MASTER THESIS

TITLE: Design and implementation of an efficient management of storage and service rate in “Anella Cultural” project of the Foundation i2CAT

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Rubén Cuadrado Pérez

DIRECTOR: Jesús Alcober

DATE: June, 18th 2007

TITLE: Design and implementation of an efficient management of storage and service rate in “Anella Cultural” project of the Foundation i2CAT

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Rubén Cuadrado Pérez

DIRECTOR: Jesús Alcober

DATE: June, 18th 2007

Overview

This project is developed in the “Anella Cultural”, project sponsored by Foundation I2CAT with the objective to promote the use of the multimedia technology on IP, in the world of culture and show.

In these surroundings due to the quality of the applications and to their origin important requirements of storage exist. For this project volumes of storage have been acquired based on the Fiber Channel technology with minimum capacity for 1,4 TB.

The aim of this project is the valuation of different alternatives that allow us to make an efficient management as far as the procedure of storage and to the service rate. The design of a platform and its consequent implementation.

Once implemented, the correct operation will be verified, for it would be desired to make a series of tests that evaluates that the system, verifying the speed of writing, reading, latency, etc.

Before beginning to create the problem we must know several elements that can help the development of the project, those elements are explained in the annex [Reference to chapter].

Gratitude

I would like to thank some people for having helped me to make this project possible, without them I wouldn't have been able to obtain it.

These people are:

- My parents, they helped me when I needed them.
- Pedro Lorente, he has always been there when I had problems about linux.
- All my friends, but in particular those from St. Joan Despí, they made me be happy again.
- And finally my teacher of English Tony Sánchez, he corrected the entire project and has taught me most of the English I know.

Thank you!

INDEX

Anella Cultural

The objective of the Anella Cultural project is to connect in network the great cultural facilities of Catalonia and to connect them with the territorial institutions to produce, to experience and to exchange knowledge and creativity in a constant circulation of ideas. The participants come as much from the cultural world as from the technological or enterprise world: great cultural centers, territorial cultural centers, universities and finally, companies.

The exchange of audio-visual files will be one of the most usual applications of *l'Anella Cultural*. With this idea it has been provided to the *Anella Cultural* with a storage capacity. In order to maximize the use, a good architecture for the management and distribution of the information is necessary.

The initial scene considers the group of points that form *l' Anella Cultural*. A storage point is formed by a computer plus a FiberChannel/SATA-II cabin.

At the moment *l' Anella Cultural* is formed by 6 points: Granollers, Lleida, Olot, Reus, MediaCAT (Castelldefels) and CCCB (Barcelona). The four first have an approximate capacity of 1.4 TB, whereas both remaining have about 4 TB. Altogether we have a total capacity of 4*2 + 1,4*4 = 13.6 TB. In each of the disc cabins we have occupied half of the slots they have, assuring this way for future extensions.

To each slot we will install an HDD. The centers have cabins with 8 slots, except MediaCAT i CCCB, they have 16 slots in each one.

The *Anella Cultural* wants a system to save all their audio-visual files, the system has to be scalable because they hope to increase their points and clients.

Next picture shows a map of Catalonia, where we can see the location of each point of *l'Anella Cultural* and their storage capacity.
The second picture shows the *Anella Cultural* Network.

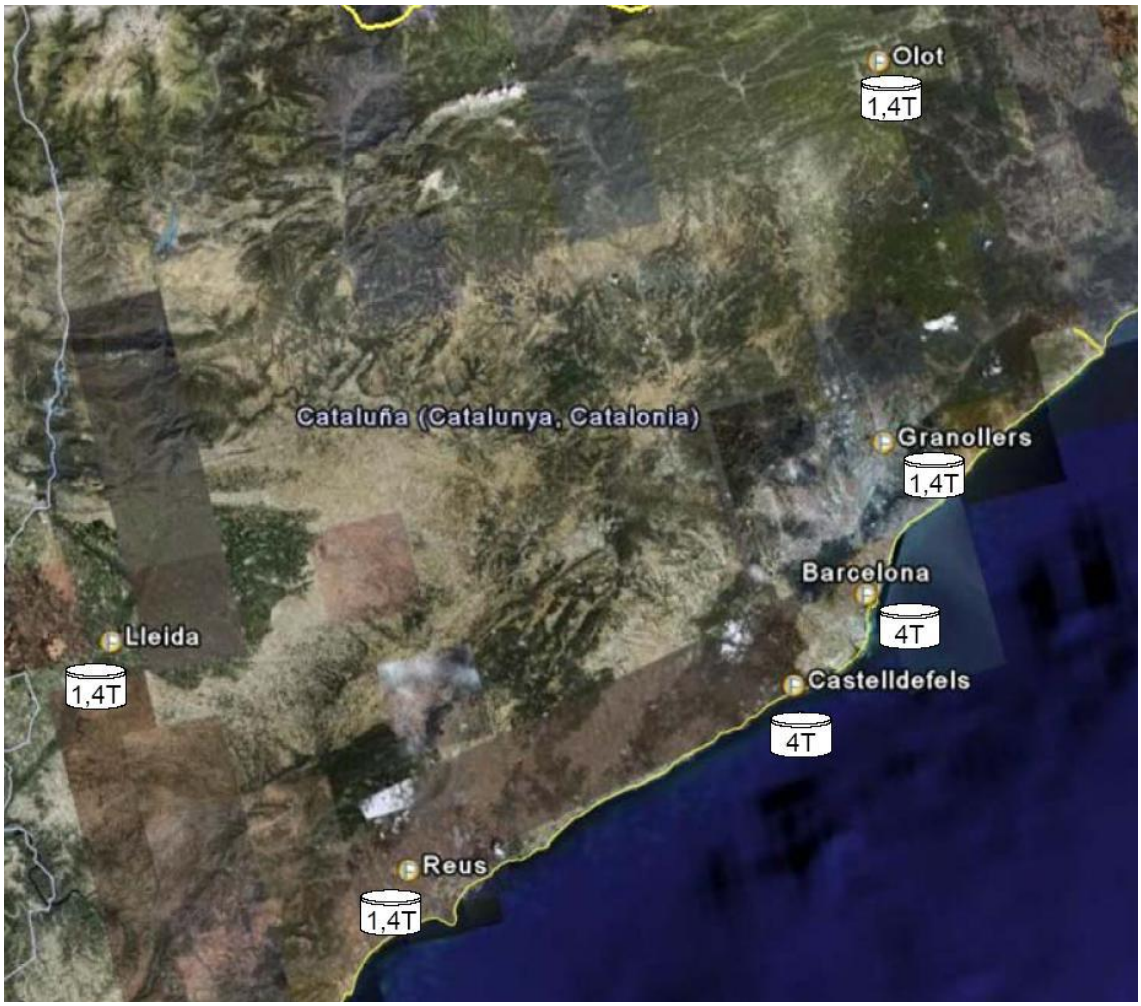


Illustration 1: *Anella Cultural* Scenario

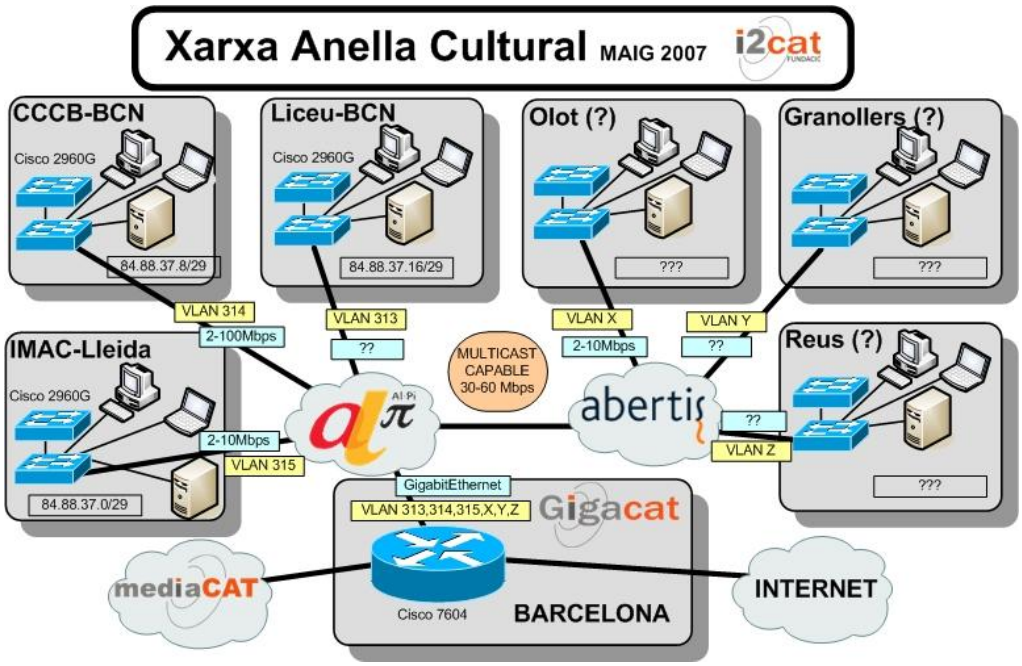


Illustration 2: Network of *l'Anella Cultural*

Introduction

This project is developed in the “Anella Cultural”, project sponsored by Foundation I2CAT with the objective to promote the use of the multimedia technology on IP, in the world of culture and show.

In these surroundings due to the quality of the applications and to their origin, important requirements of storage exist. For this project volumes of storage have been acquired based on the Fiber Channel technology with minimum capacity for 1,4 TB.

The aim of this project is the valuation of different alternatives that allow us to make an efficient management as far as the procedure of storage and to the service rate. The design of a platform and its consequent implementation in the laboratory.

Once implemented, the correct operation will be verified, for it would be desired to make a series of tests that evaluates that the system, verifying the network, cpu, etc.

Before beginning to create the problem we must know several elements that can help the development of the project, that elements are explained in the annex [Reference to chapter].

Next we show an overview of each chapter:

Chapter 1. Objectives and specifications

This chapter lists the specification of *l’Anella Cultural* and the objectives of this project.

Chapter 2. Analisys and Design of the solution

There exists a lot of architecture and software to manage storage, this chapter tries to show which the best project is. For so there is a comparative about the architecture and software.

Chapter 3. Implementation of the platform

This chapter shows how Lustre works, the way to install it step by step, its configuration, some problems that we have been having during the project and their solutions. Besides the scalability is one of the features more important in this project, this chapter also explains how to add a new OST in an existing Lustre file system.

Chapter 4. Test

After mounting the system, we have to check if it works correctly, this section shows the tests that we have been doing and their results.

Chapter 5. Conclusions

Finally, the conclusion describes whether the objectives of project are fulfilled or not, and it has a point with future works.

Chapter 6. Bibliography

The bibliography shows the source of the information of the project.

Annex I. Previous knowledge

This section gives a brief description of some elements that can help the development of the project; these elements are concepts, technologies, software, etc.

Annex II. File System Comparative

This chapter shows a file system comparison, the comparative was done thanks to software that allows us to obtain writing and reading rate. The compared file systems are EXT3, XFS, JFS, and ReiserFS.

Annex III. Installation step by step

This chapter wants to give the exactly commands to install Lustre.

Annex IV. Lustre Commands

This section gives a brief description about the lustre commands.

Annex V. Scripts of Lustre file system

The annex V shows the scripts used in chapter 4.

Annex VI. Graphs of test

This section shows the results obtained in test of chapter 4 too.

CHAPTER 1. Objectives and specifications

Once read the necessity of anella cultural, we have been able to extract a list of requisites minimum that must fulfill the system.

Specifications according to anella cultural:

- A transparent system to the user
  - The user must be able to save and to extract files as if behind any type of mounted platform did not exist.

- Easy to manage
  - The platform must be easily managed and configured.
  - The user just has to see one disk drive.

- Scalable
  - Although at the moment there are 6 nodes, it is possible that in the future more nodes can be added so the system must help to manage this increase of nodes.

- Availability
  - The system must be working the maximum time possible.

- Capacity performance
  - The system must be able to use the maximum of the storage capacity possible.

- A system of low cost
  - As in all the projects, the budget musn't be greater than the necessary.

Considering all these bases the objectives of this project are detailed next.

The project's objectives:

- Analysis and search of a system that fulfills the requirements.
  - We will specify why we have chosen the selected system.

- Design and implementation of a platform with the system selected in the laboratory.
  - With the machines available it will be evaluated which the design that better can be compared to the reality is and we will install the system by learning how it works. With this we will be able to give the support necessary to be able to reproduce it in real surroundings.

- Test of the platform.

- This objective tries to evaluate if the platform fulfills minimum levels of operation.

CHAPTER 2. Analysis and Design of the solution

In this chapter we try to obtain conclusions to be able to select the best solution. In order to do this, we have to know the arquitecture, software we will need to use.

2.1. Architecture

Once analyzed in depth the necessity of anella cultural,the first thing we must think is which type of solution it requires, the system has clients and hard disks and the architecture says to us as they are related among them, with this we mean if the architecture must be, centralized, distributed or hybrid

Next we show a small discussion about each architecture, with this, we will extract a conclusion and final decision about the architecture.

Centralized management

This type of management is the easiest and most intuitive, each node manage and storage their own files. In this case the excessive load of some nodes does not share or is balanced with the deficit of others.

Considering a management centralized we can differentiate two types:
Centralized locally and the centralized globally. In the first case each point manages its own disk drive, without worrying about the use of the rest of units. Nevertheless the second case, the application has access to each unit and it can manage the reading/writen of each node.

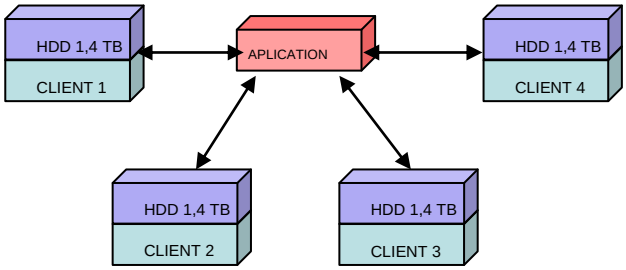


Illustration 2.1: Centralized management locally by an application (WEB, FTP, ...)

In the case of the local management the function of the application is reduced to the diffusion of the contents of each point. Each point is responsible for deciding what contained broadcasts and which one reserves.

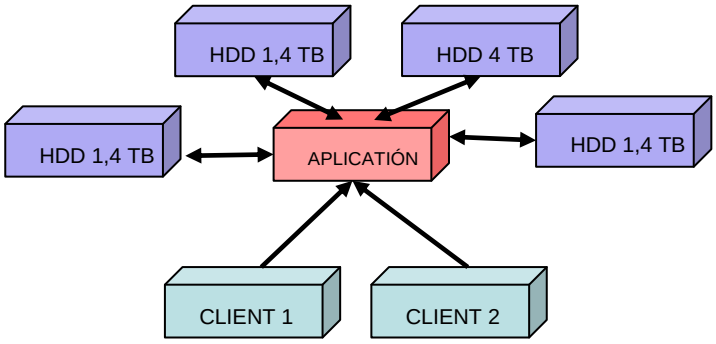


Illustration 2.2: Centralized management globally by an application (WEB, FTP, ...)

In this case the application is the one that has the intelligence of the system. All the clients use the same interface to access the storage, through for example to user names, that decide where to keep the content, or if the user has read/write permissions.

In this type of management the nodes are used following criteria, therefore the most active node can be first in filling its storage. In this case we can define another policy of distribution in order to reduce this effect.

Distributed management

Contrary to the centralized, the distributed has as an objective the collaborative work, of all the points that form *l'Anella Cultural*, in order to form an only storage with the added capacity of each node. In this case the contents are divided and distributed between all the participants. The users accede to an only disk drive. The internal functions of the system must divide and reconstruct the archives in each operation of read/write.

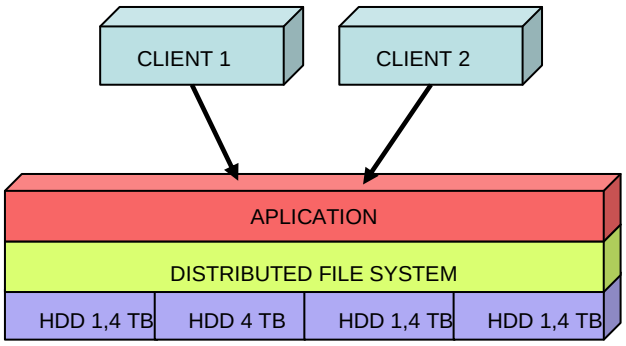


Illustration 2.3: Distributed management

The type of information that keeps in each point is not the results to make n parts of the content, but that follows a pattern, that causes that if we isolated a node, the information does not have sense[1]. From this point of view we can speak about security, considering that this one has to implement at application level. In this case the application must have methods of detecting the permissions of each user on the contents

This management makes a distribution at block level of the stored archives. In this case we divided the archives in blocks and these we distributed them between the nodes that form the system. Being able to maximize the storage capacity.

Hybrid management

The hybrid management consists of taking advantage of the advantages the two previous architectures, maximizes availability and the use of the resources of the system. This type consists of dividing the capacity of each node by half with the intention to implement a centralized and distributed management.

Once in this point it is possible to assign each one different functions:

- One to main and a other to support
- One for communal archives and the other for local archives.

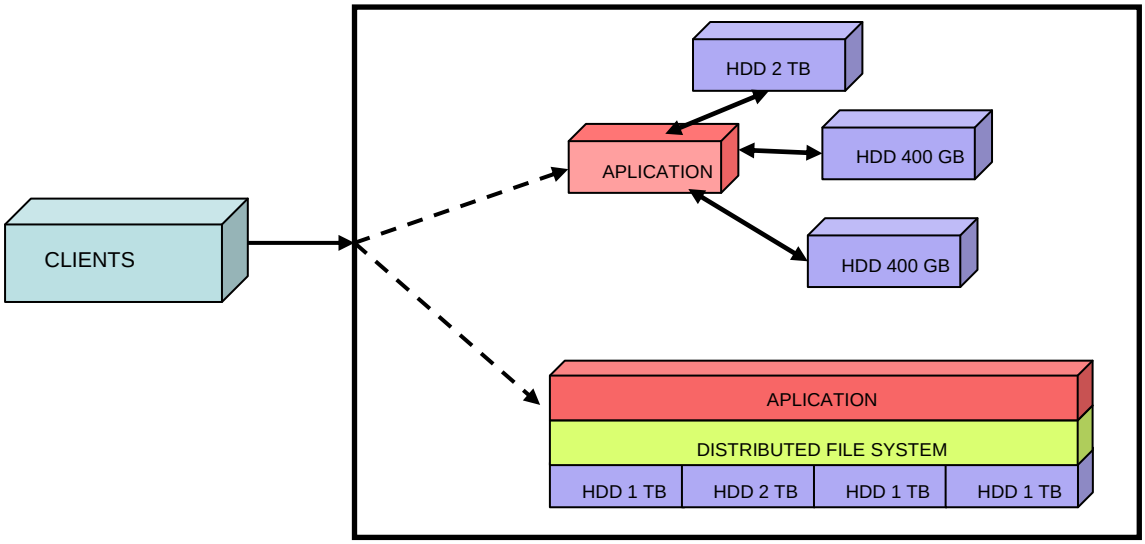


Illustration 2.4: Hybrid management

In this case we have the advantages of each one. If we used the option of a main management plus the support, if we lose a node would be possible to be

[1] Each file in blocks is divided and these are distributed among the different nodes that form the storage network.

continued using the system. In the other option if we lost a node, which it depended on, it would be possible to continue working. The main problem of a hybrid management is that it increases the complexity of the application.

Conclusions

Next we show the conclusions that give off the previous pages:

- The centralized management does not allow an efficient advantage of the storage capacity. However it is not blocked in case of fall of a node.
- The distributed management is blocked when a node has a problem, nevertheless is able to use of efficient form the storage capacity.
- The hybrid management allows us to solve the individual problems of each architecture but it increases the complexity of the system.

As global conclusion we can say that best option form the point of view of the availability is centralized architecture and from the point of view of the capacity the distributed one. Considering the cooperative scope in which *l' Anella Cultural* has been born, the best option is the distributed one too, on the other hand it is especially optimized to work with great archives (that are the case of *l' Anella Cultural*). Finally the effect of which it is sensible to failures of the nodes that form it is subject to the stability of the equipment and the network, so if the equipment is well configured the failure probability is very low.

2.2. Cluster File System Software Comparative

Once we have chosen a distributed architecture, we need software that allows us to create a cluster that it fulfills the requirements of l'Anella Cultural.

A cluster is a group of hard disks, which the clients can see like one with the capacity of all of them.

Besides, the basic requirements that we think they must fulfill good cluster software are:

- Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components.
- Scalability is a desirable property of a system which indicates its ability to be easily increased.
- High availability refers to the ability of the user to access the system.
- High performance is the property to obtain some results with the minimum operations possible.

- Reliability is the ability of a system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances.

In addition to that the software has to be an open source because it is free and in case we need, we can add some funcionality.

For that, a lot of software exists, but we have to select one, thus we have tried to make a small comparative of some software found. [Table 1.1]

Table 1.1: Cluster files system comparative

Features\SW	Lustre	NFS v4	GFS	Ceph	FDT	IBP
Cluster file system	✓	x	✓	✓	x	-
Fault tolerance	✓	-	✓	-	✓	-
Scalability	✓	-	-	✓	-	✓
High availability	✓	✓	✓	-	✓	-
High performance	✓	-	-	✓	✓	✓
Reliable	✓	✓	-	✓	-	✓
Open source	✓	✓	✓	✓	✓	✓
	Panasas	Open AFS	EMC Centera	PVFS2	Intermezzo	
Cluster file system	✓	x	-	x	✓	
Fault tolerance	-	-	-	-	-	
Scalability	✓	✓	✓	-	-	
High availability	✓	-	✓	-	✓	
High performance	✓	-	✓	✓	-	
Reliable	-	-	✓	-	-	
Open source	x	✓	x	✓	✓	
✓ = yes x =no - =?						

The comparative has been done with the available documents, we couldn't read all the information about them. So, the theory information of the table could have some mistake.

If we assume the table is correct, we can see how lustre is the software that shows more information and complies all features.

GFS is software done by RedHat and they recommend using their operative system, otherwise it might not work correctly. Besides, GFS is a good choice for small clusters (<10 nodes) and Lustre is a good choice to medium and large clusters (~100 nodes) although it will work with any size.

NFS has been successful in a variety of enterprise scenarios, it works well at small scale and in low performance environments but does not satisfy the requirements. NFS isn't software to do cluster file system.

Ceph works similar to Lustre but is still being developed, so we reckon it isn't stable. Thus ceph is discarded.

FDT isn't software to do a cluster files system, it is used to copy information from a HDD to another HDD.

IBP seem peer to peer software, because the clients can become IBP depots if they have storage and network, more than cluster software. Besides, the application cannot manage where and when data is stored.

Panasas seems good and stable cluster storage, but it isn't open source, so it has been discarded.

OpenAFS is similar to NFS, so it doesn't usually make a cluster of storage and neither satisfies the requirements.

EMC Centera is a storage system, but is not open source. Thus this solution is discarded.

PVFS doesn't satisfy the requirements because it doesn't usually do cluster storage.

Intermezzo is a new distributed file system, it has been made by the same company that Lustre, *Cluster File System Inc*. but it focus on the high availability more than on the scalability, and scalability is one of the most important features.

Conclusions

With that information we can see how Lustre fulfills all the requirements because it allows scalability, gives the possibility to increase the number of points or the capacity of some node, high availavility, good performance , it is open source and is easy to manage.
Current GFS would also be a good candidate, but is designed to small clusters and doesn't fulfill with the scalability.

