http://dl.fedoraproject.org/pub/epel/6/x86_64/
enabled=1
gpgcheck=1 " >> /etc/yum.repos.d/sl-other.repo
# permanently load InfiniBand over Lan module
/sbin/modprobe ib_ipoib
echo "modprobe ib_ipoib" >> /etc/sysconfig/modules/ib_ipoib.modules
/bin/chmod 755 /etc/sysconfig/modules/ib_ipoib.modules
# retrive mac address and last octect of ip address obtained from dhcp server
# and configure static IP address
ipaddr=‘/sbin/ifconfig eth0 | grep "inet addr" | awk -F: ’{ print $2 }’
| awk ’{ print $1 }’| awk -F. ’{ print $4 }’‘
hwaddr=‘/sbin/ifconfig eth0 | grep "HWaddr" | awk ’{ print $5 }’‘

echo "
DEVICE="eth0"
BOOTPROTO="static"
BROADCAST="xxx.xxx.255.255"
GATEWAY="xxx.xxx.xxx.xxx"
HWADDR="$hwaddr"
IPADDR="xxx.xxx.xxx.$ipaddr"
NETMASK="255.255.0.0"
NM_CONTROLLED="no"
ONBOOT="yes"
TYPE="Ethernet" " > /etc/sysconfig/network-scripts/ifcfg-eth0

# make sure nfs directory is mounted
mount xxx.xxx.xxx.xxx:/export /mnt
# copy files from directory
cp /mnt/configFiles/hosts /etc/ # install software from nfs directory
```

rpm -ivh /mnt/software/GANGLIA/*.rpm

%end

F Configure SSH password-less access

Operating System: Scientific Linux 6.2 x86_64
Software: openssh-clients.x86_64, openssh-server.x86_64

- install software on all nodes
- run server daemon on every node: “service sshd start”
- from each node access manually or with a script every other node (including the local node) in order for server authentication to occur: every node saves certificate of the other nodes in the file /home/user/.ssh/known_hosts
- on one node create rsa keys and follow wizard:
  ssh-keygenerate rsa
- copy public key in file /home/user/.ssh/authorized_keys on each node, keep private key in folder /home/user/.ssh
- you should now be able to access every node without a password and without being prompted for server authentication.

G Configure local repository

In order to configure local repository on a server you must first configure an ftp server that permits nodes to download software from the server, than write a script that downloads packages from official repository on Internet to ftp directory on your server. Run script periodically by using the cron daemon. Finally configure clients’ repository list file so that it points to your ftp server.

G.1 configure ftp server

Operating System: Scientific Linux 6.2 x86_64
Software: vsftp.x86_64
(ftp server), ftp.x86_64 (client for testing)

- install software: “yum install vsftp.x86_64 ftp.x86_64”
- modify file /etc/vsftpd/vsftpd.conf:
  listen = YES
  local_enable=YES
  anonymous_enable=YES
  anon_root=/var/ftp
  no_anon_password=NO
  listen_address=xxx.xxx.xxx.xxx
  listen_port=21

- in the previous file we are telling the ftp server to listen to requests locally and also remotely but only out of a specific interface (in our case the private one), the ftp server must work in anonymous mode (no user defined) with no password and the ftp root will be /var/ftp.
- service vstpd start
- test locally or remotely with command line : “ftp xxx.xxx.xxx.xxx” or with a browser. If prompted for user or password in anonymous mode it will be in both cases “ftp”.

G.2 create mirror script

Operating System: Scientific Linux 6.2 x86_64
Useful Links: https://www.scientificlinux.org/download/mirroring/

Regarding Scientific Linux it is possible to browse to ftp://ftp.scientificlinux.org/linux/scientific/ in order to look at which versions and packages are available, you may use ftp or rsync to create mirror.

- create script file and download mirror directly to the correct ftp directory so that it can be subsequently used by the nodes as a package repository, e.g. download_mirror:

  rsync -avkSH –delete –exclude=i386 –exclude=archive
  rsync://rsync.scientificlinux.org/scientific/6.2/
  /var/ftp/linux/scientific/6.2

- in the previous script, besides the directories that are being excluded all the rest is being mirrored precisely .