CHAPTER 3. Implementation of the platform

In this chapter we want to give an overview about Lustre system, our own experience when we installed it, and how we can configurate the Lustre to each scenario proposed by us. Finally we also explain if we had some difficult to add an OSTs when Lustre is running.

3.1. Lustre

Lustre is a Free Software distributed file system, generally used for large scale cluster computing. The name is a merge of Linux and clusters. The project aims to provide a file system to manage with clusters of tens of thousands of nodes with petabytes of storage capacity, and is available under the GNU GPL.
A Lustre file system has four functional units. These are:

- *Meta data server (MDS)* for storing the metadata.
- *Object storage target (OST)* for storing the actual data.
- *Object storage server (OSS)* for managing the OSTs.
- *Clients* for accessing and the actual usage of the data.

A MDS, OSS, and an OST can be on the same node or on different nodes. Next we show a general scenario of Lustre:

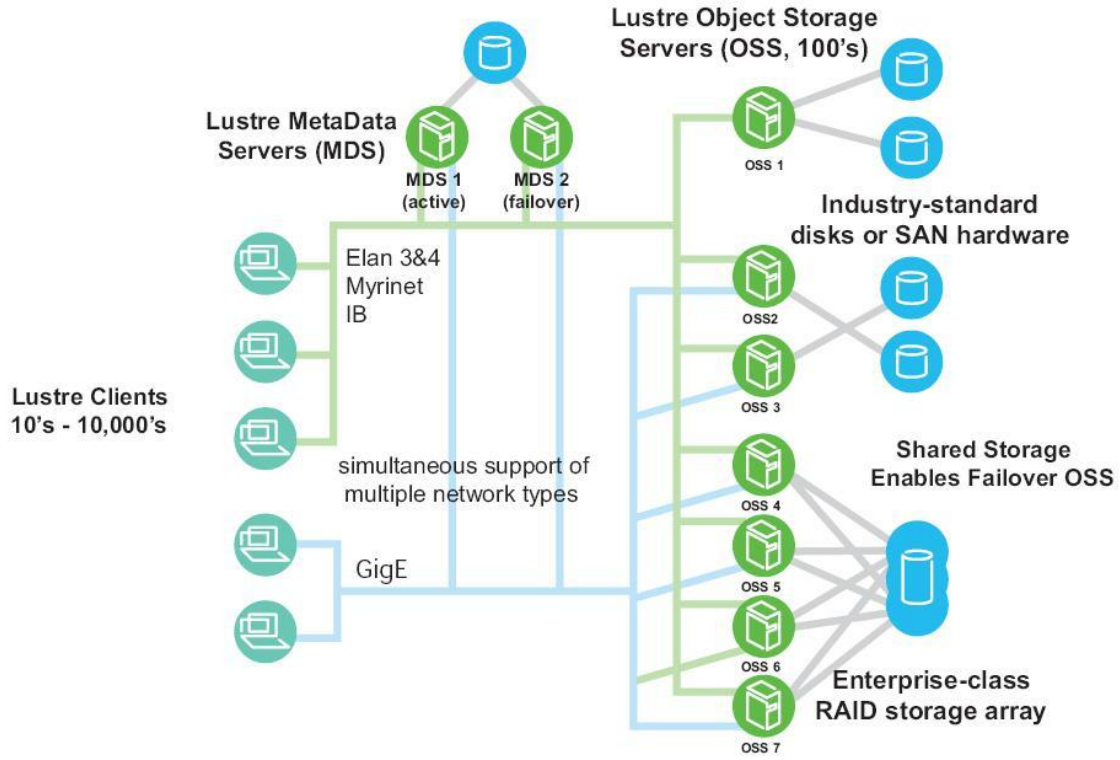


Ilustración 3.1: Lustre Scenario

3.2. Lustre functionality

Lustre treats files as objects that are located through metadata Servers (MDSs). Metadata Servers support all file system namespace operations, such as file lookups, file creation, and file and directory attribute manipulation, directing actual file I/O requests to Object Storage Targets (OSTs). Metadata servers keep a transactional record of file system metadata changes and cluster status, and support failover so that the hardware and network outages that affect one metadata Server do not affect the operation of the file system itself.

Lustre allows some configuration depending on our scenario; it can provide a security and redundancy if we have all the necessary elements.
Now we show some illustrations that explain how Lustre can give availability, fault tolerance and security.

The [Illustration 3.] shows a general functionality between Lustre subsystems, we can see a new element (LDAP), which provides security to the system because the client has to authenticate to get in on Lustre file system.

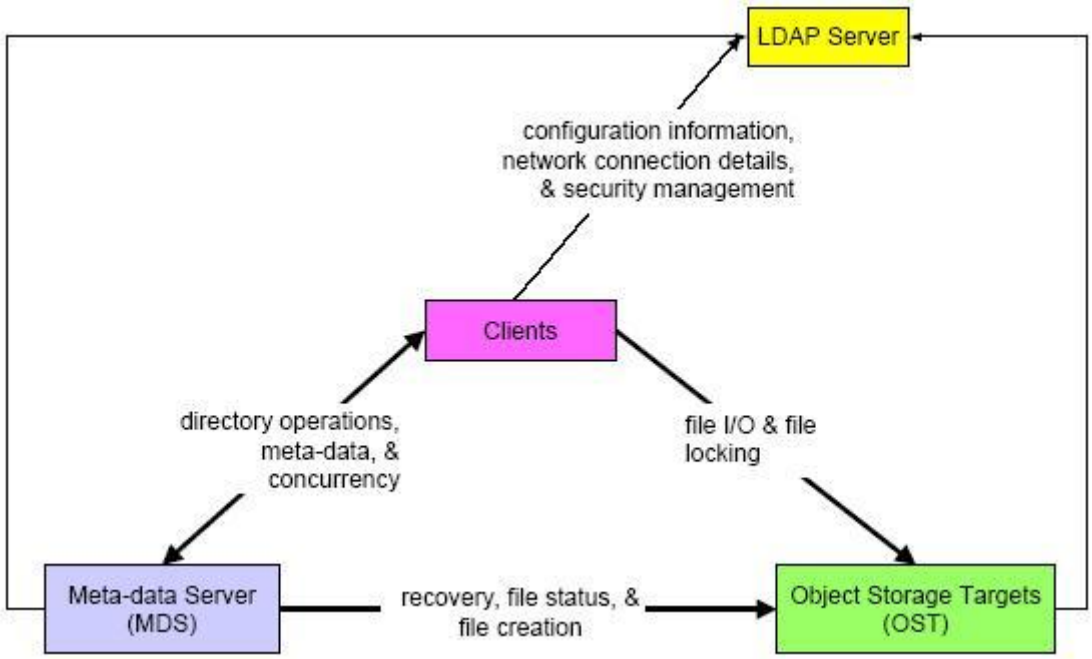


Illustration 3.2: Interactions between Lustre Subsystems

The [Illustration 3.3] shows how a client can read a file step by step with security, the client starts authenticating himself through LDAP, after the client connects with MDS to search the file, then connects to OST and download the file, and finally the client gets a key to decrypt the file through LDAP.

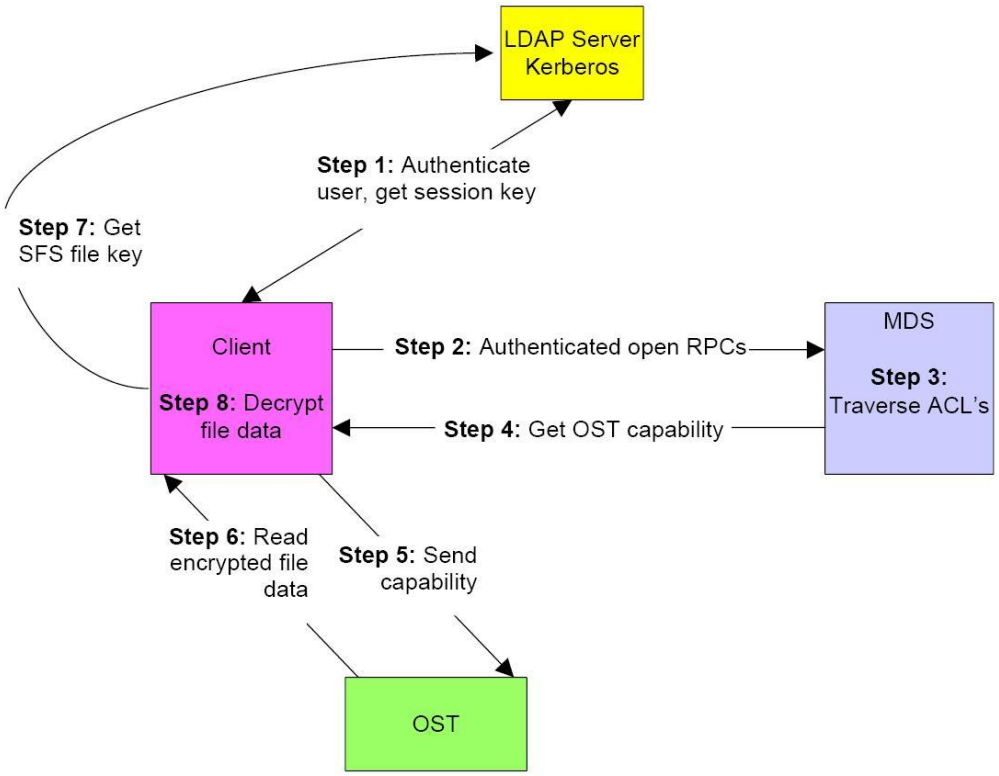


Illustration 3.3: Lustre Read File Security

The [Illustration 3.4] shows how the system of MDS redundancy works when one of them fails.

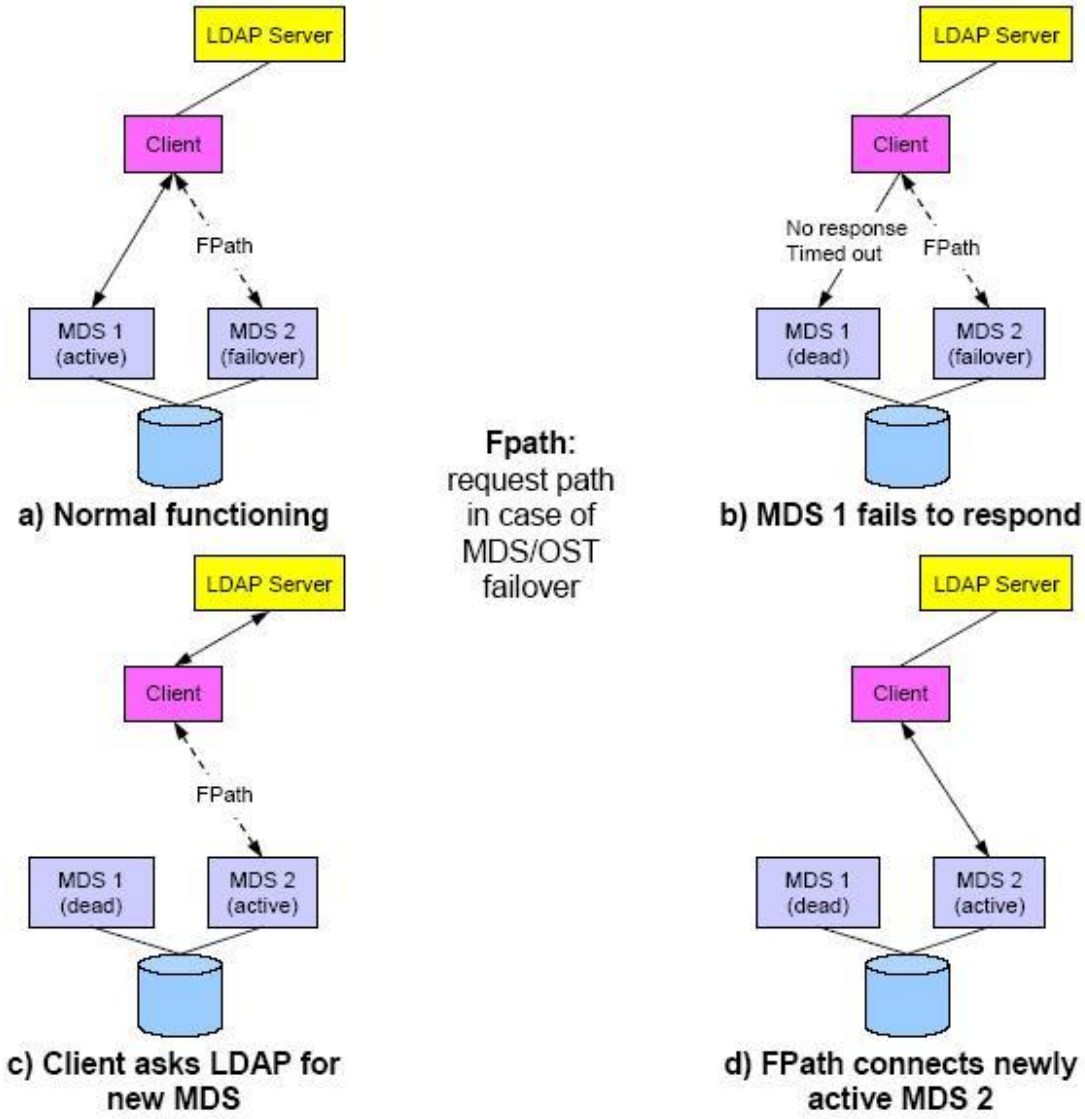


Illustration 3.4: Lustre Failover MDS

In the same way as MDS has an Active/Passive redundancy, OSS has an Active/Active, thus the availability improves.

3.3. Design

The design of *l'Anella Cultural* can be decided at this moment because Lustre doesn't allow many options, although we have to do some tests to give the final results.

- Six OSS, one per every node of *l'Anella Cultural*.

- We don’t know yet, how many OST per OSS. This point will be decided according to the tests.[4.3]
- A MDS in the most stable node.
- A client in the same node that MDS, in machine of client also must be installs some server like FTP.

The design could seem a static scenario, but with Lustre is very easy adding a new node or capacity. [3.6]

3.4. Installation

For a week we tried to install Lustre version beta in a debian with kernel 2.6.18, but we did not manage to do it, in a middle of installation an error occurred. After, we tried with Lustre, the last reliable version, but did not work either.
Finally, we tried installing the reliable version (lustre-1.4.9) in a kernel 2.6.12, and, it was installed with no problem.
We have detailed the installation step by step in the annex [III]

3.5. Lustre Configuration

In this point we want to show the configuration of Lustre and the troubles we have had during the setup.

3.5.1. Playing with Lustre

In order to do this we only needed a machine with lustre.

Illustration 3.5: Deb0 Machine with Lustre

Lustre installation has an example of configuration called *local.sh*, this file is a single system test (MDS, OSS, and client are all in the same system), which is the simplest Lustre installation; it is a configuration in which all three subsystems execute on a single node.

When we tried to execute, an error occurred. The machine could not connect because the NID is not the same as that network.
In order to solve that problem, we had to change the /etc/hosts:

Before:

```
127.0.0.1 localhost deb0
```

After:

```
127.0.0.1 localhost
192.168.48.86 deb0
```

After the change we can run the script *local.sh*, this script makes a new file with the Lustre configuration called *local.xml*, with the lconf command and this file, we can to setup Lustre.

Now we show the commands:

```
$ cd lustre-<release-ver>/lustre/test
$ sh local.sh
$ lconf --reformat --verbose local.xml
```

When the configuration is finished we can verify that the file system has been mounted from the output of df:

Table 3.1: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
inwe_script	46M	8,4M	35M	20%	/mnt/lustre

In order to stop the Lustre configuration, another script called *llmountcleanup.sh* is used.

Next we showed the command:

```
$ NAME=inwe_script sh llmountcleanup.sh
```

Once verified that the example works correctly, we created another configuration in order to learn the operation.

This script is based in *lmc* commands [Annex IV.4], now we explain how we can to create a script:

At the beginning we have to remove the old configuration in order to create a new file with new configuration.

```
$ rm -f local.xml
```

Now we create a node and its network associating a NID, all of this is possible with these commands:

```
$ lmc -o local.xml --add node --node deb
$ lmc -m local.xml --add net --node deb --nid deb --nettype tcp
```

Once the node has been created we can make and configure an mds, to do so we have some arguments like fstype, that allow us to select de protocol, or dev, that allow us to select the device, etc.

```
$ lmc -m local.xml --add mds --node deb --mds mds1 --fstype ldiskfs --dev /tmp/mds1 --size 50000
```

Next, we have to generate the lov, which link mds with the osts, the stripes arguments use to configure where and how we save the data.

```
$ lmc -m local.sh --add lov --lov lov1 --mds mds1 --stripe_sz 1048576 stripe_cnt 0 --stripe_pattern 0
```

Now we are going to create an osts, for that we have to link the ost with a lov and specify a file system and a device.

```
$ lmc -m local.sh --add ost –node deb --lov lov1 –ost ost1 --fstype ldiskfs --dev /tmp/ost1 --size 400000
```

Finally we have to do the mount point, it means mounting the lustre system in a directory, per default in /mnt/lustre.

```
$ lmc -m local.xml --add mtpt --node deb --path /mnt/lustre --mds mds1 --lov lov-test
```

Now we have all configuration save it in local.sh, then we have to execute the *lconf* command [Annex IV.1], as we show in next command:

```
$ lconf --reformat local.xml
```

3.5.2. Single node Lustre

In order to do this we needed a machine with lustre and a storage tray.

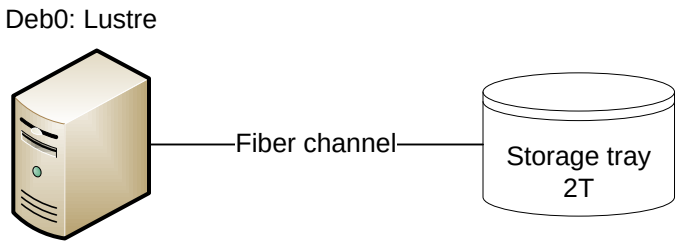


Illustration 3.6: Single Node Lustre Scenario

After playing with the *lmc* commands we created a new script, which is the simplest Lustre installation. It is a configuration in which all three subsystems execute on a single node.

In the script we configured 1MDS, 2 OST and 1 client.

Next, we show the script where we can see some different respect the previous *lmc* commands.

```
#!/bin/sh

if [ "$1" == "reformat"  ];then

        #Create a node:

        rm -f inwe_script.xml

        #lmc -m inwe_script.xml --add node --node deb

        lmc -m inwe_script.xml --add net --node deb --nid deb --nettype tcp
        lmc -m inwe_script.xml --add net --node client --nid '*' --nettype tcp

        #Add the MDS:

        lmc -m inwe_script.xml --format --add mds --node deb --mds mds1 --fstype ext3 --dev /dev/sdb1

        #Add the logical object volume (Note the relationship to the MDS):

        lmc -m inwe_script.xml --add lov --lov lov-test --mds mds1 --stripe_sz 65536 --stripe_cnt -1 --
stripe_pattern 0

        #Add the OSTs: (Note the linkage to the LOV)

        lmc -m inwe_script.xml --add ost --node deb --lov lov-test --ost ost1 --fstype ldiskfs --dev /dev/sdb2
        lmc -m inwe_script.xml --add ost --node deb --lov lov-test --ost ost2 --fstype ldiskfs --dev /dev/sdb3

        #Define the mount point:
        lmc -m inwe_script.xml --add mtpt --node client --path /mnt/lustre --mds mds1 --lov lov-test

        #To configure using LCONF:

        lconf --reformat inwe_script.xml
else
        lconf inwe_script.xml
```

```
fi


mount -t lustre deb:/mds1/client /mnt/lustre
```

There are two ways to execute the script, with argument o without argument.

```
$ sh inwe_script.sh reformat

$ sh inwe_script.sh
```

First command creates a new file XML and formats the osts, and the second command only mounts the system with xml file without formatting the osts.

Finally we mounted the client in /mnt/lustre with last command of script.

After executing the script we tested the correct work with some tests like the command:

```
$ df –h
```

The output of which, we can see in the next chart:

Table 3.2: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
Deb:/mds1/client	1,3G	33M	1,2G	3%	/mnt/lustre

Another test was to know if the mds and osts servers had loaded themselves well. For that we used next command:

```
$ lfs check servers
```

The output of the previous command is a list of the servers with the current state:

```
OSC_deb_ost1_mds1 active.
OSC_deb_ost2_mds1 active.
OSC_deb_ost1_MNT_client-f7345200 active.
OSC_deb_ost2_MNT_client-f7345200 active.
MDC_deb_mds1_MNT_client-f7345200 active.
```

Finally for the last test we saved a big file in a directory in of /mnt/lustre, and then we could see how the folder was striped in the different osts. For that we used next command:

```
$ lfs getstripe /mnt/lustre/stripetest
```

The previous command shows the way a file is saved in each ost. We can see how the osts are active, and in the end the file is saved in ost 0 with identifier 1. Next is the output of the command:

```
OBDS:
0: ost1_UUID ACTIVE
1: ost2_UUID ACTIVE
stripetest/lost.avi
      obdidx       objid       objid       group
           0           1         0x1           0
```

If we want to share the file in all osts, before saving the file we have to configure the folder, to do so, there exists the next command which allows us to configure the stripe, which is the first ost to save the first stripe and how many osts will be used.

```
$ lfs setstripe /mnt/lustre/stripetest 0 -1 2
```

Now if we save again the same file with another name, we can see the new way to save it.

We can see how the file, now, is saved in the 2 osts, in the ost 0 with identifier 2 and in the ost 1 with identifier 1.

```
OBDS:
0: ost1_UUID ACTIVE
1: ost2_UUID ACTIVE
stripetest/lost2.avi
      obdidx       objid       objid       group
           0           2         0x2           0
           1           1         0x1           0
```

In addition we can see how much storage is used in each ost with next command:

```
$ lfs df -h
```

The answer to the previous command is the next chart:

Table 3.3: Output of lfs df -h

UUID	MBytes	Used MB	Available MB	Use %	Mounted on
Mds1_UUID	837.2	64	773.3	7	/mnt/lustre[MDT:0]
Ost1_UUID	645.8	576.6	69.3	89	/mnt/lustre[OST:0]
Ost2_UUID	645.8	224.8	421	34	/mnt/lustre[OST:1]
Filesystem summary	1300	801.3	490.3	62	/mnt/lustre

3.5.3. Multiple node Lustre

In order to do this we needed two machines with lustre and one storage tray for each one, all machines linked through switch.

Like the first contact with Lustre, when we tried to execute, an error occurred. The machine could not connect because the NID is not the same as that network.
In order to solve that problem, we had to change the /etc/hosts again:

```
127.0.0.1 localhost
192.168.48.86 deb0
192.168.48.105 deb1
```

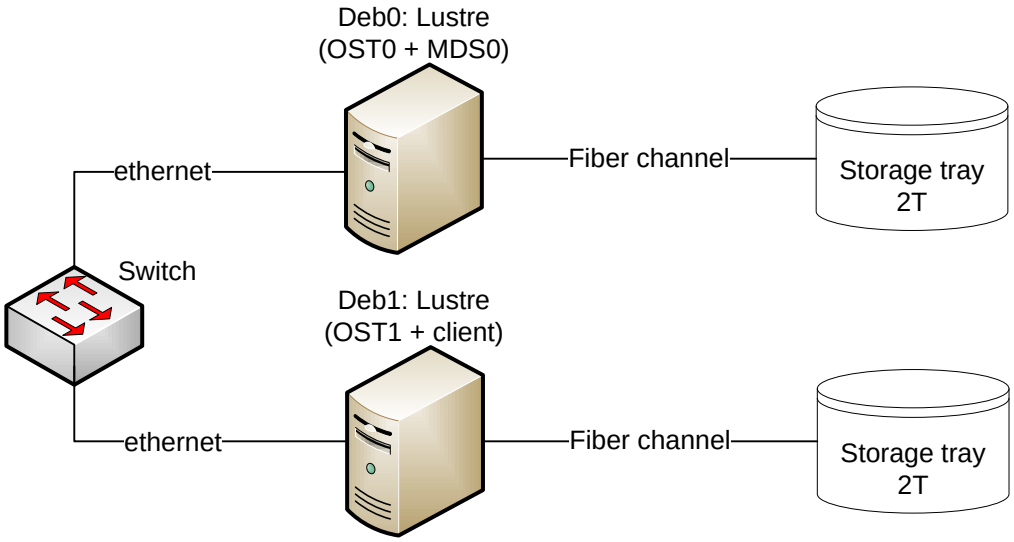


Illustration 3.7: Multiple Node Lustre Scenario

Next step after the single node lustre is multiple node lustre, now we have two machines with lustre, *deb0* and *deb1*, where in deb0 has the mds and one ost and deb1 who has another ost and a client.