- the initial download will take many hours if not days and usually download size is of the order of hundreds of GBs.
- In order to keep mirror synchronized it is useful to create cron daemon that runs the script above once a day
- on the client side the file must be modified in order to point to the local repository, e.g. /etc/yum.d/yum.repos.d/sl-other.repo:

  $[loc - sl]$
  name=scientific linux local base repository
  baseurl=ftp://xxx.xxx.xxx.xxx/linux/scientific/6.2/x86_64/os/
  enabled=1
  gpgcheck=0

- you must disable any other repository entry that point to servers on the Internet.

H Configure local DNS server

Operating System: Scientific Linux 6.2 x86_64
Software: bind.x86_64
Software Dependencies: bind-chroot.x86_64, bind-libs.x86_64, bind-utils.x86_64

- Install software;
- in the main configuration file, you can set options and define direct and inverse zones, it is divided into an options section, a zone section and an include section, /etc/named.conf:

  options {
  listen-on port 53 { 127.0.0.1; xxx.xxx.xxx.xxx;};
  listen-on-v6 port 53 none; ;
  directory "/var/named";
  dump-file "/var/named/data/cache_dump.db";
  statistics-file "/var/named/data/named_stats.txt";
  memstatistics-file "/var/named/data/named_mem_stats.txt";
  allow-query 127.0.0.1;xxx.xxx.xxx.0/24;; allow-transfer none; ; allow-recursion none;;
  recursion no; #forwarders { xxx.xxx.xxx.xxx; };

  dnssec-enable yes;
  dnssec-validation yes;
  dnssec-lookaside auto;

  /* Path to ISC DLV key */
  bindkeys-file "/etc/named.iscdlv.key";

  managed-keys-directory "/var/named/dynamic";
  };

- additional options, for example switching off IPv6 requests can be set in /etc/sysconfig/named:
  [...] OPTIONS = " -4";
  [...]
- the actual database where records are kept is in the file /var/named/fwd.mydomain:
  $TTL 86400
  IN SOA mgm2.srt-grid. root.srt-grid. (
  2013061002 ;Serial
  3600 ;Refresh
  1800 ;Retry
  604800 ;Expire
  86400 ;Minimum TTL
)
  IN NS management.mydomain.

  node1 IN A xxx.xxx.xxx.201
  node2 IN A xxx.xxx.xxx.202
  [...]

- the inverse resolution records file is kept in /var/named/rev.mydomain:

  $TTL 86400
  IN SOA mgm2.srt-grid. root.srt-grid. (
  2013061001 ;Serial
  3600 ;Refresh

1800 ;Retry
604800 ;Expire
86400 ;Minimum TTL
)
IN NS management.mydomain.

management IN A xxx.xxx.xxx.xxx
206 IN PTR management.mydomain.

node1 IN A xxx.xxx.xxx.202
202 IN PTR node1.mydomain
node2 IN A xxx.xxx.xxx.203
203 IN PTR node2.mydomain

- modify server's /etc/resolv.conf:
  search mydomain
  nameserver 127.0.0.1
  # local dns
  nameserver xxx.xxx.xxx.xxx # secondary dns for public lan and internet

- modify client's /etc/resolv.conf:
  search mydomain
  nameserver xxx.xxx.xxx.xxx # local dns server's IP address nameserver xxx.xxx.xxx.xxx
  # secondary dns for public lan and internet

- start service: "service named start"

I Configure NAT

Operating System: Scientific Linux 6.2 x86_64

- supposing the eth1 is the public interface while the eth0 is the private one, create bash script file, e.g. myNat:

  #!/bin/bash

  IPT=/sbin/iptables
  NAME=myIptablesNat
  STATUS=myIptablesNatStatus

  case "$1" in
  start)
  $IPT -t nat -A POSTROUTING -o eth1 -j MASQUERADE
  $IPT -t filter -A FORWARD -i eth1 -o eth0 -m state –state RELATED,ESTABLISHED
  -j ACCEPT
  $IPT -t filter -A FORWARD -i eth0 -o eth1 -j ACCEPT
  echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
  echo "service is on" > $STATUS

```
;;
stop)

$IPT -t nat -D POSTROUTING -o eth1 -j MASQUERADE
$IPT -t filter -D FORWARD -i eth1 -o eth0 -m state –state RELATED,ESTABLISHED
-j ACCEPT
$IPT -t filter -D FORWARD -i eth0 -o eth1 -j ACCEPT
echo 0 > /proc/sys/net/ipv4/conf/all/forwarding
echo "service is off" > $STATUS
;;
)
echo "Usage: $NAME {start|stop|status}" >&2
exit 1
;;
esac

exit 0
```

- the previous file takes traffic from the private network in the eth0 interface and it forwards it to the public eth1 interface, it masquerades the private IP with the public one, it permits reply traffic from servers on the Internet to the private network and it always permits traffic from the private network to the public one. When the service is stopped, rules are removed from the nat and filter tables and forwarding is disabled.