Each scenario needs a new script, in the case that the scenario has more than one machine everyone of them needs the same script, therefore we have to share or send the file. In our case we sent the script through *scp* as we show in the next command:

```
$scp multiplesNode.sh root@192.168.48.86:/home/inwe/lustre/lustre-
1.4.9/lustre/tests/pedro/
```

Now, we show the script of the scenario with multiples machines:

```
#!/bin/bash

config="${config:-multiplesNode.xml}"

rm multiplesNode.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

We have to execute the script in each machine with the next command:

```
$ sh multiplesNode.sh
```

Now we already have the file xml with the configuration of script in each machine, then we have to use the lconf command in each machine too, in order to load all the modules of lustre system:

```
$ lconf –reformat multiplesNode.xml
```

Once the lustre is running in all machines, we have to mount the client, which we did in deb1 with next command:

```
$ mount -t lustre deb0:/deb0-mds0/client /mnt/lustre/
```

Now, the system has already been mounted, and is ready to work, the first test that we did to know so, was executing the next command in deb1:

```
$ lfs check servers
```

And its output was:

```
OSC_deb0_deb0-ost0_MNT_client-f7708400 active.
OSC_deb0_deb1-ost1_MNT_client-f7708400 active.
MDC_deb0_deb0-mds0_MNT_client-f7708400 active.
```

We can see how the mds and the osts were working correctly.

After we also executed the next command:

```
$ mount
```

And its output was:

```
/dev/sda1 on / type ext3 (rw,errors=remount-ro)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
deb0:/deb0-mds0/client on /mnt/lustre/ type lustre (rw)
```

We can see in the last line that the client of lustre was mounted correctly.

Next test was saving in /mnt/lustre/stripetest some films trough ftp, therefore we could prove the way of storage and write rate.
Before starting to save, we configured the stripe of folder stripetest as we show in the next command:

```
$ lfs setdtripe /mnt/lustre/stripetest 0 -1 -1
```

This configuration of folder is the default per lustre.

Once the films were in the lustre system we wanted to know how they had been saved, for this we had to execute the next command:

```
$ lfs getstripe /mnt/lustre/stripetest/
```

The output of the previous command is shown next:

```
OBDS:
0: deb0-ost0_UUID ACTIVE
1: deb1-ost1_UUID ACTIVE
/mnt/lustre/stripetest/
default stripe_count: -1 stripe_size: 0 stripe_offset: -1
/mnt/lustre/stripetest//El Ilusionista.avi
      obdidx          objid          objid          group
           0              1            0x1              0
```

```
        1          1          0x1          0

/mnt/lustre/stripetest//Eragon.avi
      obdidx      objid      objid       group
        1          2          0x2          0
        0          2          0x2          0

/mnt/lustre/stripetest//La Maquina de Bailar.avi
      obdidx      objid      objid       group
        0          3          0x3          0
        1          3          0x3          0
```

Also thanks to the next command we can know the available storage in the lustre system:

```
$ df –h
```

The output of which, we can see it in the next table:

Table 3.4: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
Deb0:/deb0-mds0/client	3,7G	2,3G	1,3G	65%	/mnt/lustre

On the other hand, in order to save the films, an ftp server was installed in deb1 because it is the client, to do this, we went to aptitude and downloaded the ftpd, it is an easy ftp server.

The write rate obtained when we uploaded the films through the ftp and connection of 1G/s is approximately 250 Mb/s.
That rate is not a very good result, but in the next chapter we will see some test with special software and we will be able to see the write and read rate more exactly.

3.6. Scalability of Lustre

This point explains how we can add more capacity or a new node if Lustre system is running.

3.6.1. Adding an OST on a New OSS

First at all, we have to change the /etc/hosts of all OSS and MDS, in it we have to add the IP and alias. If not when we tried to execute, an error will occur.

For this test, we ran one MDS, one OST and a client on node *deb0*. Assuming the configuration XML file is called `oldConf.xml`. This XML file is the current configuration that was running before the new OST was added.

Next we show the initial scenario:

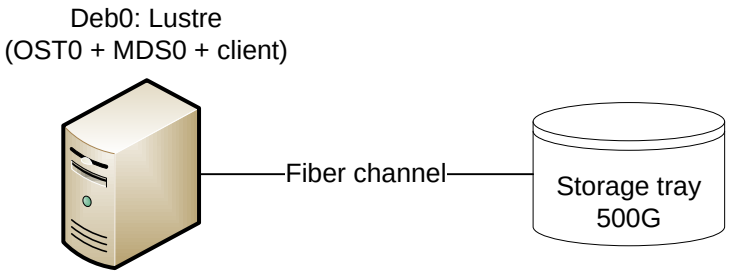


Illustration 3.8: Inicial configuration

The following is the script we used for building the initial Lustre file system.

```
#!/bin/bash

config="${config:-oldConf.xml}"

rm oldConf.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
```

```
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

After mounting the Lustre file system, we configured a folder with default stripe, so we saved in it two files and ran df –h command, below we show the result:

Table 3.5: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
Deb:/mds1/client	447G	5,7G	418G	2%	/mnt/lustre

If we want to know how the file has been saved, we have to execute net command:

```
$ lfs getstripe /mnt/lustre/stripetest
```

The output of previous command is the next we can see that the files have been saved in OST-0.

```
OBDS:
0: deb0-ost0_UUID ACTIVE
/mnt/lustre/stripetest/
default stripe_count: -1 stripe_size: 0 stripe_offset: -1
/mnt/lustre/stripetest// espectaclellarg.m2t
      obdidx        objid        objid         group
         0            1         0x1             0

/mnt/lustre/stripetest// espectacle_africans.m2t

      obdidx        objid        objid         group
         0            2         0x2             0
```

Step 1: Stop the client

We logged into the client node and stopped the client using *umount*.

```
Deb0:$ umount –t lustre deb0:/deb0-mds0/client /mnt/lustre
```

Step 2: Stop the MDS

We logged into the MDS node and stopped the MDS using *lconf*.

```
Deb0:$ lconf -d oldConf.xml
```

We did not shut down the existing OSTs. They can be running or shutdown.

Step 3: Create a new XML configuration file

The new XML file should reflect the added OST on the new OSS (new machine). We added a new OSS, node *deb1*. On that node, we added one OST.

The following script, `newConf.sh`, was used to create the new XML configuration file.

```
#!/bin/bash

config="${config:-newConf.xml}"

rm newConf.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

Step 4: Format the new OST on the new OSS

Only the new OST needs to be reformatted. So, we went to the new node, *deb1*, and ran the following command.

```
Deb1:$ lconf --reformat newConf.xml
```

If the OST resides on a new OSS (a new machine) then only that OST gets reformatted. This keeps the original data intact. However, if you try to use a new OST on an existing OSS, Lustre will format all of the OSTs.

There is a way to use an existing OSS, and that is covered in the next section, adding an OST on an existing OSS.

Step 5: Update the MDS

This is a key step to add the new OST without losing any data on the existing Lustre file system. We used the *lconf* command, but we used an option so it would add the OST to the LOV.

```
Deb0:$ lconf newConf.xml
```

Notice that this command was run on the MDS node, *deb0*.

Step 6: Mount the client

To mount the clients, we just logged into the client node and use the *lconf* command as we did before.

```
Deb0:$ mount –t lustre deb0:/deb0-mds0/client /mnt/lustre
```

If you run the *df -h* command on the file system, you will see that the space has grown.

Table 3.6: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
Deb:/mds1/client	894G	6,1G	841G	1%	/mnt/lustre

In order to compare how the files have been striped we use the same command as before:

```
$ lfs getstripe /mnt/lustre/stripetest
```

We saw that the result was the same, but if we save another file in the system we obtained next result:

```
OBDS:
0: deb0-ost0_UUID ACTIVE
/mnt/lustre/stripetest/
default stripe_count: -1 stripe_size: 0 stripe_offset: -1
/mnt/lustre/stripetest// espectaclellarg.m2t

      obdidx          objid          objid          group
          0              1            0x1              0

/mnt/lustre/stripetest// espectacle_africans.m2t

      obdidx          objid          objid          group
          0              2            0x2              0
```

```
/mnt/lustre/stripetest// espectaclellarg2.m2t
      obdidx        objid        objid         group
           0            3          0x3             0
           1            1          0x1             0
```

We could prove that the stripe configuration is working, because the new file is striped correctly.

The new scenario is next:

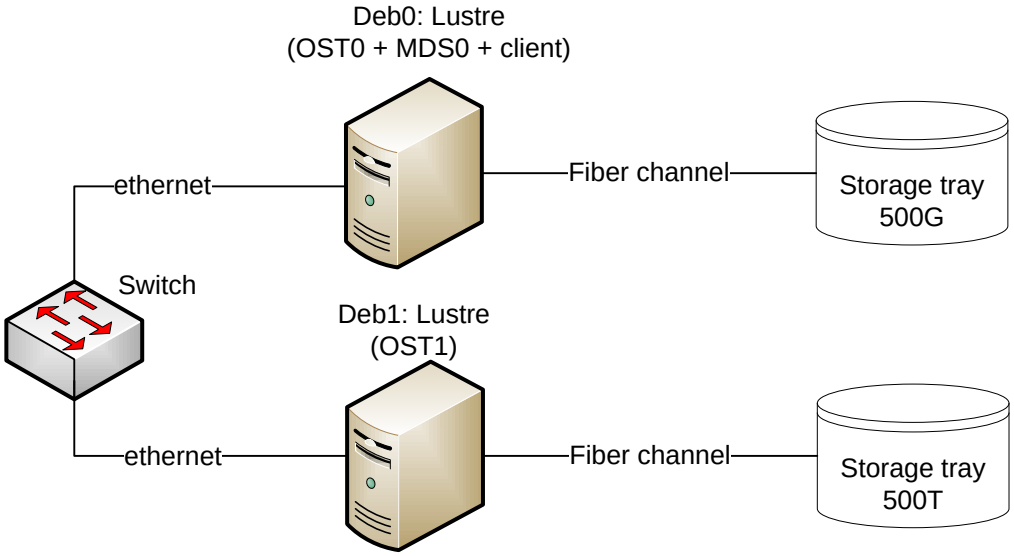


Illustration 3.9: Final configuration

3.6.2. Adding an OST to an Existing OSS

In this case we musn't change the /etc/hosts, because there aren't new nodes.

For this test, we ran one MDS on node *deb0* and one OST (`/dev/sdb1`) on node *deb1* (the OSS). A generic client was run on node *deb0*. We added a new OST that was located on the existing OSS (node *deb1*). The key to do this work is to use a new name for the OSS for only the new OST. Otherwise, when you reformat the new OST, Lustre will reformat all of the OSTs.

Next we show the initial scenario:

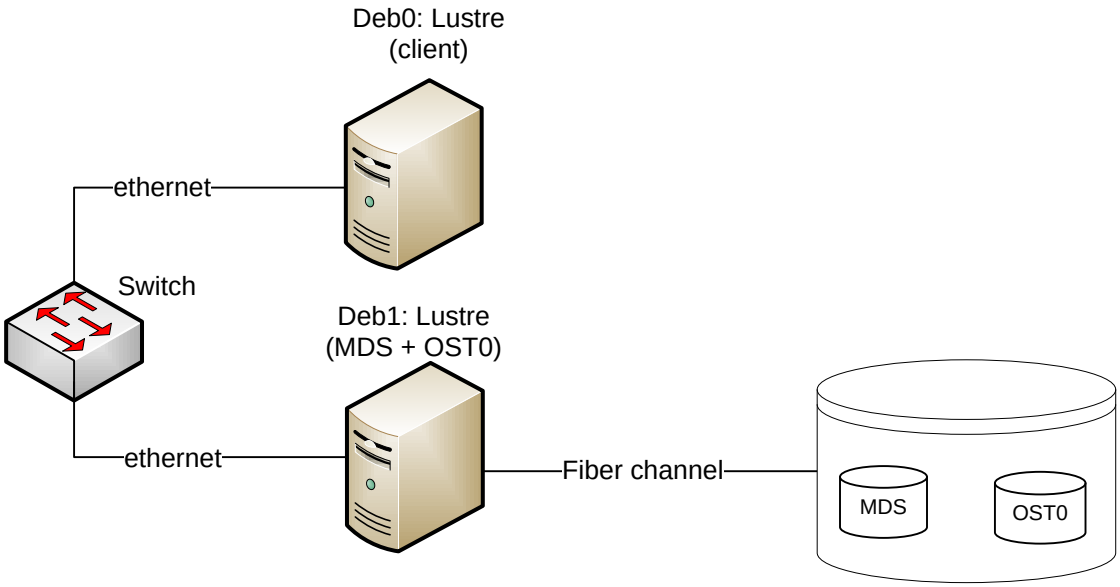


Illustration 3.10: Initial configuration

Assume the configuration XML file is called `oldConf2.xml`. This XML file is the current configuration that was running before the new OST was added.

We used the following for building the initial Lustre file system.

```
#!/bin/bash

config="${config:-oldConf2.xml}"

rm oldConf2.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1
```

```
# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

After mounting the Lustre file system, we configured a folder with default stripe, so we saved in it two files and ran df –h command, below we show the result:

Table 3.7: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
Deb:/mds1/client	447G	13G	412G	3%	/mnt/lustre

If we want to know how the file has been saved, we have to execute net command:

```
$ lfs getstripe /mnt/lustre/stripetest
```

The output of previous command is as it follows, we can see that the files have been saved in OST-0.

```
OBDS:
0: deb0-ost0_UUID ACTIVE
/mnt/lustre/stripetest/
default stripe_count: -1 stripe_size: 0 stripe_offset: -1
/mnt/lustre/stripetest// espectaclellarg.m2t
      obdidx          objid          objid          group
          0              1          0x1              0

/mnt/lustre/stripetest// espectacle_africans.m2t

      obdidx          objid          objid          group
          0              2          0x2              0
```

Step 1: Stop the client

We logged into the client node and stopped the client using *umount*.

```
Deb0:$ umount –t lustre deb0:/deb0-mds0/client /mnt/lustre
```

Step 2: Stop the MDS

We logged into the MDS node and stopped the MDS using *lconf.*

```
Deb0:$ lconf -d oldConf2.xml
```

We did not shut down the existing OSTs. They can be running or shutdown.

Step 3: Create a new XML configuration file

The new XML file should reflect the added OST on the existing. We used an existing OSS, node *deb1*. On that node we added one OST, `/dev/sdb2`. We

used the following script, `newConf2.sh`, to create the new XML configuration file.

```
#!/bin/bash

config="${config:-newConf2.xml}"

rm newConf2.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node deb1a
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp
lmc -m $config --add net --node deb1a --nid deb1 --nettype tcp


# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1
lmc -m $config --add ost --node deb1a --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb1-ost2

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

This script is different from the previous one. We defined a new node called *deb1a* in the Create Nodes section of the script.

```
$ lmc -m $config --add node --node deb1a
```

We also added a network to this node.

```
$ lmc -m $config --add net --node deb1a --nid deb1 --nettype tcp
```

But notice that the *nid* is the same for this node.

Everything else was the same until we defined the OSTs:

```
$ lmc -m $config --add ost --node deb1a --lov lov1 --fstype ldiskfs --dev
/dev/sdb2 --ost deb1-ost2
```

In this case, we used the new node name, *deb1a*, and a new OST name, *deb1-ost2*. We also used a new device, `/dev/sdb2`, for this new OST. Finally we ran the script to create the new XML file.

Step 4: Format the new OST on the new OSS
Only the new OST needs to be reformatted. So, we went to the existing node, *deb1*, and ran the following command.

```
Deb1:$ lconf –reformat –node deb1a newConf2.xml
```

Because Lustre thinks that the new OST is on a new OSS, *deb1a*, it will reformat only that OST.

Step 5: Update the MDS

This is a key step to adding the new OST without losing any data on the existing Lustre file system. We used the *lconf* command, but we used an option so it would add the OST to the LOV.

```
Deb0:$ lconf  --write_conf newConf2.xml

Deb0:$ lconf  newConf2.xml
```

Notice that this command was run on the MDS node, *deb0*.

Step 7: Mount the client

To mount the clients, we just logged into the client node and use the `lconf` command as we did before.

```
Deb0:$ mount –t lustre deb0:/deb0-mds0/client /mnt/lustre
```

If you run the *df -h* command on the file system, you will see that the space has grown.

Table 3.8: Output of df

File system	Size	Used	Available	Use%	Mounted en
dev/sda1	19G	2,2G	16G	13%	/
tmpfs	1013M	0	1013M	0%	/dev/shm
Deb:/mds1/client	894G	13G	835G	2%	/mnt/lustre

In order to compare how the files have been striped we use the same command as before:

```
$ lfs getstripe /mnt/lustre/stripetest
```

We saw that the result was the same, but if we save another file in the system we obtained the next result:

```
OBDS:
0: deb0-ost0_UUID ACTIVE
/mnt/lustre/stripetest/
default stripe_count: -1 stripe_size: 0 stripe_offset: -1
/mnt/lustre/stripetest// espectaclellarg.m2t

      obdidx       objid       objid       group
           0           1         0x1           0

/mnt/lustre/stripetest// espectacle_africans.m2t

      obdidx       objid       objid       group
           0           2         0x2           0

/mnt/lustre/stripetest// espectaclellarg2.m2t
      obdidx       objid       objid       group
           0           3         0x3           0
           1           1         0x1           0
```

We could prove that the stripe configuration is working, because the new file is striped correctly.

The new scenario is next:

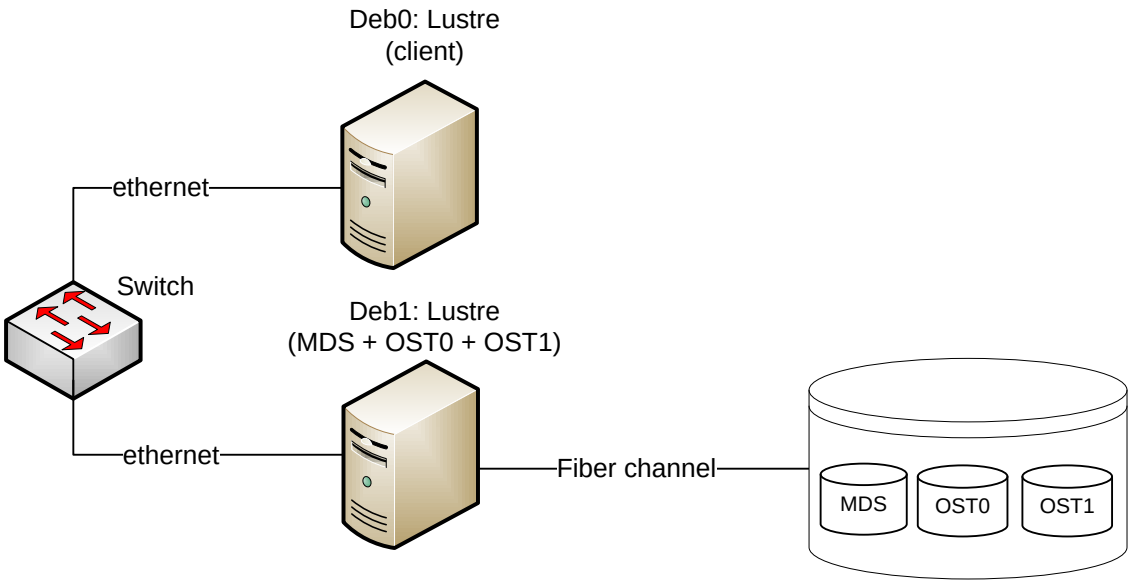


Illustration 3.11: Final configuration

CHAPTER 4. Test

In this chapter we are going to evaluate the Lustre file system, to do so, we created different scenarios and obtained some characteristics of cpu and network, with that we will can decide which is the best scenario to *l'Anella Cultural*.

In each scenario we did the same test, which consisted of copying a file of 8GB from the folder in the same PC where the client of Lustre is, to another folder that is where the Lustre file system has been mounted.

Before starting with the test, we wanted to compare the time between copying the file into the Lustre file system to copying the file in another partition, and the result was the next:

- Folder of client → Lustre system
  - 2min 6 seconds
- Folder of client → to another partition
  - 5min 30 seconds

At the beginning these results were rare, but after we arrived to the conclusion that it could be that sendind the information by Ethernet is faster than saving it in the hard disk.

Now we are going to describe the test step by step:

First at all, we have to save a file of 8GB in the /root/ in the PC of client of Lustre, and then we have to configure the scenario where we run the *sar*[2] in each OSS in order to obtain all the parameters and later to be able to create graphs with the gathered data.

The command to run the *sar* is shown next:

```
$ sar -ucdr -n DEV -n SOCK -I SUM -o prueba 2 150
```

Once the data has been gathered we have to change and prepare the file to be able to create the graphs. We use the next command to do that:

```
$ sar -ucdr -n DEV -n SOCK -I SUM -f prueba > prueba.sar
```

Finally we have to create the graphs with the next command:

```
$ ./sar2gp -f ./prueba.sar -t sar700 -d netd4 netd5 -d2 cpu5 cpu7 -h
./prueba.html -b –v
```

[2] Monitor software

In each scenario the previous command could have been slightly modified according to the necessities of each one.

4.1. Scenarios

The scenarios have been created with two machines and two storages cabins, the interconnection between machines is a gigabit switch. With this equipment we configure next scenarios:

- 1OSS+1OST+1Client

In order to realize this scenario we configured a *MDS* and an OST in *deb0* and a client in *deb1*. We can find the script with the configuration in annex [V.1].
Next we show a schematic scenario:

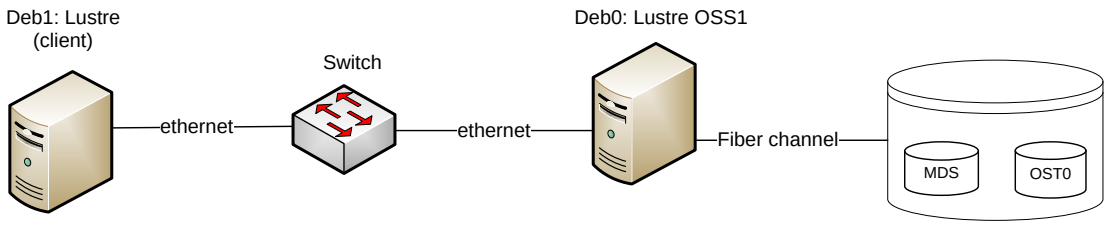


Illustration 4.1: Scenario 1OSS+1OST+1client

In addition, in this scenario we did more test, we proved a different behaviour if we configured a different stripe size:

Next we show the command to configure the stripesize:

- Stripe size 50MB
  $ lfs setstripe /mn/lustre/striptest 52428800 -1 -1
- Stripe size 500MB
  $ lfs setstripe /mn/lustre/striptest 524288000 -1 -1
- Stripe size 1GB
  $ lfs setstripe /mn/lustre/striptest 1073741824 -1 -1

- 1OSS+2OST+1Client

In order to realize this scenario we configured a *MDS* and two OST in *deb0* and a client in *deb1*. We can find the script with the configuration in annex [V.2].
Next we show a schematic scenario:

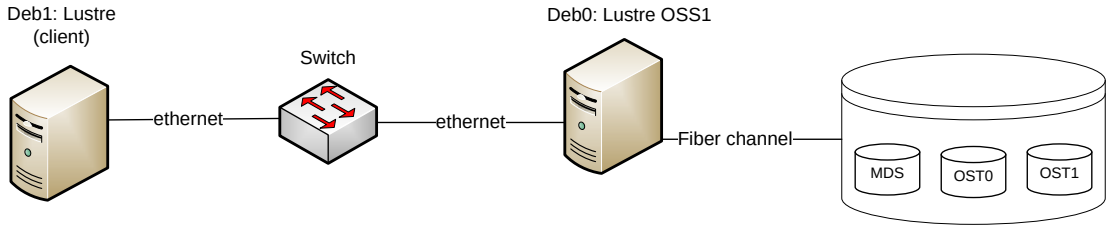


Illustration 4.2: Scenario 1OSS+2OST+1client

- 1OSS+3OST+1Client

In order to realize this scenario we configured a *MDS* and three OST in *deb0* and a client in *deb1*. We can find the script with the configuration in annex [V.3].
Next we show a schematic scenario:

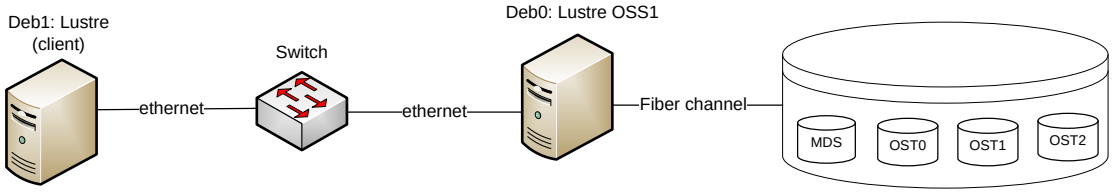


Illustration 4.3: Scenario 1OSS+3OST+1client

- 2OSS+2OST+1Client

In order to realize this scenario we configured a *MDS* and an OST in *deb0* and a client and another OST in *deb1*. We can find the script with the configuration in annex [V.4].
Next we show a schematic scenario:

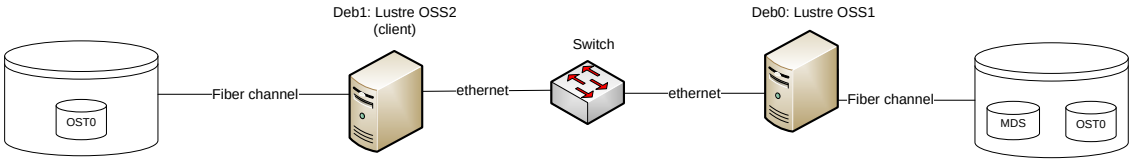


Illustration 4.4: Scenario 2OSS+2OST+1client

- 2OSS+3OST+1Client

In order to realize this scenario we configured a *MDS* and two OST in *deb0* and a client and another OST in *deb1*. We can find the script with the configuration in annex [V.5].
Next we show a schematic scenario:

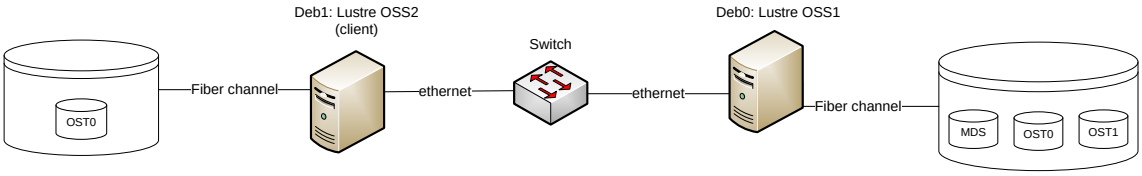


Illustration 4.5: Scenario 2OSS+3OST+1client

- 2OSS+4OST+1Client

In order to realize this scenario we configured a *MDS* and two OST in *deb0* and a client and two more OST in *deb1*. We can find the script with the configuration in annex [V.6].
Next we show a schematic scenario:

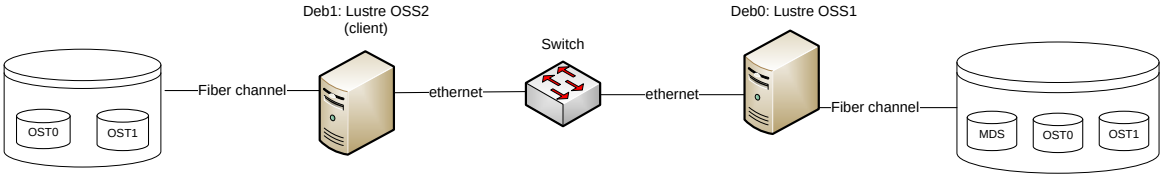


Illustration 4.6: Scenario 2OSS+4OST+1client

4.2. Results

We identified which are the characteristics that allowed us to value if the operation of the system is correct or no. Then we compared these results.

At the same time we compared if there were any differences among scenarios.
We can see all the graphs in annex [VI].

CPU results

If we see all the graphs of CPU, we can observe that in every case the client or *deb1* has less loads than *deb0*.
It could be due to the client having to wait some time to write, we think so because in the client, the waiting of CPU due to writing is bigger than *deb0*.

In the scenarios with just OSS, we can see how the load of CPU is always at 40% in the client PC, while in the OSS machine, if we increase in more OSTs the load of CPU falls.
In the same way as the previous observation, it's because the CPU has to wait when it is written in the disc.

We can see that conclusion in next illustration:

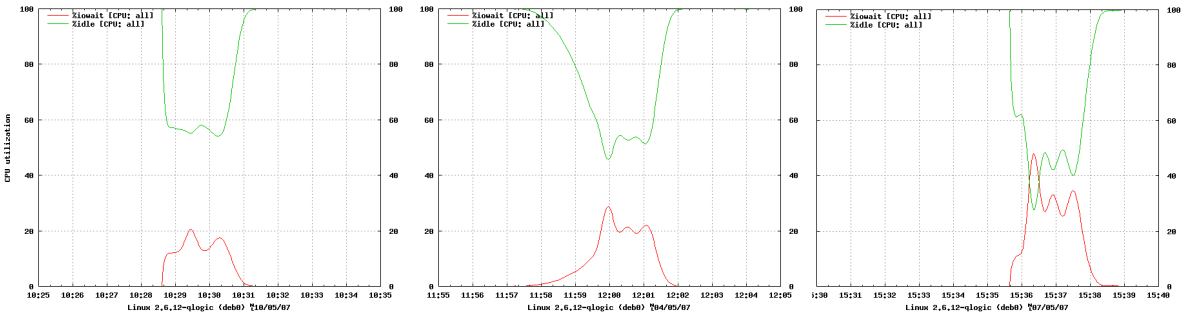


Illustration 4.7: 1OST 2OST 3OST

Regarding to the scenario with two OSS we have to say that in the client machine or *deb1* the load of CPU is again around of 40% in every case.
In this scenario if we increase the number of OSTs, it does not modify the load of CPU, which in every case is around of 80%.
We can see that conclusion in the next illustration:

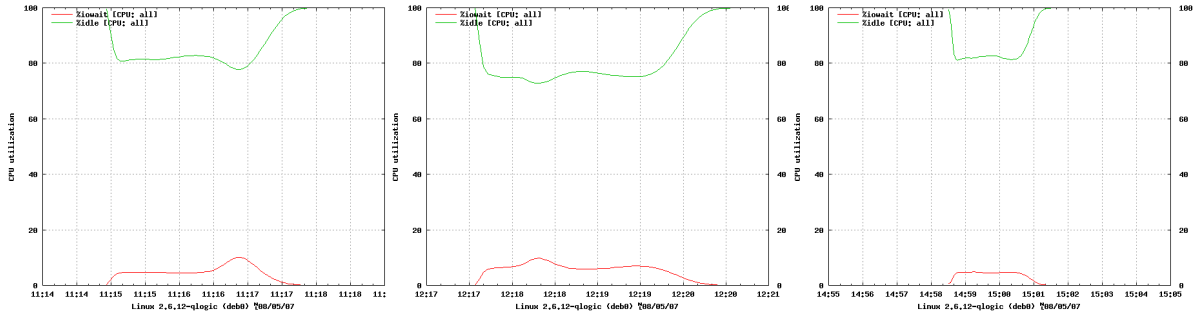


Illustration 4.8: 2OST 3OST 4OST

Network results

If we see all the graphs of the network, we can see the different rates between the scenarios with one OSS or two OSSs.
In the case of just OSS the rate in all of cases is around the 560Mbps whereas in the other scenario the rate falls around the 320Mbps.
This answer could be because in the second case the client has to send and save at the same time, so it can't send faster than in the first case where the client just has to send data to *deb0*.

Between the machines there aren't great differences, because it is normal that the transmission rate in client node is the same as the reception rate in the *deb0* node.

We can see all of conclusion in the graphs in the annex [VI].

Stripe Size results

According to the obtained graphs of the tests done, the fact of changing stripe size does not alter the behavior of the machines in CPU or rate of network.

We can see this conclusion in the graphs in the annex [VI].

4.3. Conclusions

According to the results, the CPU and network change with number of OSS, however we can not decide the numbers of OSS, we need one per point of *Anella Cultural*.
On the other hand we can select the number of OST per OSS and if we look at the graphs, we can see that the different of rate among the number of OSTs are very similar. Then it is easier to configure one OST per OSS than two or three per OSS.

Finally, the proposed scenario according to the results is:

- Six OSS, one per every node of *l'Anella Cultural*.
- One OST per OSS.
- A MDS in the node most stable.
- A client in the same node that MDS, in the machine of client a server like FTP.must also be installed.

CHAPTER 5. Conclusions

The project has been made without many problems; therefore we think that it has fulfilled its objectives.

We evaluated different alternatives that would allow us to make an efficient management of storage of easy way, and we selected Lustre.
After, we designed a platform for the *Anella Cultural*, thinking of the real scenario, and then mounted a scale model in the laboratory in order to test their correct performance.

Once the project has been finished, the *Anella Cultural* knows that Lustre is good solution for their scenario. So, now they can decide whether they want to mount the real scenario or not.

5.1. Environmental impact

This project does not affect the environment; it just tries to use software to fulfill a necessity.
The only thing that can affect is the power consumption of the machines, because the scenario needs several PCs where they must be operative at any moment.

5.2. Future works

The future works are:

- A new release could be installed.
  - During the project, a new stable version was created, we propose to install the new release.

- It could tune the network to improve the rate.
  - We think that if we configure the network with jumbo frames, we will get to improve the network rate.

- It could do some test with Iozone and run clients at the same time.

- It could do some tests to try the storage performance.
  - This test consists in configure the OSTs with different capacities and see if Lustre loses their storage performance.

CHAPTER 6. Bibliography

[1] D. R. Alexander, C. Kerner, J. Kuehn, J. Layton, P. Lucas, H. Ong, S. Oral, L. Stein, J. Schroeder, S. Woods, S. Studham, “A How-To Guide for Installing and Configuring Lustre”,May 2005. [Document online] URL: http://www.lustre.org

[2] Cluster file system Inc., “Lustre Whitepaper Version 1.0”, November 2002. [Document online] URL: http://www.lustre.org

[3] Cluster file system Inc., “Lustre 1.4.8 Operations Manual”, February 2007. [Document online] URL: http://www.lustre.org

[4] Cluster file system Inc., “Selecting a Scalable Cluster File System”, November 2005. [Document online] URL: http://www.lustre.org

[5] http://wikipedia.org/

IBP
[6] http://loci.cs.utk.edu/
Panasas
[7] http://www.panasas.com/activestor.html
OpenAFS
[8] http://openafs.org/
EMC Centera
[9] http://www.emc.com/products/systems/centera.jsp
PVFS2
[10] http://www.pvfs.org/
Intermezzo
[11] http://www.inter-mezzo.org/
Global file system
[12] http://www.redhat.com/software/rha/gfs/
[13] http://wikipedia.org/
Fiber channel:
[14] http://www.iol.unh.edu/services/testing/fc/training/tutorials/fc_tutorial.php
fstab:
[15] http://www.tuxfiles.org/linuxhelp/fstab.html
Zabbix:
[16] http://www.zabbix.com/index.php
IOzone:
[17] http://www.iozone.org/
I/O process:
[18] http://tldp.org/LDP/sag/html/buffer-cache.html

ANNEX I. Previous knowledge

I.1 Fiber channel:

Fiber channel is a gigabit-speed network technology primarily used for storage networking. Despite its name, Fiber Channel signaling can run on both twisted-pair copper wire and fiber optic cables.
There are three major Fiber Channel topologies: *Point to Point, Loop,* and *Switched fabric* [Illustration I.3]

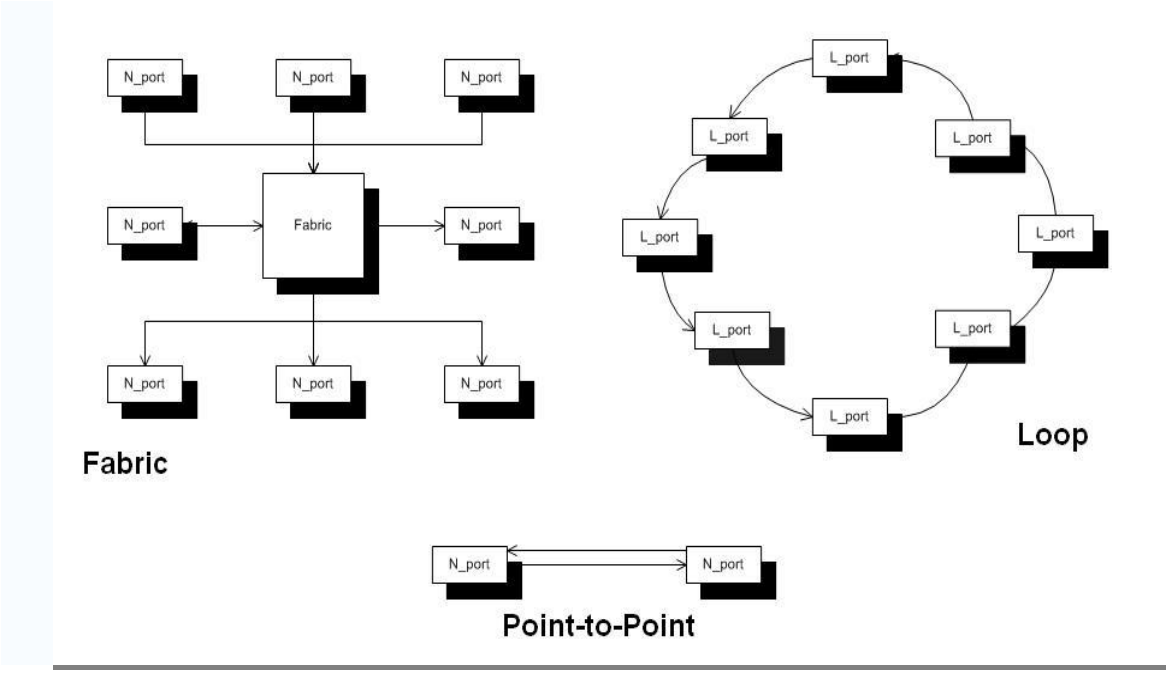


Illustration I.3: Fiber Channel Topologies

Fiber Channel has 5 layers, namely:

- **FC0** The physical layer, which includes cables, fiber optics, connectors, pinouts, etc.
- **FC1** The data link layer, which implements the 8b/10b encoding and decoding of signals.
- **FC2** The network layer, defined by the FC-PI-2 standard, consists of the core of Fiber Channel, and defines the main protocols.
- **FC3** The common services layer, a thin layer that could eventually implement functions like encryption or RAID.
- **FC4** The Protocol Mapping layer. Layer in which other protocols, such as SCSI, are encapsulated into an information unit for delivery to FC2.

I.2 SCSI:

SCSI (Small Computer System Interface) [Illustration I.4] is a set of standards for physically connecting and transferring data between computers and peripheral devices.
The primary objective of SCSI is to provide an independent mechanical device to attach and access devices to host computers. SCSI is designed to provide an efficient peer-to-peer I/O bus that supports multiple devices, including one or more hosts. Thus, through a single SCSI interface different disk drives, tape drives, printers, optical media drives, and other devices can be added to the host computers without requiring modifications to generic system hardware or software.

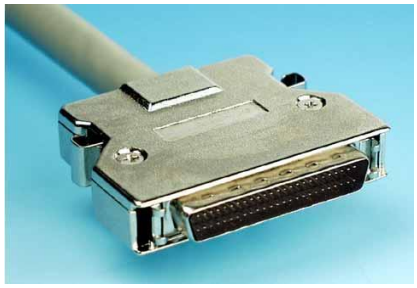


Illustration I.4: SCSI connector

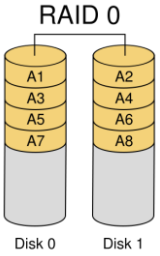
The following varieties of SCSI are currently implemented:

- **SCSI-1:** Uses an 8-bit bus, and supports data rates of 4 MBps
- **SCSI-2:** Same as SCSI-1, but uses a 50-pin connector instead of a 25-pin connector, and supports multiple devices. This is what most people mean when they refer to plain *SCSI*.
- **Wide SCSI:** Uses a wider cable (168 cable lines to 68 pins) to support 16-bit transfers.
- **Fast SCSI:** Uses an 8-bit bus, but doubles the clock rate to support data rates of 10 MBps.
- **Fast Wide SCSI:** Uses a 16-bit bus and supports data rates of 20 MBps.
- **Ultra SCSI:** Uses an 8-bit bus, and supports data rates of 20 MBps.
- **SCSI-3:** Uses a 16-bit bus and supports data rates of 40 MBps. Also called *Ultra Wide SCSI.*
- **Ultra2 SCSI:** Uses an 8-bit bus and supports data rates of 40 MBps.
- **Wide Ultra2 SCSI:** Uses a 16-bit bus and supports data rates of 80 MBps.

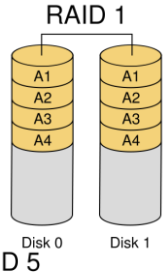
I.3 Raid:

RAID (originally redundant array of inexpensive disks, also known as redundant array of independent disks) refers to a data storage scheme using multiple hard drives to share or replicate data among the drives. Depending on the configuration of the RAID (typically referred to as the RAID level), the benefit of RAID is to increase data integrity, fault-tolerance, throughput or capacity, compared with single drives.

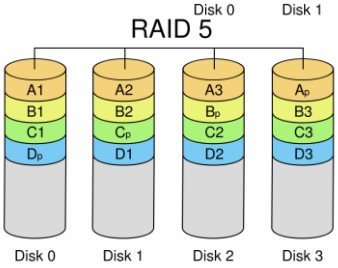
There are a lot of Raid levels, but we just explain the most used:



- Raid 0 (also known as a stripe set or striped volume) splits data evenly across two or more disks without parity information for redundancy, thus it doesn't offer fault-tolerance, (if there is some fault, the information would lose and you had recovered it from a backup).



- Raid 1 creates an exact copy (or mirror) of a set of data on two or more disks. This is useful when read or reliability is more important than data storage capacity.



- Raid 5 uses block-level striping with parity data distributed across all member disks. RAID 5 has achieved popularity due to its low cost of redundancy. Generally, RAID 5 is implemented with hardware support for parity calculations. A minimum of 3 disks is generally required for a complete RAID 5 configuration.

I.4 File system:

File system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them.
Ext3 is the default file system for many popular Linux distributions, is like ext2 but ext3 add the journaling (is like a log).
Ext3 use i-nodes to manage the data, i-node is a data structure which saves and stores basic information about a regular file, directory, or other file system object. Besides, it has a reference that indicates the file location in the hard disk. [Illustration I.5]

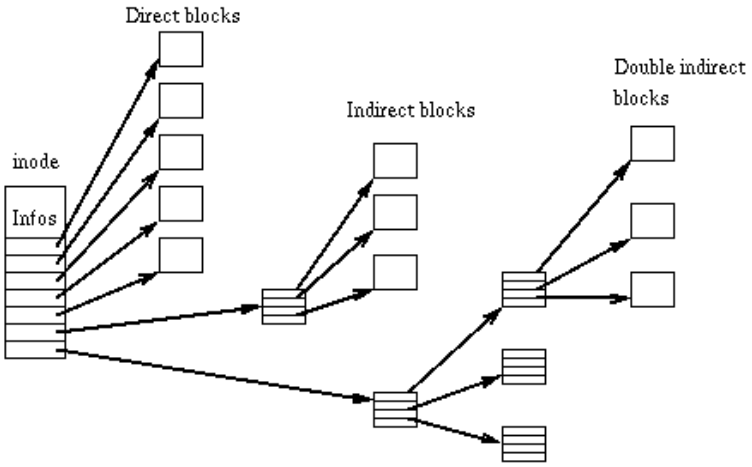


Illustration I.5: i-node structure

I.5 Kernel 2.6

The kernel is the central component of most computer operating systems (OSs). Its responsibilities include managing the system's resources and the communication between hardware and software components. [Illustration I.6]
The version 2.6 is the latest one, and was done in 2003, so kernel 2.6 is the version with more functionalities and hardware supports.

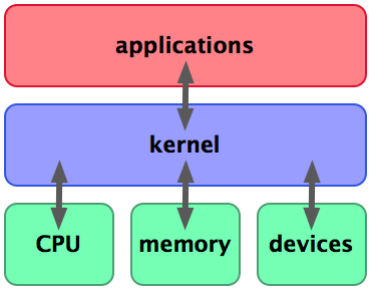


Illustration I.6: UNIX Architecture

Kernel of Linux has a modular design. When the system is booting, it just loads a part of kernel, so when user needs some characteristics that kernel doesn't have, the system will load a new module of kernel automatically. After a specific period of inactivity, the module will be erased of the memory.
The initial ramdisk or initrd is a temporary file system used by the Linux kernel during the boot. The initrd is typically used for making preparations before the real root file system can be mounted.

I.6 Tuning the file system:

Tuning the file system is the way to try to improve the system setting some parameters, for that, there exists some software like hdparm and tune2fs.

- Hdparm is a command line utility for the Linux operating system to set and view IDE hard disk hardware parameters. It can set parameters such as drive caches, sleep mode, power management, acoustic management, and DMA settings. Changing hardware parameters from suboptimal conservative defaults to their optimal settings can improve performance greatly.
- Tune2fs adjusts tunable file system parameters on a Linux second extended file system.

I.7 Heartbeat (HA):

Heartbeat is this software that provides high availability because it can acts automatically on the system when a failure happens.

Its most important features are:

- No fixed maximum number of nodes - Heartbeat can be used to build large clusters as well as very simple ones
- Resource monitoring: resources can be automatically restarted or moved to another node on failure
- Mechanism to remove failed nodes from the cluster
- Sophisticated policy-based resource management, resource inter-dependencies and constraints
- Time-based rules allow for different policies depending on time
- Several resource scripts (for Apache, DB2, Oracle, PostgreSQL etc.) included
- GUI for configuring, controlling and monitoring resources and nodes

I.8 Mount (/etc/fstab):

The fstab file is commonly found on Unix and Unix-like systems and is part of the system configuration. The fstab file contains information of where your partitions and storage devices should be mounted and how. [Illustration I.7: /etc/fstabIllustration I.7]

The following is an example of an fstab file on a Red Hat Linux system:

```
# device name mount point fs-type options dump-freq pass-num
LABEL=/                 /                       ext3    defaults        1 1
none                    /dev/pts                devpts  gid=5,mode=620  0 0
none                    /proc                   proc    defaults        0 0
none                    /dev/shm                tmpfs   defaults        0 0

# my removable media
/dev/cdrom              /mnt/cdrom              udf,iso9660 noauto,owner,kudzu,ro 0 0
/dev/fd0                /mnt/floppy             auto    noauto,owner,kudzu 0 0

# my NTFS Windows XP partition
/dev/hda1               /mnt/WinXP              ntfs    ro,defaults     0 0

/dev/hda6               swap                    swap    defaults        0 0

# my files partition shared by windows and linux
/dev/hda7               /mnt/shared             vfat    umask=000       0 0
```

Illustration I.7: /etc/fstab

I.9 ZABBIX:

ZABBIX is some open source software designed to monitor and track the status of various network services, servers, and other network hardware. [Illustration I.8]
It uses MySQL, PostgreSQL or Oracle to store data. ZABBIX offers several monitoring options. Simple checks can verify the availability and responsiveness of standard services such as SMTP or HTTP without installing any software on the monitored host. A ZABBIX agent can also be installed on UNIX and Windows hosts to monitor statistics such as CPU load, network utilization, disk space, etc. As an alternative to installing an agent on hosts, ZABBIX includes support for monitoring via SNMP.

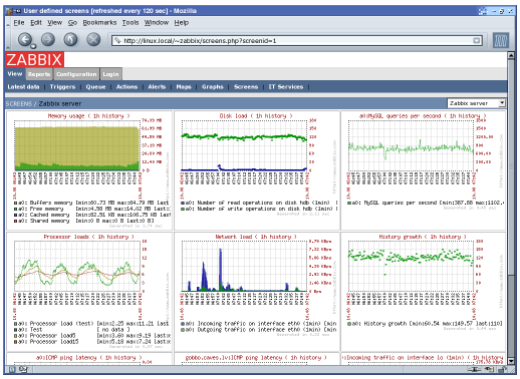


Illustration I.8: Zabbix

I.10 IOzone:

IOzone is a file system benchmark tool. The benchmark generates and measures a variety of file operations. IOzone has been ported to many machines and runs under many operating systems.

IOzone is useful for performing a broad file system analysis of a computer platform. The benchmark tests file I/O performance for the following operations:
Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write

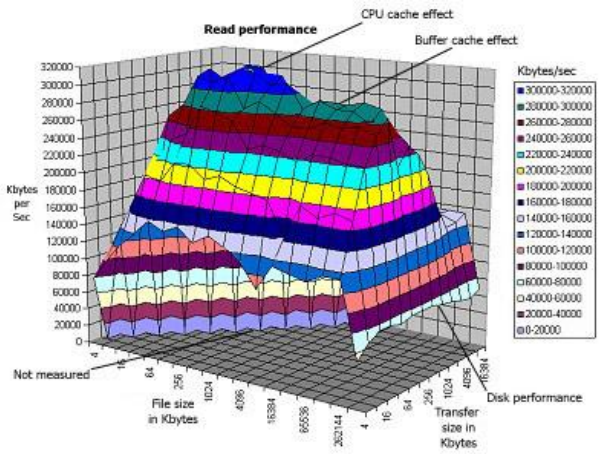


Illustration I.9: IOzone results

I.11 I/O Process:

[Illustration I.10] shows the general scheme of I/O process when we want to write in HD.

Two main policies exist:

- Write Through, when it is written a block in cache memory directly updates the information also in RAM.
- Write Back, when a block is written in cache memory, dirty bit is marked like dirty. When the block is evacuated of cache memory, the dirty bit is verified, and if it's activated, the information of this block writes in RAM.

Finally the RAM writes in HD by DMA, so the application doesn't notice any delay.

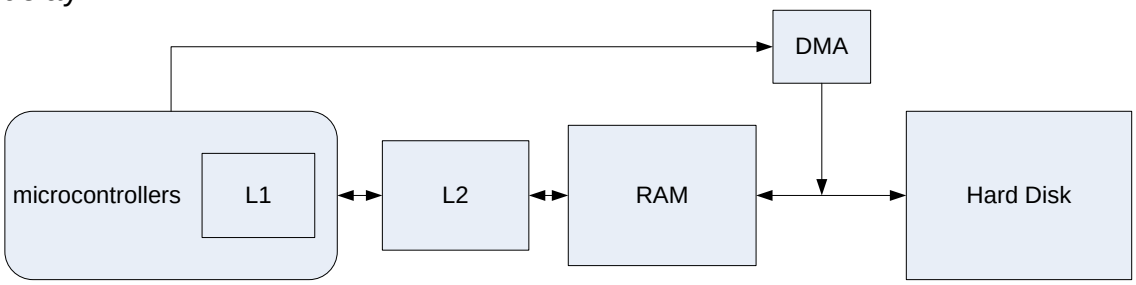


Illustration I.10: General schematic of I/O process

ANNEX II. File System Comparative

In this chapter we want to show some results of file system comparison that have helped us to choose a file system.

We have been testing with EXT3, JFS, XFS and ReiserFS, in addition we found some comparison that helped to decide which the best file system for our scenario is.

Next we indicate the features hardware and software of which we used to do the test: [Table & Table]

II.1 Specifications

Table II.1: Hardware specifications

HARDWARE	
COMPUTER:	Supermicro 5015m-MT
CPU:	Intel Xeon 3Ghz
RAM:	2048MB
SWAP:	800MB
CONTROLLER:	Intel ICH7

Table II.2: Software specification

SOFTWARE	
KERNEL:	linux-2.6.12
COMPILER USED:	gcc-3.3.3
EXT3:	e2fsprogs-1.35
JFS:	jfsutils 1.1.7-1
REISERFS:	reiserfsprogs 3.6.19-1
XFS:	xfsprogs 2.6.20-1

II.2 Used software

In order to do the test we used the Bonnie++, it is software that allows us to measure the writing and reading rate, seek per second and the CPU in each case.

We downloaded from the next website: http://sourceforge.net/projects/bonnie/
In order to install bonnie++ we just untarred the file:

```
$ tar –xvzf bonnie++-1.03a.tgz
```

Now we had to go into the folder and execute the "*configure"* and to do the make.

```
$ cd bonnie++-1.03a
$ ./Configure
$ make
```

Now the software is ready to be used.

```
$ cd bonnie++-1.03a
$ ./bonnie++ -n 0 -u 0 -r 2024 -s 40480 -f -b -d /mnt/qlogic/ >Prueba2_xfs_40480M &
```

With the previous command we could write and read a file of 40480 Mbytes in the folder /mnt/qlogic/ and save the results in another folder, in this case in Prueba2_xfs_40480M.

II.3 Results

Before showing the results we have to say that we had been searching other file system comparisons and in all of them they say that JFS, XFS and ReiserFS are better than EXT3, but they didn't say which among the three of them are the best. Finally we found a comparison in Debian website where they recommend XFS.
Next [Illustration II.1] shows a little comparison about the writing and reading rate, and seek per second in random mode.

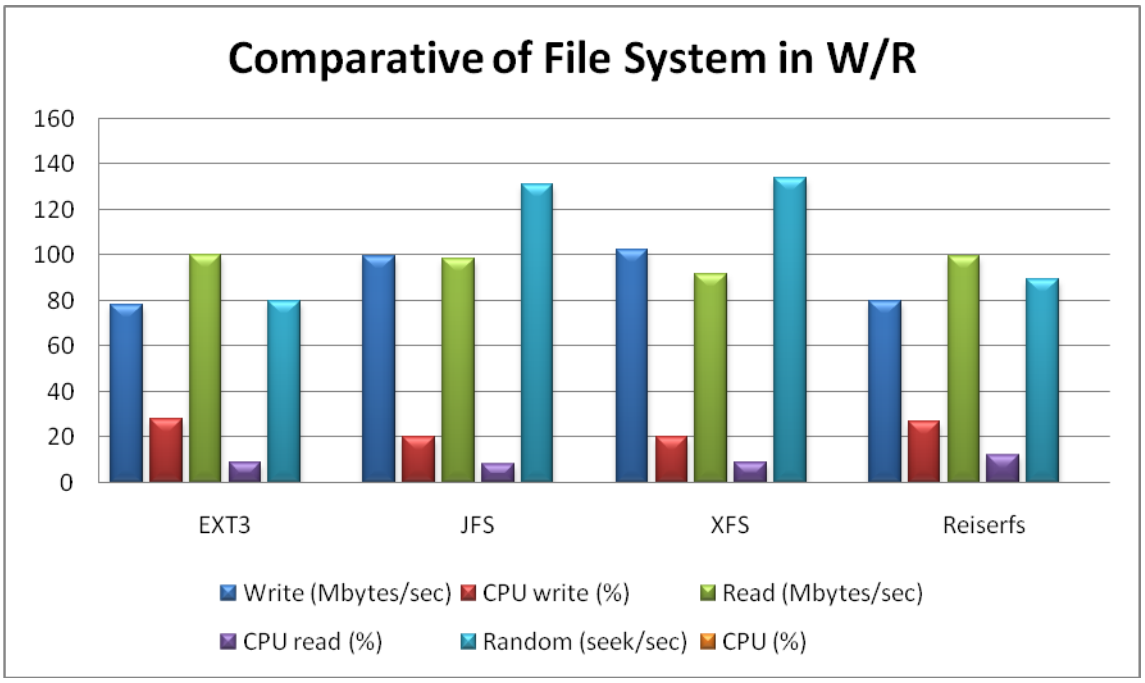


Illustration II.1: Comparative of file system

We can see how JFS and XFS are better than EXT3 and ReiserFS, the rate in writing and reading and seek per second is bigger than the rest of file system.

Now we have to decide between JFS and XFS, both are very similar, they are good writing and reading rate, but if we trust in others comparisons we would select the XFS[3].

[3] Althought we chose XFS, we didn't get Lustre works correctly with XFS, the we finally used EXT3

ANNEX III. Installation step by step

Lustre will require some changes to the core Linux kernel, to do it so there exists a Patch management called Quilt, there were several patches needed to install Lustre. We installed the last version (Quilt-0.46).

First of all we downloaded Lustre and quilt and we unpacked it under /home/lustre/.

We can obtain the Quilt in the following website:

http://savannah.nongnu.org/projects/quilt

And Lustre in:

http://www.clusterfs.com/download.html

We have the kernel source tree in /usr/src/linux-2.6.12

Now we show how to install the quilt:

```
$ cd /usr/src/linux-2.6.12
$ ln -s /home/lustre/lustre-1.4.9/lustre/kernel_patches/series/2.6-\rhel4.series
./series
$ ln -s / home/lustre /lustre-1.4.8.x/lustre/kernel_patches/patches .
$ cd /home/lustre/quilt-0.46/
$ ./configure
$ make install
$ quilt push –av
```

Once quilt is installed, we are going to explain how to install Lustre:

```
$ cd /usr/src/linux-2.6.12
$ cp /boot/config-'uname -r' .config
$ make oldconfig || make menuconfig
$ make include/asm
$ make include/linux/version.h
$ make SUBDIRS=scripts
$ cd /home/inwe/lustre/lustre-1.4.9
$ ./configure --with-linux=/usr/src/linux-2.6.12
$ make install
```

Now, Lustre is installed and ready to be used.

ANNEX IV. LUSTRE COMMANDS

IV.1 lconf

Lustre file system configuration utility – This utility configures a node following directives in the `<XML-config file>`. There is a single configuration file for all the nodes in a single cluster. This file should be distributed to all the nodes in the cluster or kept in a location accessible to all the nodes. One option is to store the cluster configuration information in lightweight directory access protocol (LDAP) format on an LDAP server that can be reached from all of the cluster nodes.

IV.2 lctl

Low level Lustre file system configuration utility – This utility provides very low level access to the file system internals.

IV.3 lfs

Lustre utility to create a file with a specific striping pattern and find the striping pattern of existing files – This utility can be used to create a new file with a specific striping pattern, determine the default striping pattern, and gather the extended attributes (object numbers and location) for a specific file. It can be invoked interactively without any arguments or in a noninteractive mode with one of the arguments supported.

IV.4 lmc

Lustre configuration maker – This utility adds configuration data to a configuration file. In the future, `lmc` will also be able to remove configuration data or convert its format. A Lustre cluster consists of several components: metadata servers (MDSs), client mount points, object storage targets (OSTs), logical object volumes (LOVs), and networks. A single configuration file is generated for the complete cluster. In the `lmc` command line interface, each of these components is associated with an object type.

IV.5 lwizard

Lustre configuration wizard – The configuration files for Lustre installation are generally created through a series of `lmc` commands; this generates an XML file that describes the complete cluster. The lwizard eliminates the need to learn

`lmc` to generate configuration files, instead, lwizard achieves the same result through asking some simple questions. The XML configuration file generated using lwizard still has to be made accessible to all the cluster nodes either by storing it on an LDAP server, network file system (NFS), or by copying it to all the involved nodes. Then `lconf` is run on all nodes to start the various Lustre services and device setups or to mount the file system. Using lwizard allows the user to simply answer a series of questions about the various pieces of the cluster, and the lwizard completes the configuration.

ANNEX V. SCRIPTS OF LUSTRE FILE SYSTEM

V.1 Script 1OSS+1OST+1client

```
#!/bin/bash

config="${config:-Prueba_1cliente_1OST.xml}"

rm Prueba_1cliente_1OST.xml

# Create nodes
lmc -o $config --add node --node deb0
#lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
#lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
#lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

V.2 Script 1OSS+2OST+1client

```
#!/bin/bash

config="${config:-Prueba_1cliente_2OST.xml}"

rm Prueba_1cliente_2OST.xml

# Create nodes
lmc -o $config --add node --node deb0
#lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
#lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
```

```
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb3 --ost deb0-ost1

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

V.3 **Script 1OSS+3OST+1client**

```
#!/bin/bash

config="${config:-Prueba_1cliente_3OST.xml}"

rm Prueba_1cliente_3OST.xml

# Create nodes
lmc -o $config --add node --node deb0
#lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
#lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb3 --ost deb0-ost1
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb4 --ost deb0-ost2


# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

V.4 **Script 2OSS+2OST+1client**

```
#!/bin/bash
```

```
config="${config:-Prueba_1cliente_2OSS_2OST.xml}"

rm Prueba_1cliente_2OSS_2OST.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

V.5 **Script 2OSS+3OST+1client**

```
#!/bin/bash

config="${config:-Prueba_1cliente_2OSS_3OST.xml}"

rm Prueba_1cliente_2OSS_3OST.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0
```

```
# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb3 --ost deb0-ost1
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

V.6 **Script 2OSS+4OST+1client**

```
#!/bin/bash

config="${config:-Prueba_1cliente_2OSS_4OST.xml}"

rm Prueba_1cliente_2OSS_4OST.xml

# Create nodes
lmc -o $config --add node --node deb0
lmc -m $config --add node --node deb1
lmc -m $config --add node --node client

# Add net
lmc -m $config --add net --node deb0 --nid deb0 --nettype tcp
lmc -m $config --add net --node deb1 --nid deb1 --nettype tcp

# Generic client definition
lmc -m $config --add net --node client --nid '*' --nettype tcp

# Configure mds server
lmc -m $config --add mds --node deb0 --mds deb0-mds0 --fstype ext3 --dev /dev/sdb1

# Create LOVs
lmc -m $config --add lov --lov lov1 --mds deb0-mds0 --stripe_sz 1048576 --stripe_cnt -1 --stripe_pattern
0

# Configure OSTs
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb0-ost0
lmc -m $config --add ost --node deb0 --lov lov1 --fstype ldiskfs --dev /dev/sdb3 --ost deb0-ost1
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb1 --ost deb1-ost1
lmc -m $config --add ost --node deb1 --lov lov1 --fstype ldiskfs --dev /dev/sdb2 --ost deb1-ost2

# Create client config
lmc -m $config --add mtpt --node client --path /mnt/lustre --mds deb0-mds0 --lov lov1
```

ANNEX VI. GRAPHS OF TEST

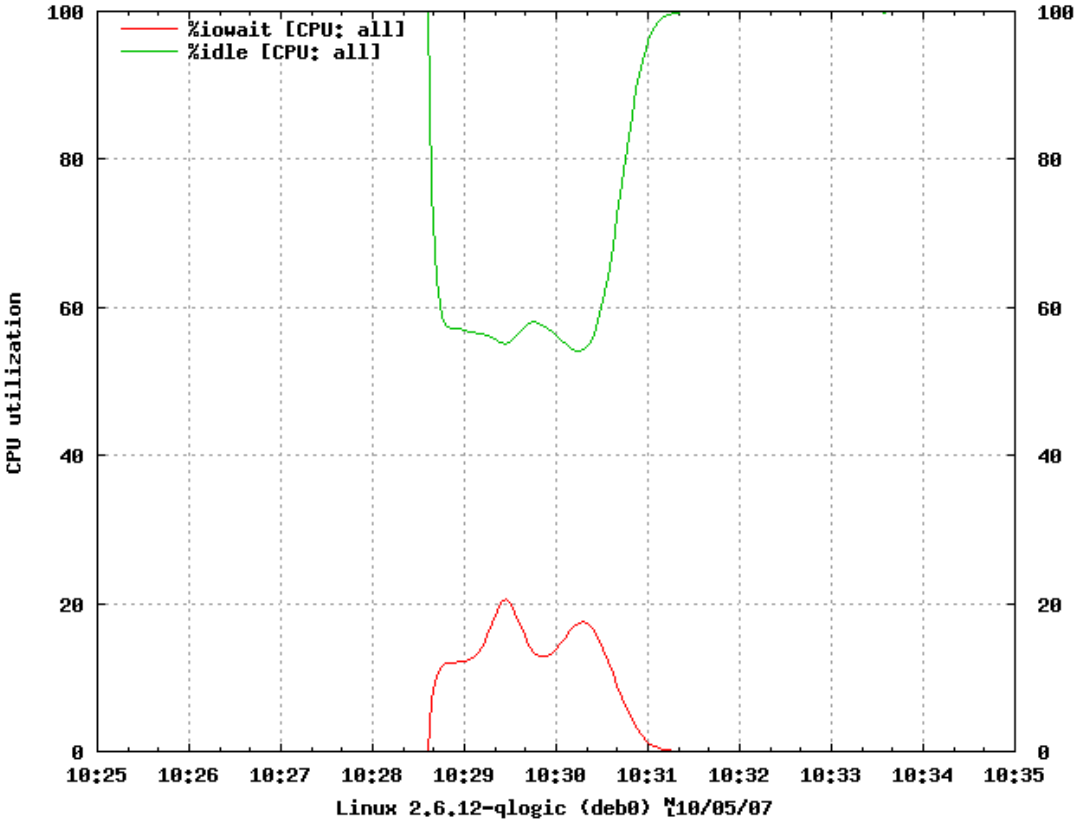


Illustration V.1: 1OSS-1OST-deb0

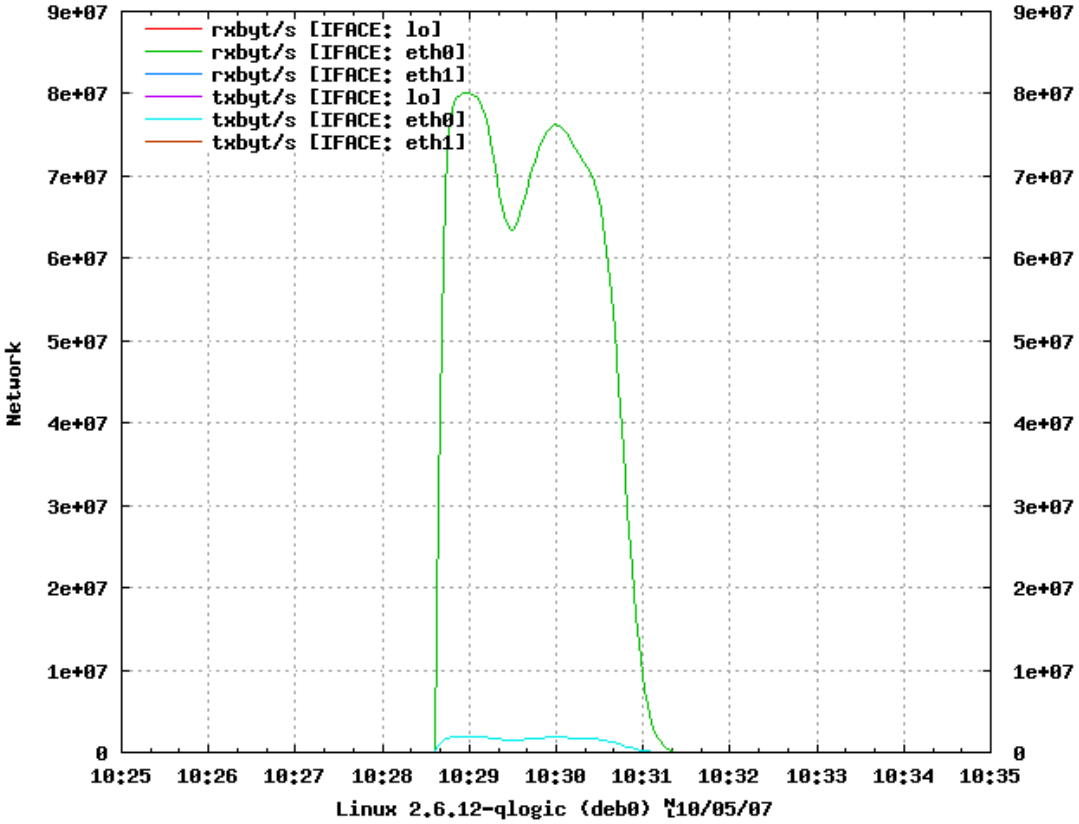


Illustration V.2: 1OSS-1OST-deb0

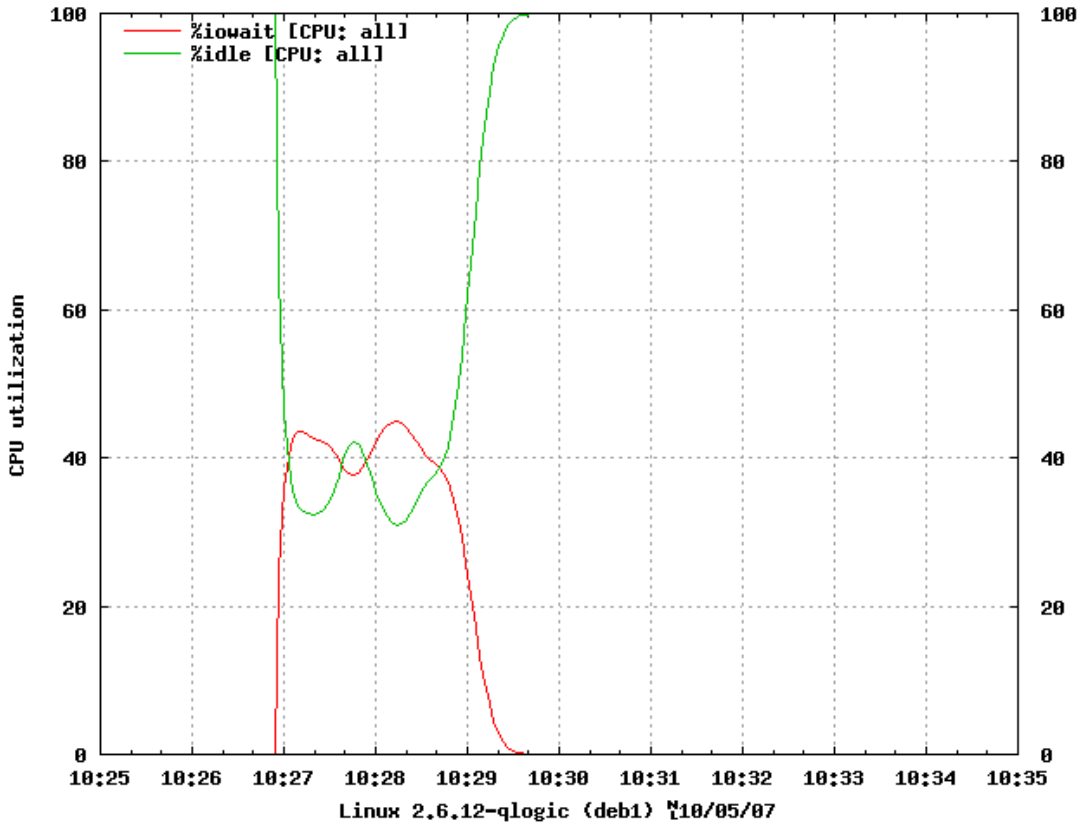


Illustration V.3: 1OSS-1OST-deb1

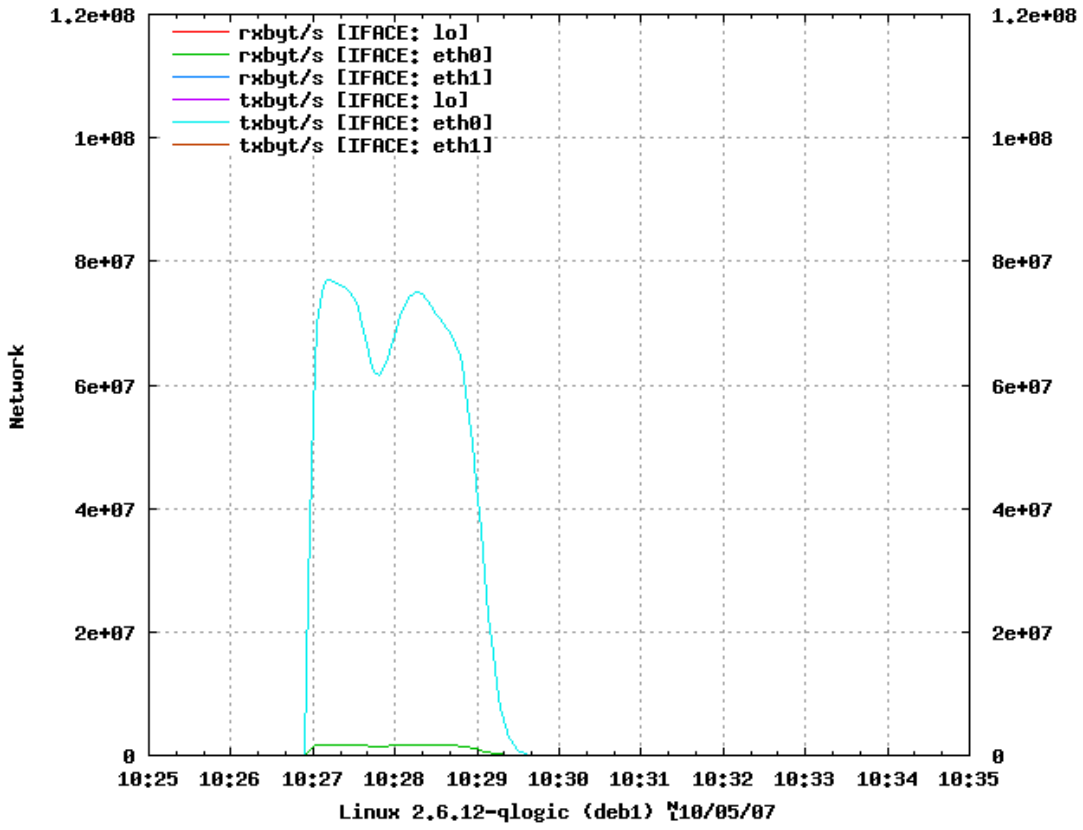


Illustration V.4: 1OSS-1OST-deb1

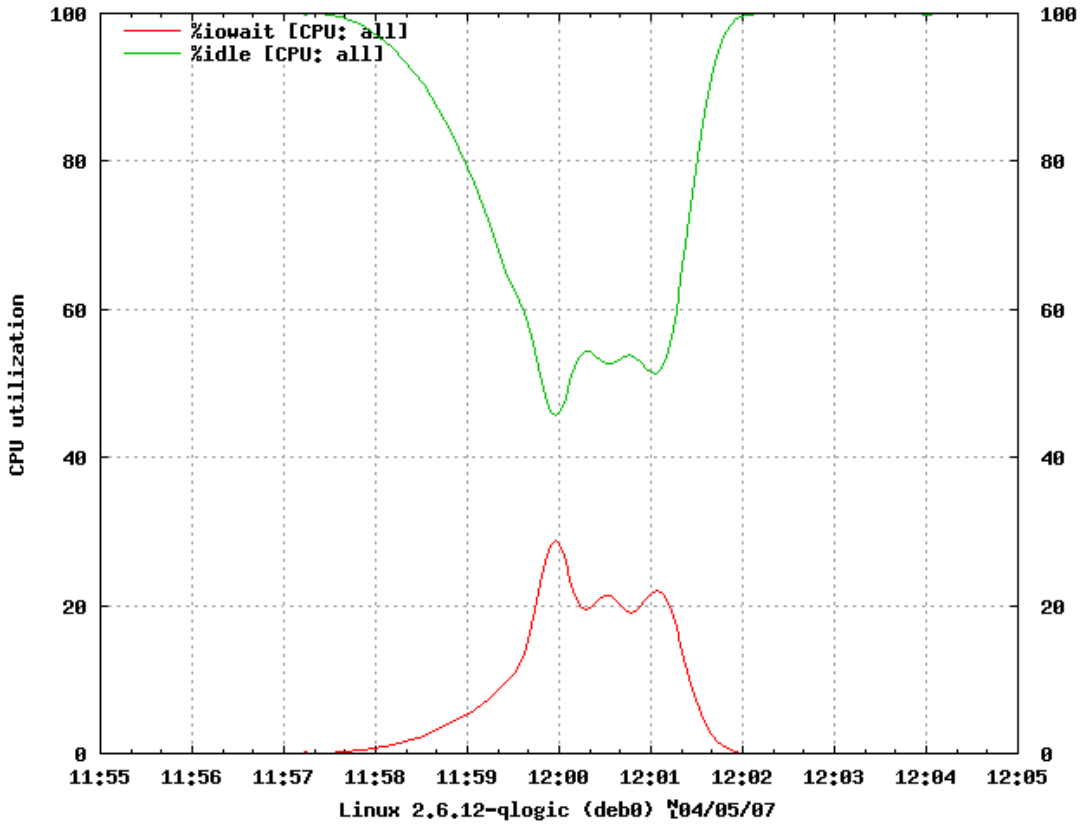


Illustration V.5: 1OSS-2OST-deb0

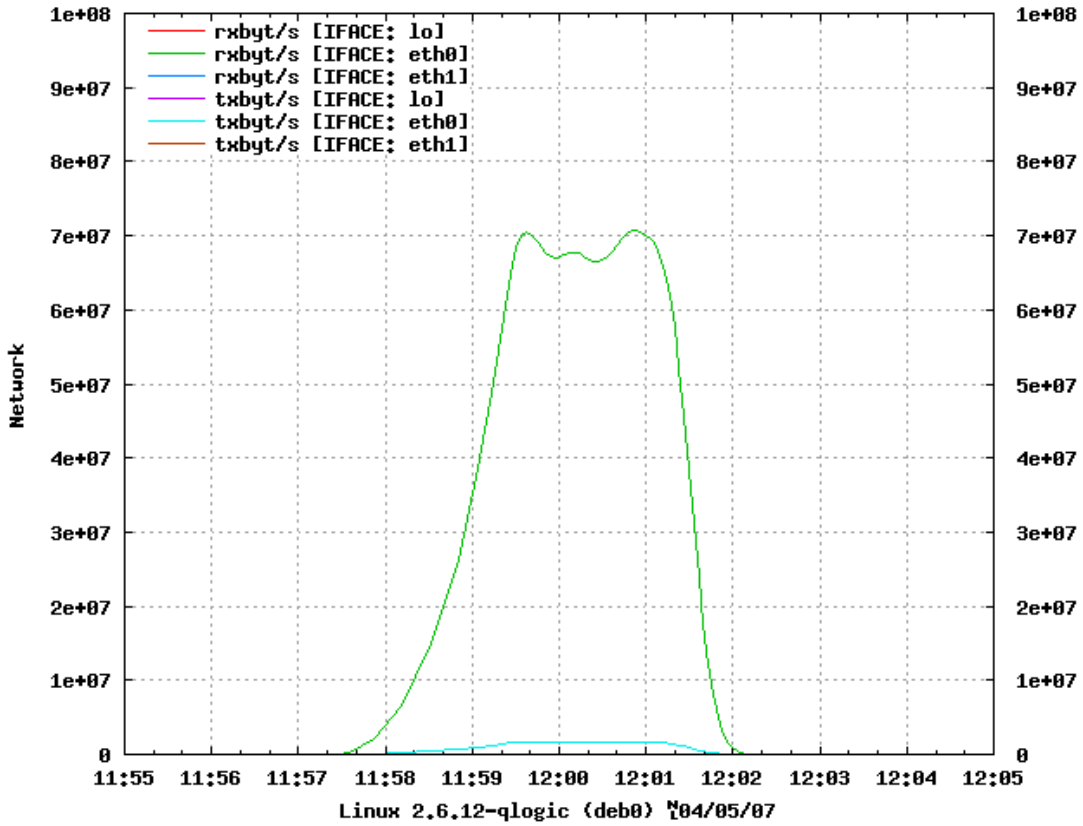


Illustration V.6: 1OSS-2OST-deb0

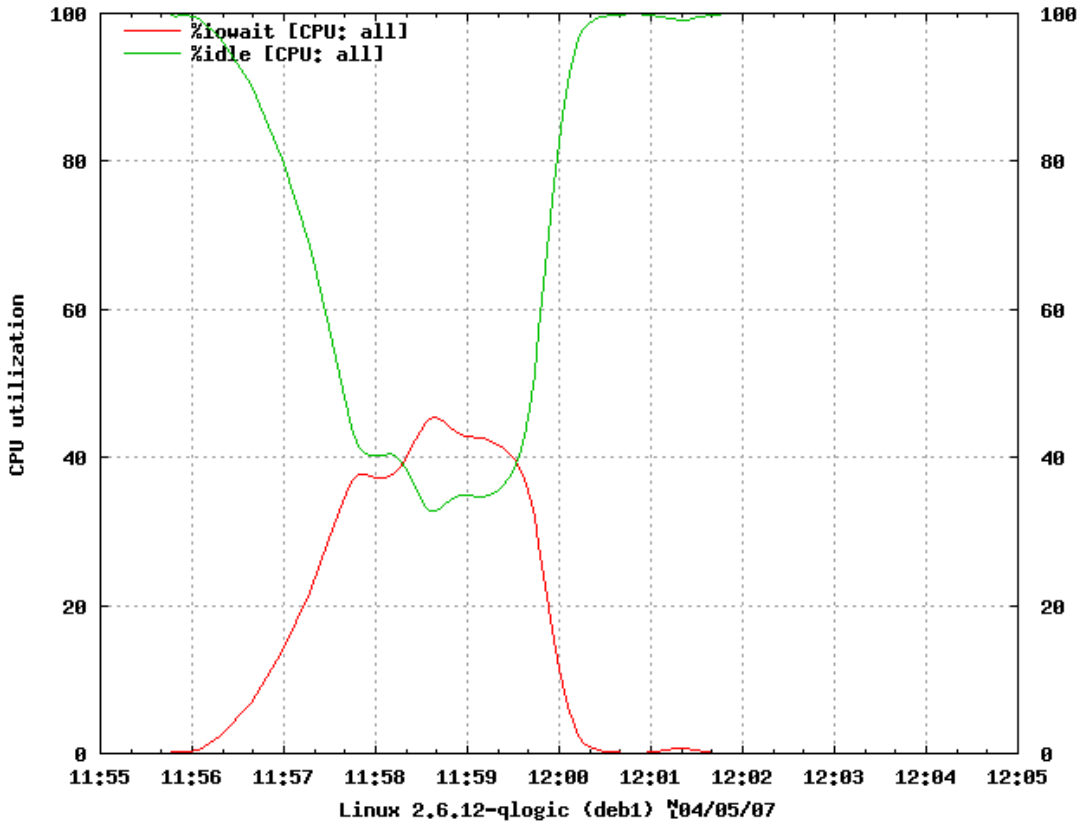


Illustration V.7: 1OSS-2OST-deb1

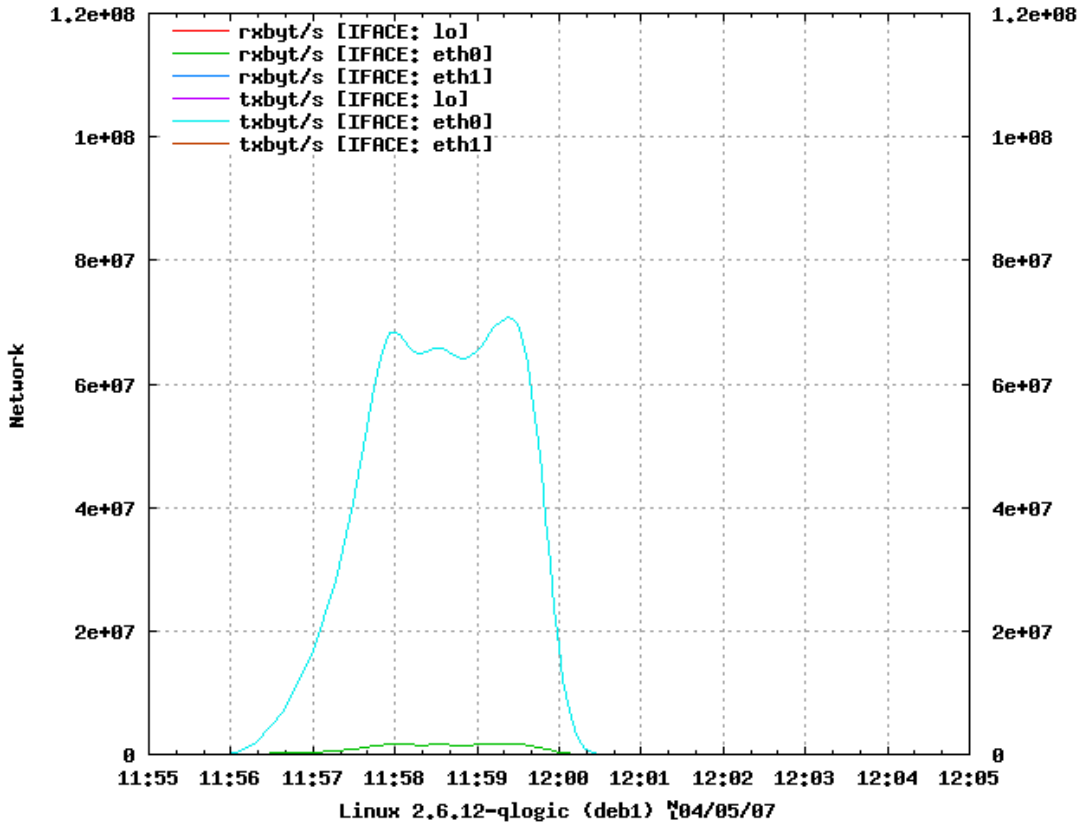


Illustration V.8: 1OSS-2OST-deb1

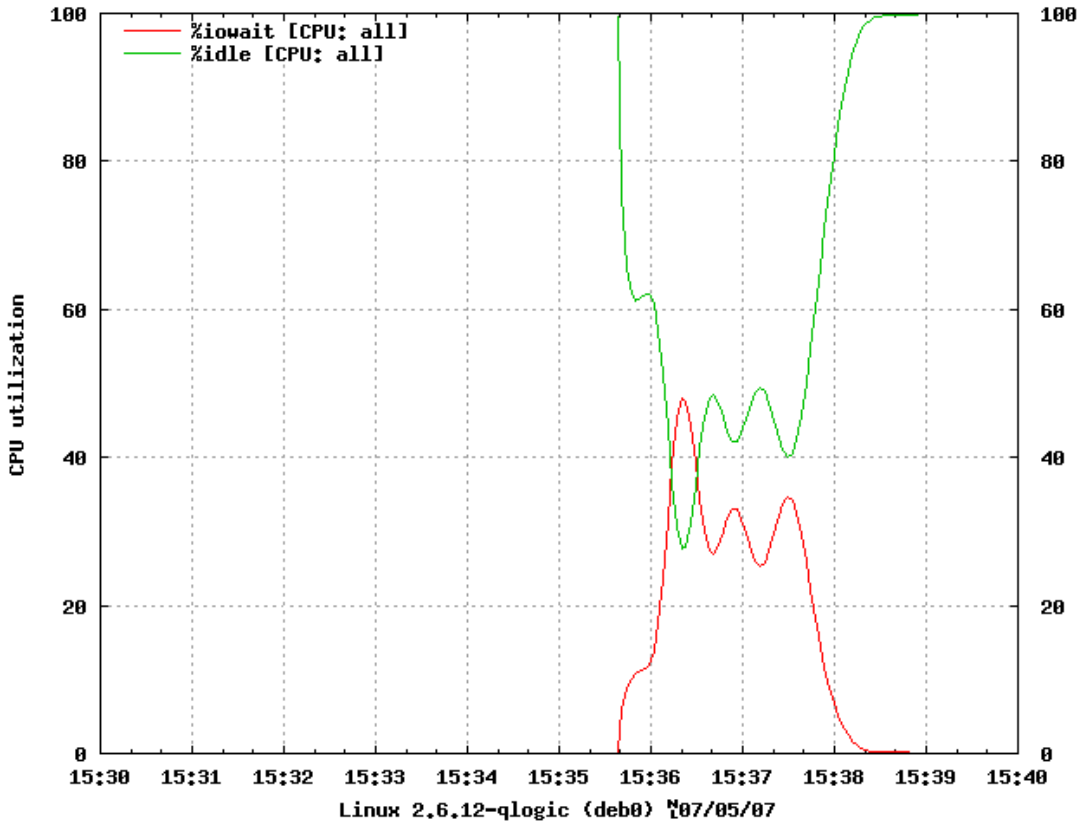


Illustration V.9: 1OSS-3OST-deb0