- start iptables daemon: “service iptables start”
- start service: “./myNat start”

J IPMI configuration

System: IBM system x3550
Operating System: Scientific Linux 6.2 x86_64
Software : OpenIPMI.x86_64 , ipmitool.x86_64

- Install OpenIPMI which provides the drivers (kernel_modules) such as ipmi_si, ipmi_devintf, ipmi_msghandler and install ipmitool which provides the command line interface which permits to interact with the IPMI architecture.
- start up the ipmi daemon: service ipmi start, now there should be a /dev/ipmi0 file
- Now ipmitool can be used locally in order to check sensor values and so on. If we want to use ipmitool remotely than we must choose a IPMI channel (e.g., 1-10) for lan connections and configure LAN settings, user and so on.
- In order to configure LAN settings on channel 1, we must configure: static IP use, the IP itself, netmask, channel access and other information, first of all it is useful to look at the current settings:

  - ipmitool lan print 1
  - ipmitool lan set 1 ipsrc static

- ipmitool lan set 1 ipaddr xxx.xxx.xxx.xxx
- ipmitool lan set 1 netmask xxx.xxx.xxx.xxx
- ipmitool lan set 1 defgw ipaddr xxx.xxx.xxx.xxx
- ipmitool lan set 1 defgw macaddr xxx.xxx.xxx.xxx
- ipmitool lan set 1 arp respond on xxx.xxx.xxx.xxx
- ipmitool lan set 1 auth ADMIN MD5
- ipmitool lan set 1 access on

- create a user with administrator privileges, associate user to a user id, and other useful paramenters, first of all you may want to look at channel 1 current user configuration:

  - ipmitool user list 1
  - ipmitool user set name userid username
  - ipmitool user set password userid yourpw
  - ipmitool channel setaccess 1 userid link=on ipmi=on callin=on privilege=4
  - ipmitool user enable userid

- if configuration has been performed on each node, using appropriate user, password, subnet and so on, then it is possible to use the following basic useful commands for remote administration such as checking chassis status, turning on or off computers, choosing boot device for a given system, observing SEL or SDR logs and so on:

  - ipmitool -H xxx.xxx.xxx.xxx -U username -P yourpw chassis power {on|off|status}
  - ipmitool -H xxx.xxx.xxx.xxx -U username -P yourpw chassis bootdev {pxe|disk|bios|cdrom}
  - ipmitool -H xxx.xxx.xxx.xxx -U username -P yourpw sel {info|list|elist|clear}
  - ipmitool -H xxx.xxx.xxx.xxx -U username -P yourpw chassis sensor
  - ipmitool -H xxx.xxx.xxx.xxx -U username -P yourpw channel info
  - ipmitool -H xxx.xxx.xxx.xxx -U username -P yourpw fru info

K Ganglia configuration

Operating System: Scientific Linux 6.2 x86_64
Software : ganglia 3.1 x86_64

On server node:

- include in repository http://*dl.fedoraproject.org/pub/epel*/6/*x*86_64/;
- yum install ganglia-gmetad ganglia-gmond ganglia-web rrdtool httpd;
- configuration files are under /etc/ganglia: gmond.conf and gmetad.conf.