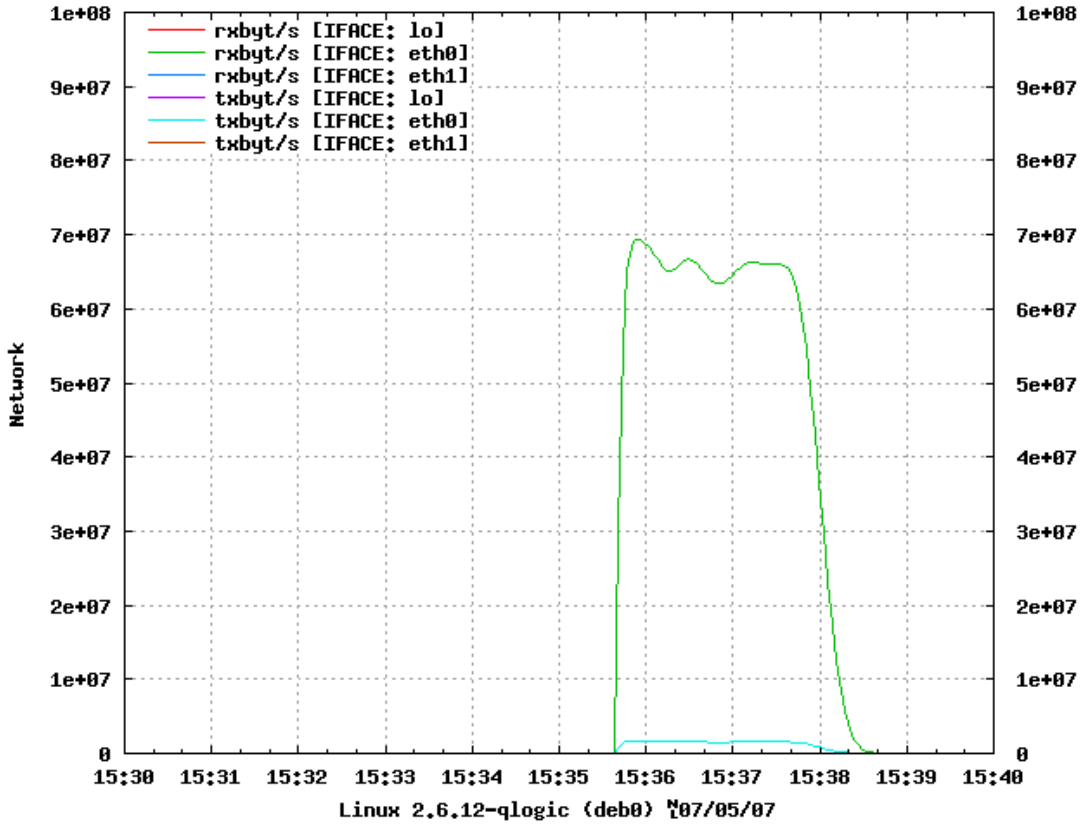


Illustration V.10: 1OSS-3OST-deb0

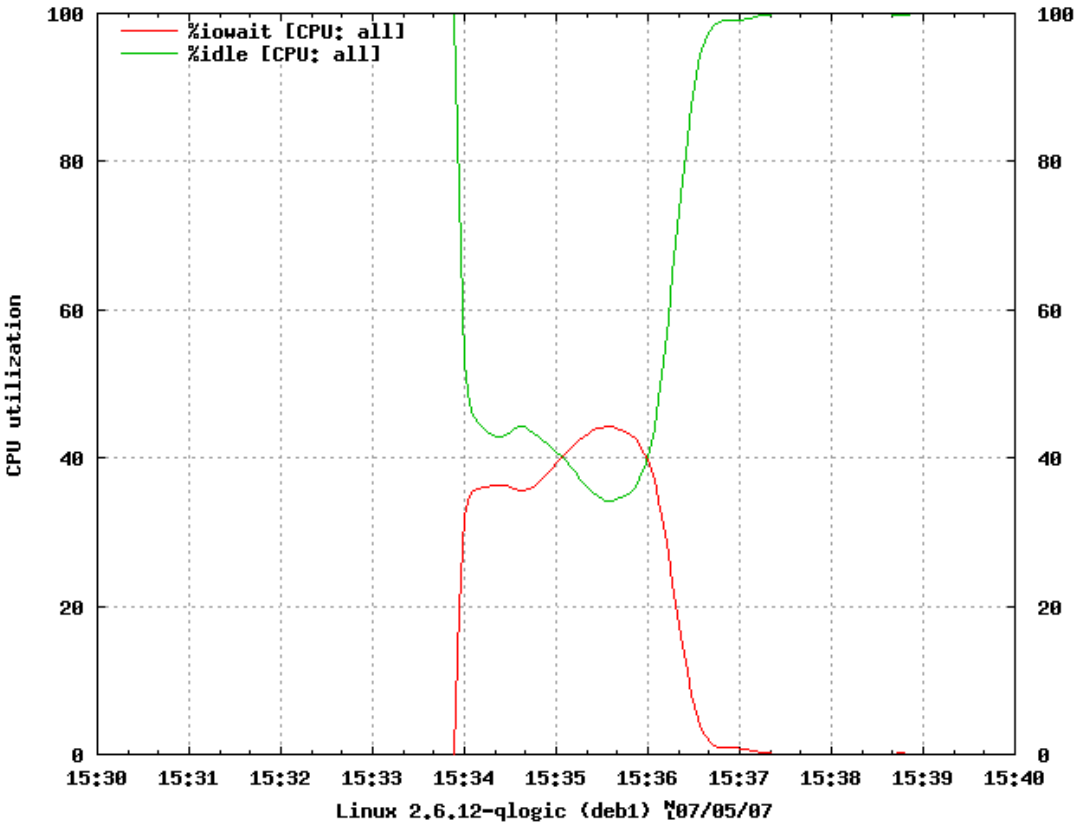


Illustration V.11: 1OSS-3OST-deb1

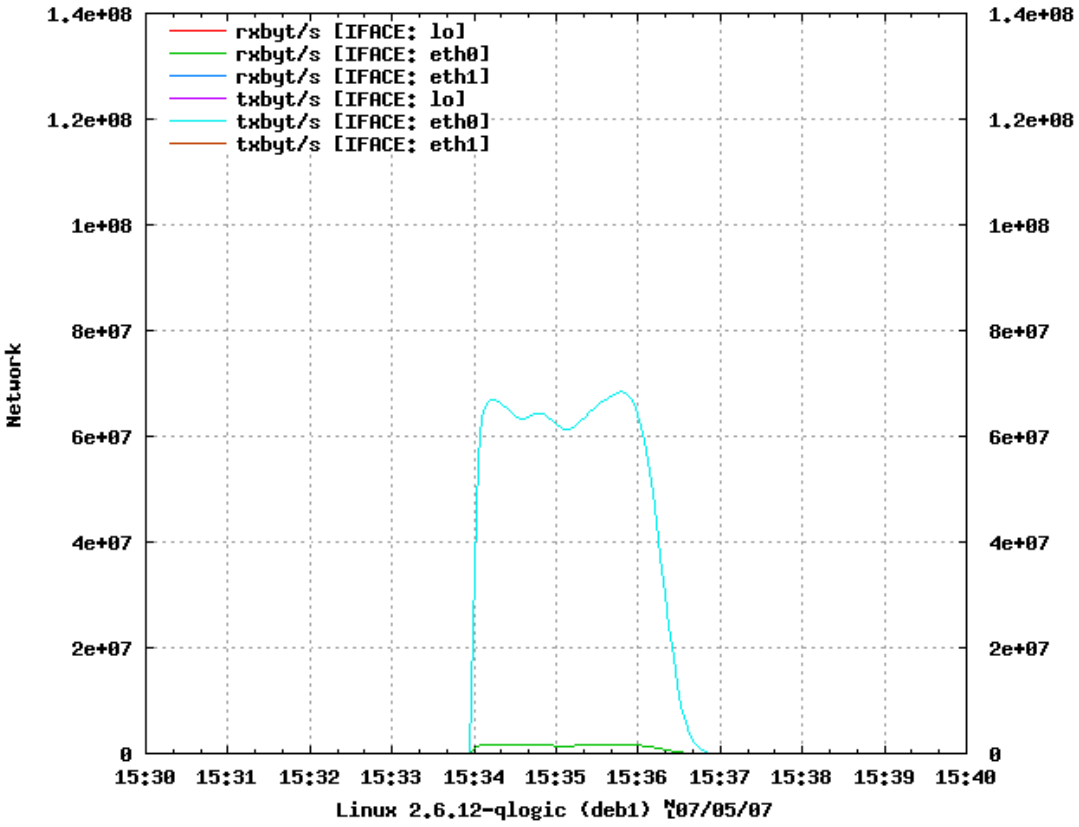


Illustration V.12: 1OSS-3OST-deb1

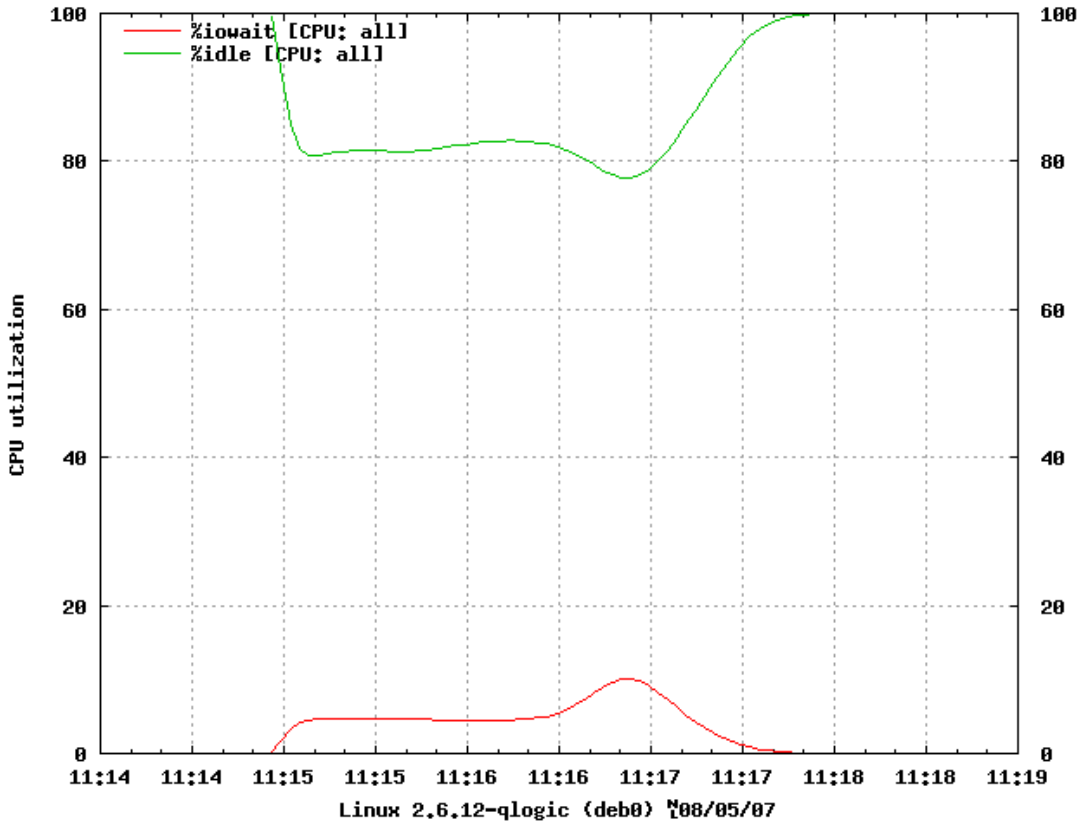


Illustration V.13: 2OSS-2OST-deb0

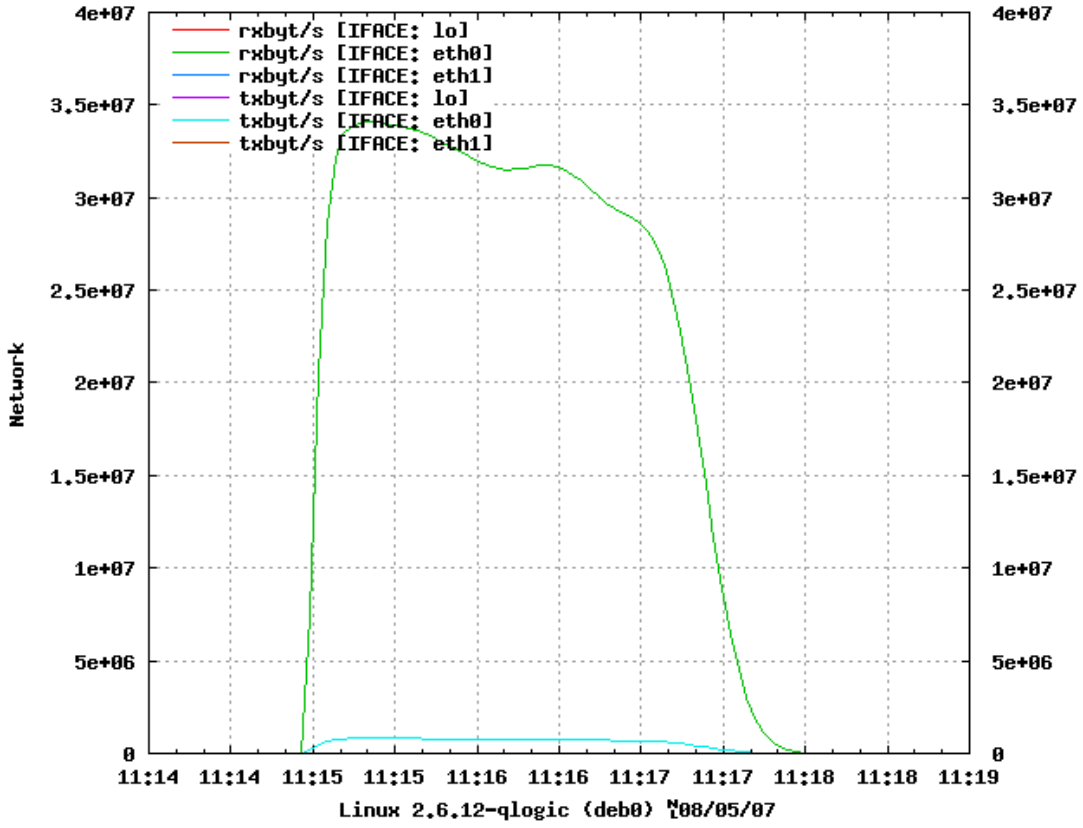


Illustration V.14: 2OSS-2OST-deb0

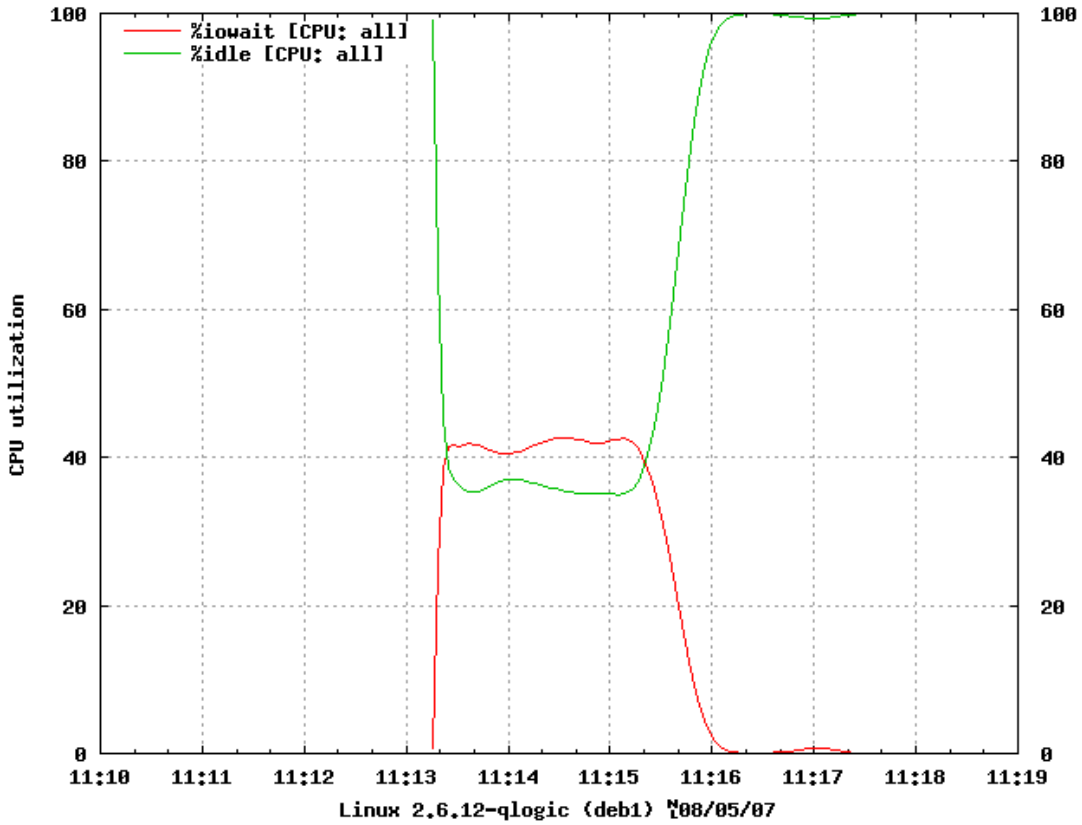


Illustration V.15: 2OSS-2OST-deb1

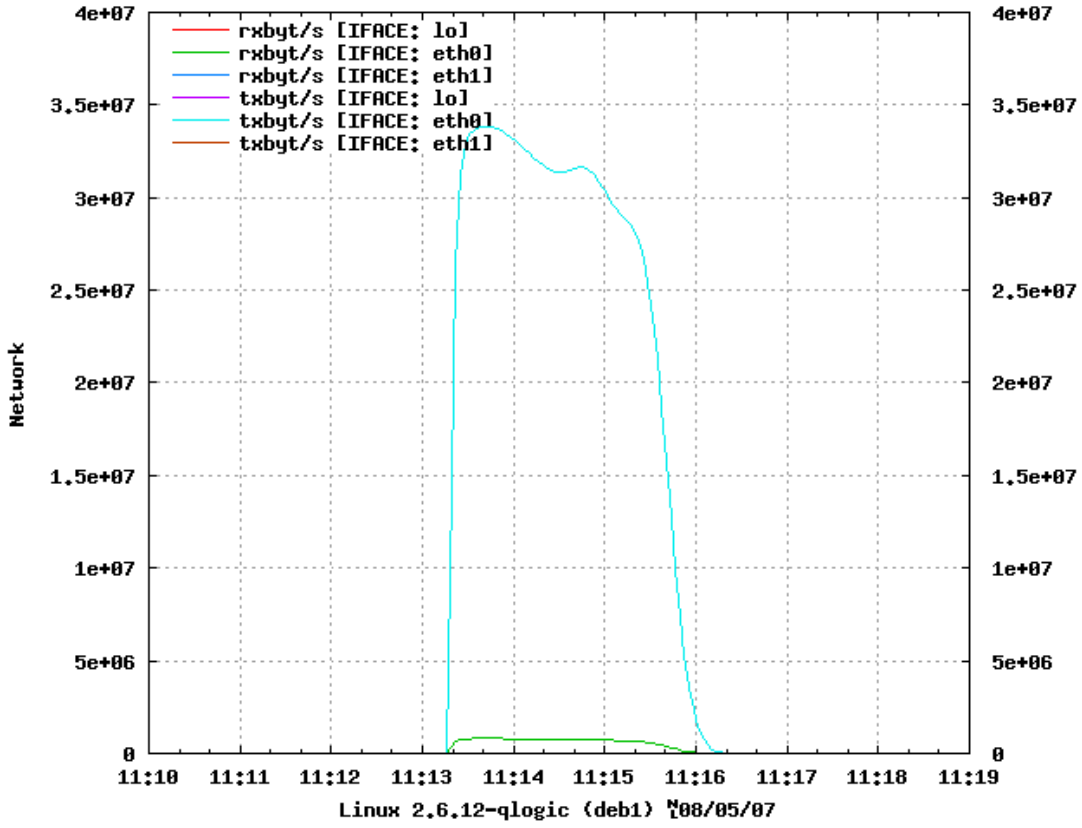


Illustration V.16: 2OSS-2OST-deb1

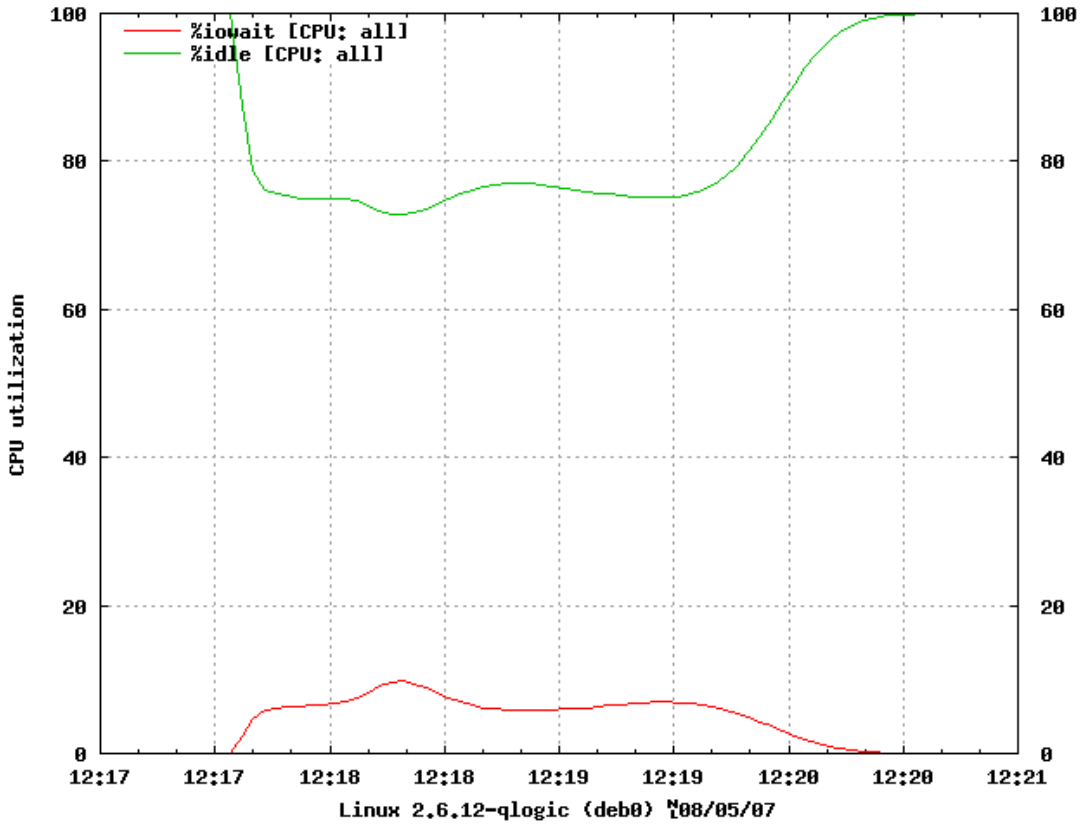


Illustration V.17: 2OSS-3OST-deb0

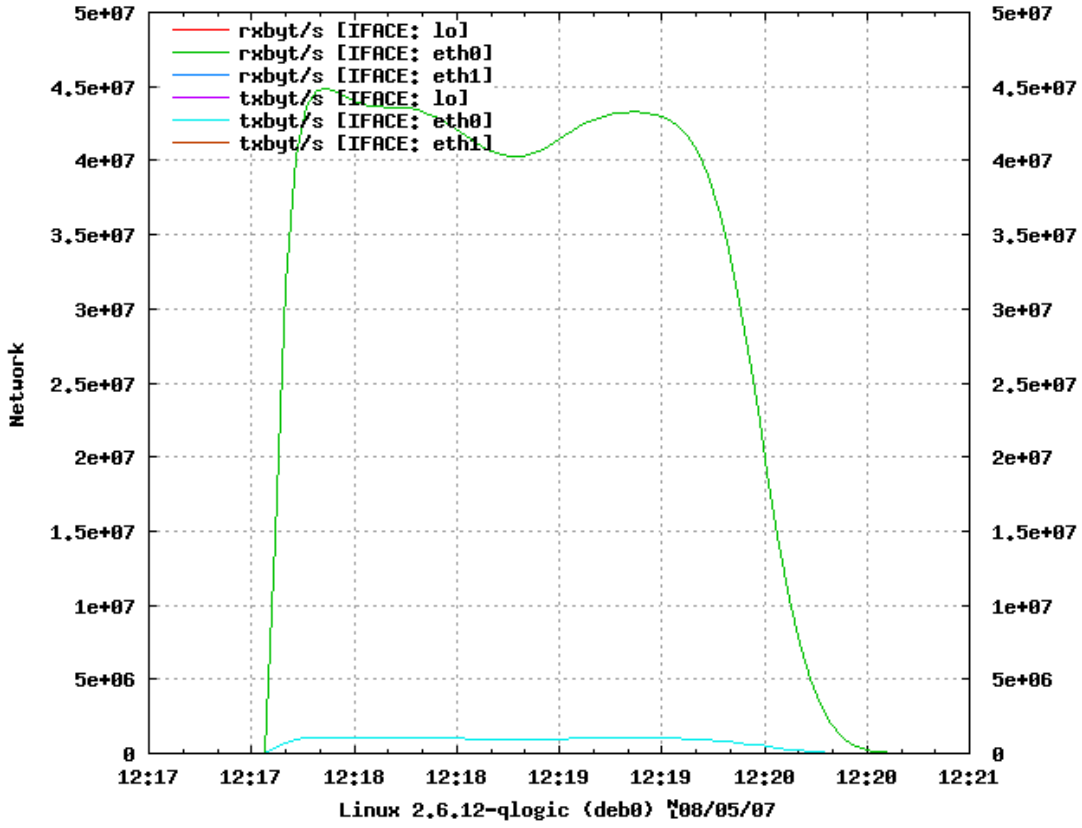


Illustration V.18: 2OSS-3OST-deb0

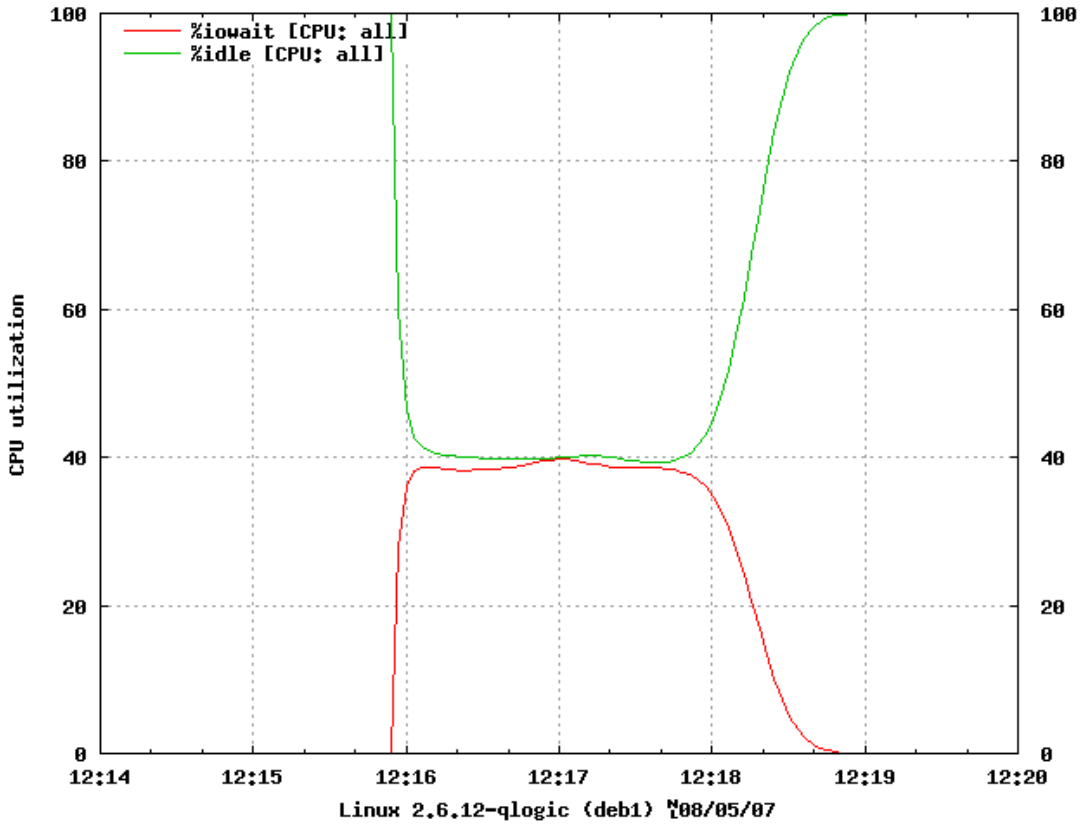