- gmetad.conf:
  data_source "clusterName" 15 localhost #modify this line
- gmond.conf:
  globals{
  [...]
  send_metadata_interval = 60
  }
  cluster{

name = "clusterName"
[...]
}
[...]
udp_send_channel{
#bind_hostname = yes #mcast_join = 239.2.11.71
#ttl=1 host = xxx.xxx.xxx.xxx #ganglia server ip
port = 8649
}
udp_receive_channel{
#mcast_join = 239.2.11.71
port = 8649
#bind 239.2.11.71
}

[...]

- /etc/httpd/conf/httpd.conf:
  [...]
  Listen 127.0.0.1
  [...]

- start daemons: gmond, gmetad and httpd
- cp /usr/share/ganglia /var/www/html/
- ln -s /var/www/html/conf.php /etc/ganglia/conf.php
- open a browser and digit 127.0.0.1/ganglia to use interface, for now only client is the server itself.

On client nodes:

- yum ganglia-gmond
- configure /etc/ganglia/gmond.conf as in server
- customize metrics using gmetrics, suppose you have a script that produces cpu temperature that you can send this value to the ganglia server:
  gmetrics –name cputemp –value ‘./scriptcpu‘ –type int16 –units celcius
  in order for the customized metrics to be sent periodically you must include the abose command in a script which is run by the cron daemon.

L Configure pconsole

Operating System: Scientific Linux 6.2 x86_64
Software : pconsole 1.0 x86_64

- perform ssh key exchange and authentication between management node and all the other nodes;
- download, build and install pconsole tar.gz from e.g, http://www.heiho.net/pconsole/;

- yum install xterm;
- In /usr/local/bin you should find the binary pconsole and the scripts pconsole.sh and ssh.sh, make that this directory is in you path;
- If you want to vary the dimension and font of your terminal and control window modify settings in pconsole.sh:
  [...]
  P_TERM_OPTIONS="-bg white -fg red -bc -geometry 42x10"
  [...]
  P_CONSOLE_OPTIONS="-geometry 42x10"
  [...]
- You may create alias in .bashrc in order to simplify execution of command: e.g., alias pcon = /usr/local/bin nodename1 nodename2 ... ;

M Configure MPI

Operating System: Scientific Linux 6.2 x86_64
Software : mpich 1.2 x86_64

- make sure that ssh key exchange, name resolution (through dns or /etc/hosts) and a common work place (e.g., network exported /home directory) is present on all the nodes;
- download and install on all nodes mpich2.x86_64 mpich2-devel.x86_64 mpich2-doc.x86_64 the first package includes mpi libraries, the second compilers and the third documentation and examples;
- under your /home/user directory on the front-end node (master node) create file .mpd.conf in which to write some security key, for example:
  MPD_SECRETWORD=mr45-j9z
- create file mpd.hosts that indicates node names on which the mpd daemon must run, for example:
  master
  slave1
  slave2
  slave3
- create file exec.hosts that indicates on which hosts you want to execute you program and how many CPUs on each node may be used, for example:
  slave1:8
  slave2:8
  slave3:8
- for testing copy helloworld or some simple program from /usr/share/mpich2 compile it:
  mpi -o helloworld helloworld.c
  (if mathematical libraries are needed, the -lm option is useful to link such libraries during compile time)

- run the mpd daemon on all nodes which will be used including your master specifying communication protocol , file in which nodes are listed, how many nodes you want to use from that list:
  mpdboot -r ssh -f mdp.hosts -n 3

- use mpdtrace to see on which nodes mpd is running
- execute program indicating on which nodes and how many processes you want to spawn:
  mpiexec -machinefile exec.hosts -n 16 ./helloworld
- when you are finished with mpd turn off daemons:
  mpdcleanup -f mpd.hosts

N Acronyms

BIND: Berkeley Internet Name Daemon
BIOS: Basic Input Output System
BMC: Broadband Management Controller
CPU: Central Processing Unit
DHCP: Dynamic Host Configuration Protocol
DNS: Domain Name System
FLOGI: Fabric Login
FC: Fiber Channel
FLOPS: Floating point Operation Per Second
FRU: Field Removable Unit
FTP: File Transfer Protocol
GPFS: General Parallel File System
GPGPU: General Purpose Graphical Processing Unit
HBA: Host Bus Adapter
HCA: Host Channel Adapter
TCA: Target Channel Adapter
HD: Hard Drive
HPC: High Performance Computing
HTTP: Hypertext Transfer Protocol
IB: InfiniBand
IBM: International Business Machines
ICMB: Intelligent Chassis Management Bus
IP: Internet Protocol
IPMI: Intelligent Platform Management Interface
KVM: Keyboard Video Mouse
LAN: Local Area Network
LC: Little Connector
MIMD: Multiple Instruction Multiple Data
MISD: Multiple Instruction Single Data
MPI: Message Passing Interface
NFS: Network File System
OS: Operating System
PLOGI: Port Login
PU: Processing Unit
PVM: Parallel Virtual Machine
RAID: Redundant Array of Independent Disks
RDMA: Remote Direct Memory Access
RSH: Remote Shell
RRD: Round Robin Database
SC: Standard Connector
S_ID: Source ID
SIMD: Single Instruction Multiple Data
SISD: Single Instruction Single Data
SMB: System Management Bus
SMP: Symmetric Multi Processor
SOL: Serial Over LAN
SSH: Secure Shell
TCP: Transport Control Protocol
WAN: Wide Area Network
WWID: World Wide ID

WWNN: World Wide Node Name
XML: Extensible Markup Language
XDR:External Data Representation

References

[1] GPFS Best Practices, Programming, Configuration, Environment and Performance Perspectives. Tutorial for GPFS versions 3.3 and earlier
http://www.greatplains.net/download/attachments/131460/tutorial.v17.2.pdf
last accessed 17/09/2013.

[2] IBM System x3655 Installation Guide.
http://pdf.superwarehouse.com/specs/IBM_79854AU_manual.pdf
last accessed 17/09/2013.

[3] IBM System x3550 Installation and User's Guide.
http://download.boulder.ibm.com/ibmdl/pub/systems/support/system_x_pdf/00d9281.pdf
last accessed 17/09/2013.

[4] IBM System Storage and DS4000 and Storage Manager V10.30.
http://www.redbooks.ibm.com/redbooks/pdfs/sg247010.pdf
last accessed 17/09/2013.

[5] IBM Using Intelligent Platform Management Interface (IPMI) under IBM Linux Platforms
http://pic.dhe.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai.ipmi/liaaiipmi_pdf.pdf
last accessed 17/09/2013.

[6] Introduction to Parallel Computing.
https://computing.llnl.gov/tutorials/parallel_comp/
last accessed 15/09/2013.

[7] Knurr Manual CoolLoop Water-cooled cabinet for lateral attachment to server cabinets.
http://www.emersonnetworkpower.com/en-EMEA/Brands/Knurr/Documents/en/manuals/CoolLoop-en-version_d.pdf
last accessed 17/09/2013.

[8] Nemeth E., Snyder G., Hein T. R., Whaley B.: "Unix and Linux Administration Handbook. Fourth Edition", pp. 14. Publishing Company, City, Country (2010).

[9] Sloan J. D.: "High Performance Linux Clusters: With Oscar, Rocks, openMosix, And MPI", pp. 14. Publishing Company, City, COUNTRY (2009).

[10] Soft Panorama: Linux Multipath.
http://www.softpanorama.info/Commercial_linuxes/Devices/multipath.shtml
last accessed 15/09/2013.

[11] Sterlin T. L.: "Beowulf Cluster Computing with Linux", pp. 14. Publishing Company, City, COUNTRY (2009).

[12] Tropens U., Erkens R., Muller-Friedt W., Wolafka R., Haustein N.: "Storage Networks Explained", pp. 14. Wiley, Chichester, UK (2009).