Illustration V.19: 2OSS-3OST-deb1

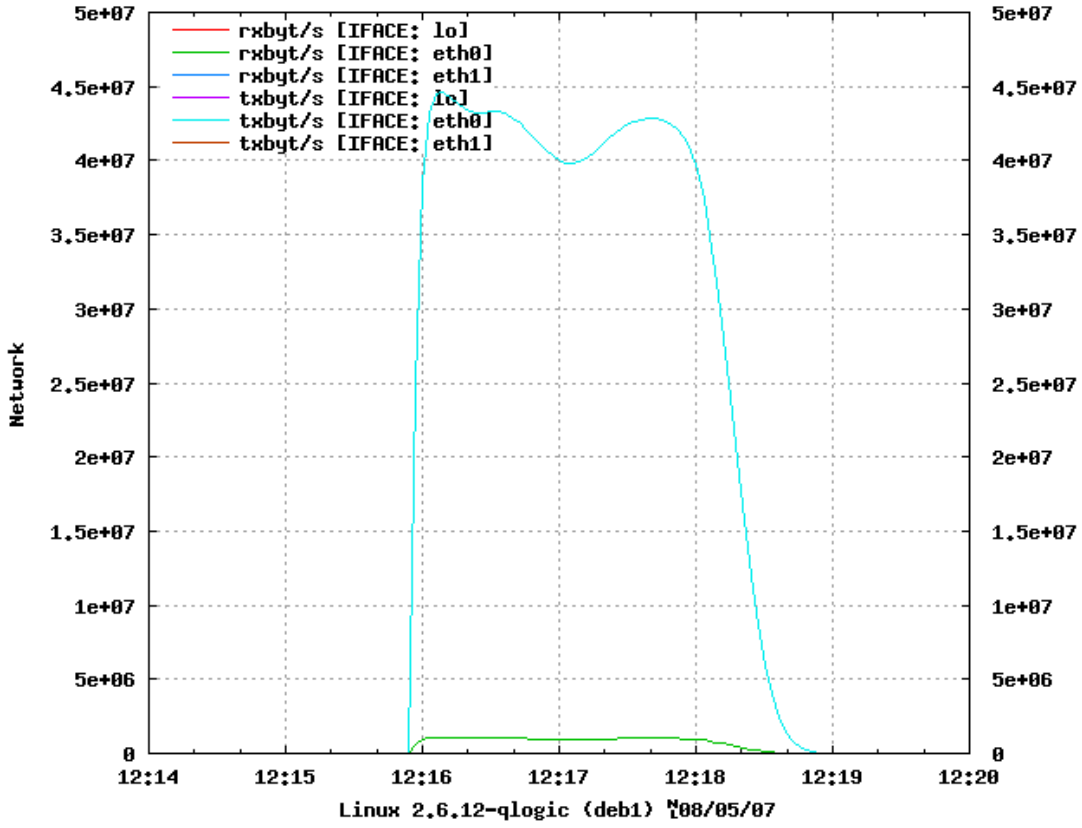


Illustration V.20: 2OSS-3OST-deb1

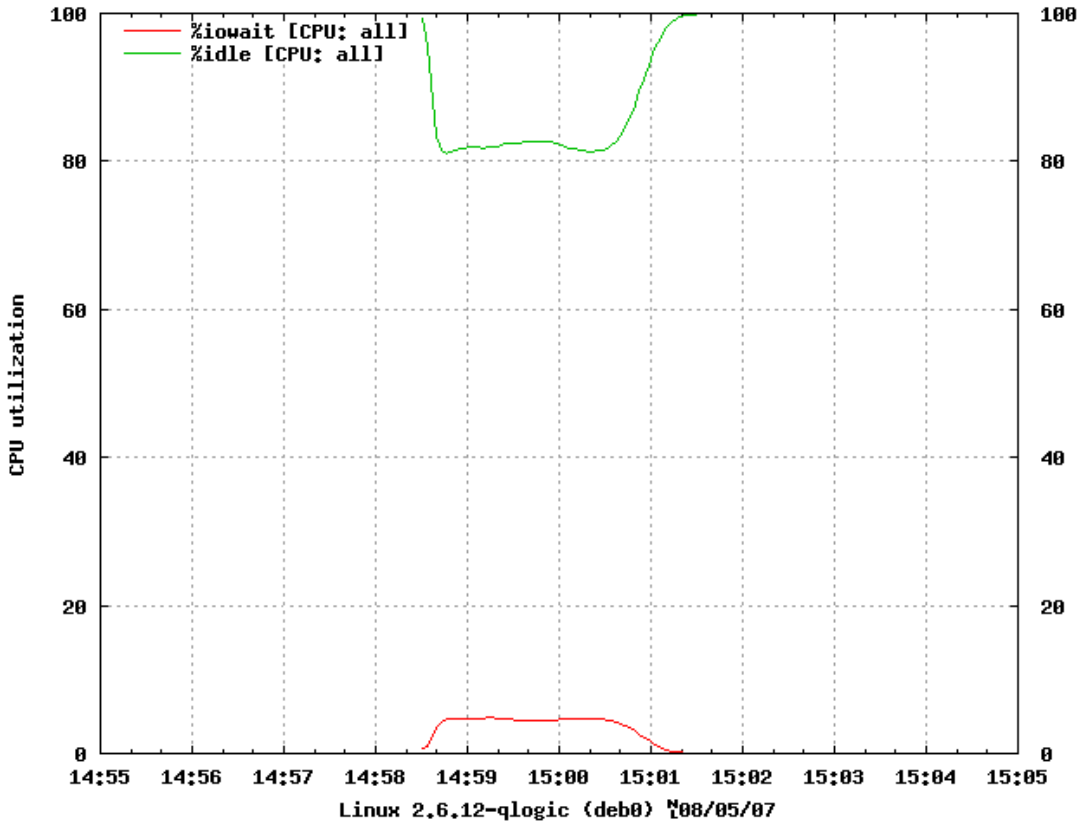


Illustration V.21: 2OSS-4OST-deb0

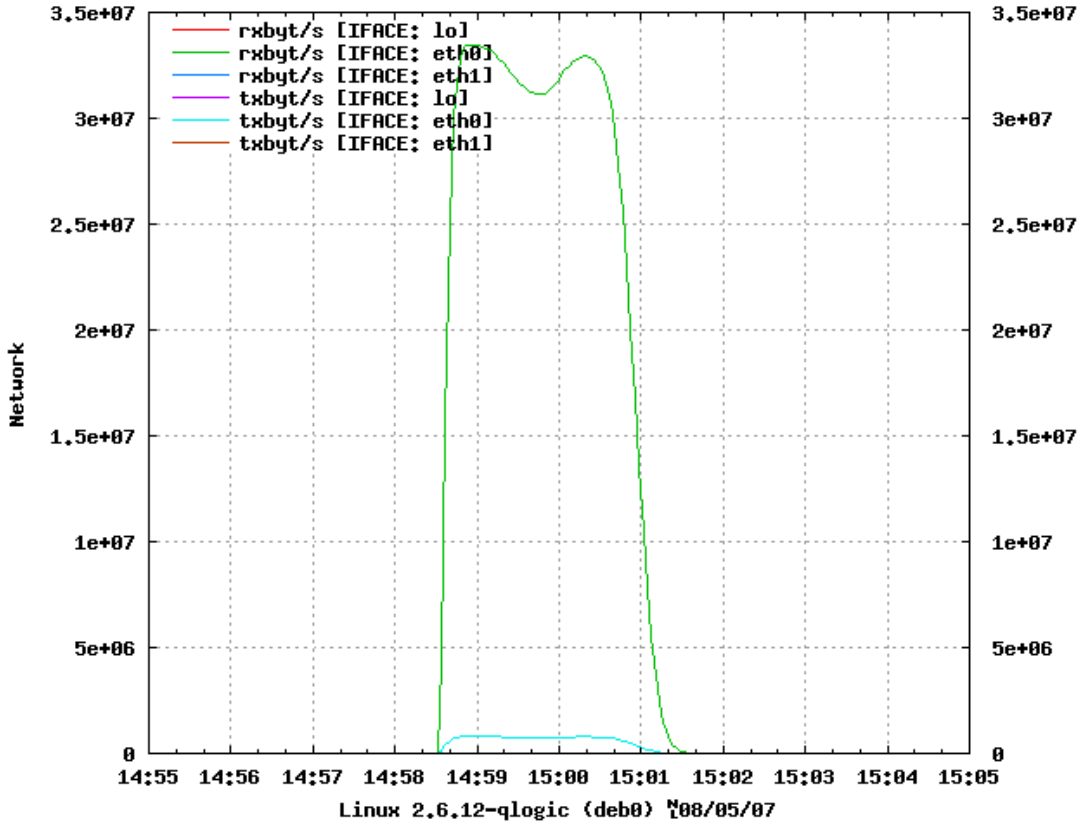


Illustration V.22: 2OSS-4OST-deb0

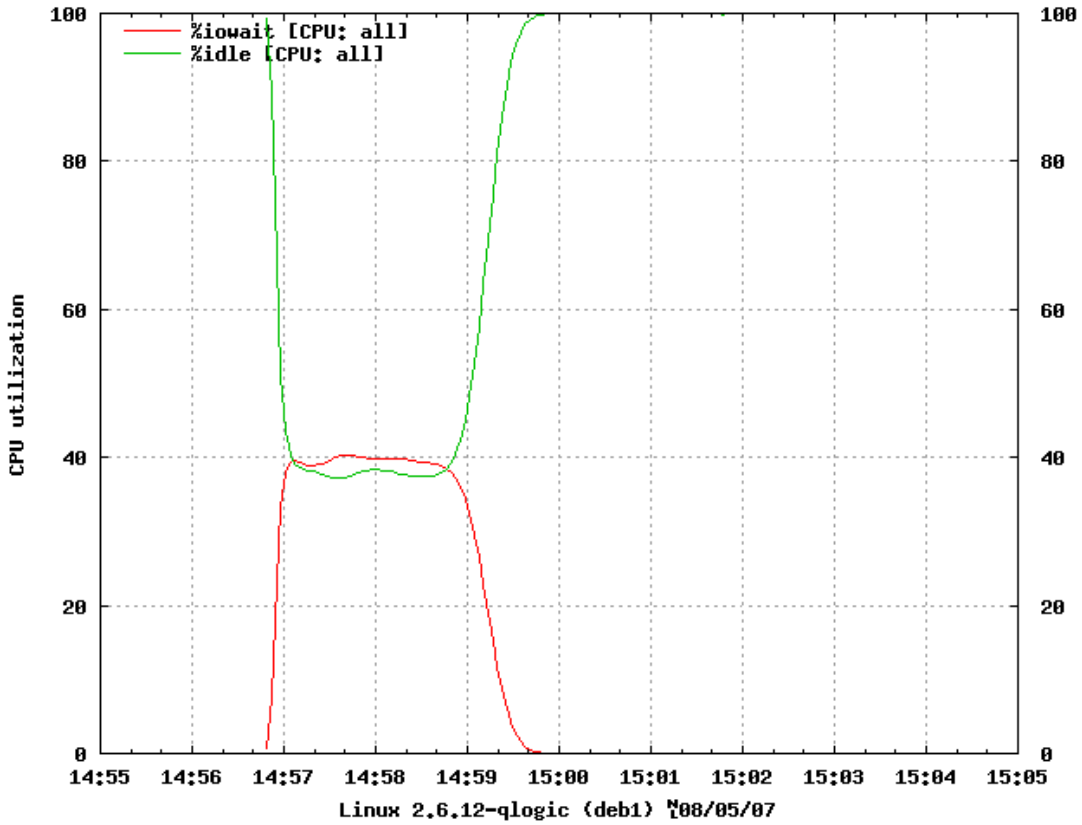


Illustration V.23: 2OSS-4OST-deb1

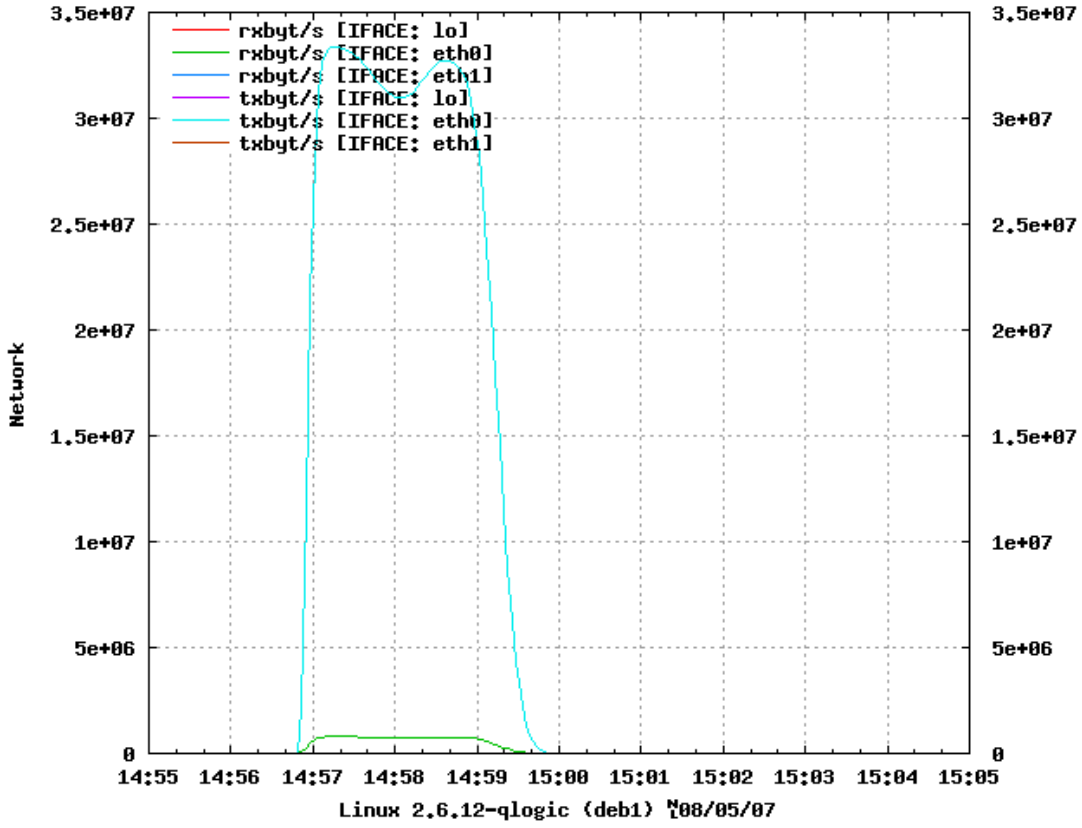


Illustration V.24: 2OSS-4OST-deb1

Scenario with 1OSS and 1OST (Test of StripeSize)

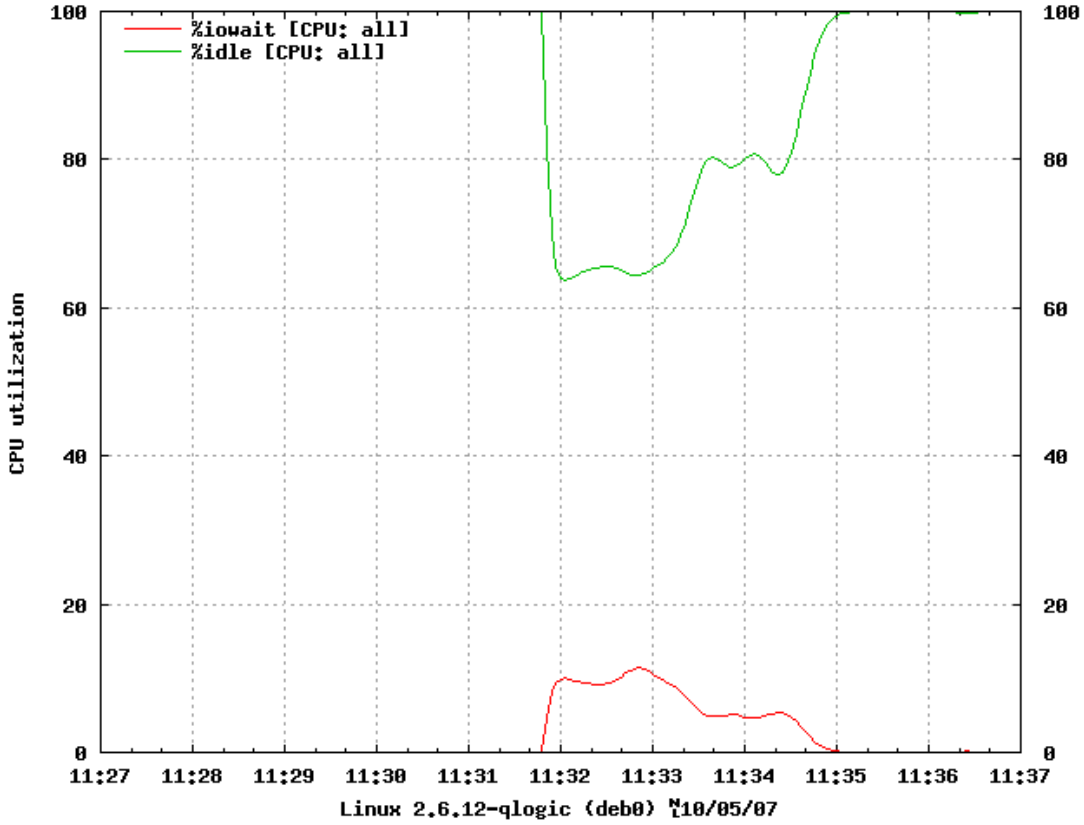


Ilustración V.25: Stripetest 50MB deb0

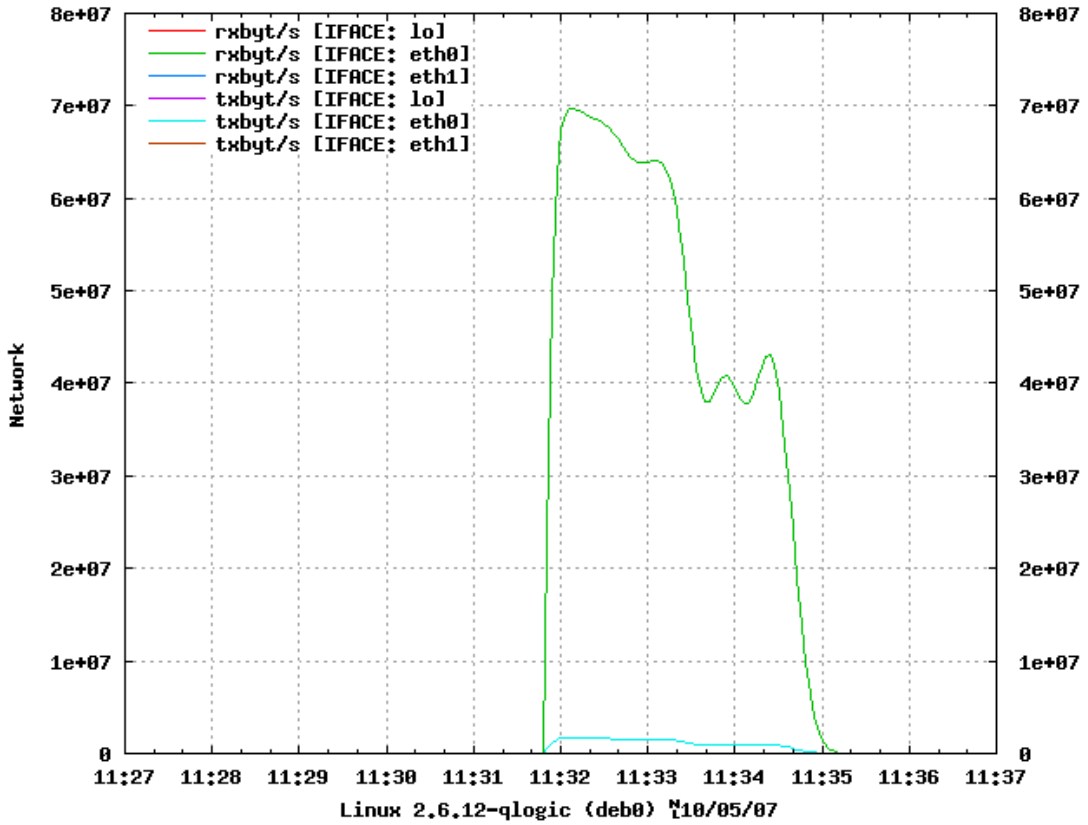


Ilustración V.26: Stripetest 50MB deb0

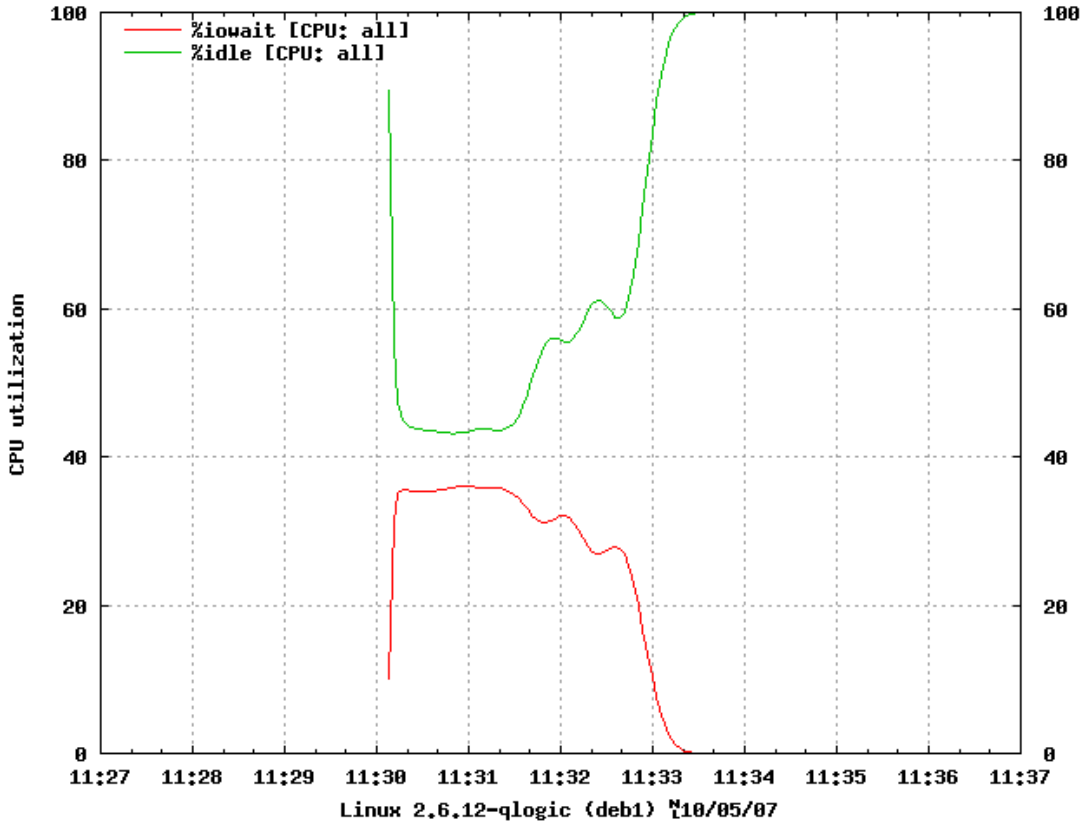


Ilustración V.27: Stripetest 50MB deb1

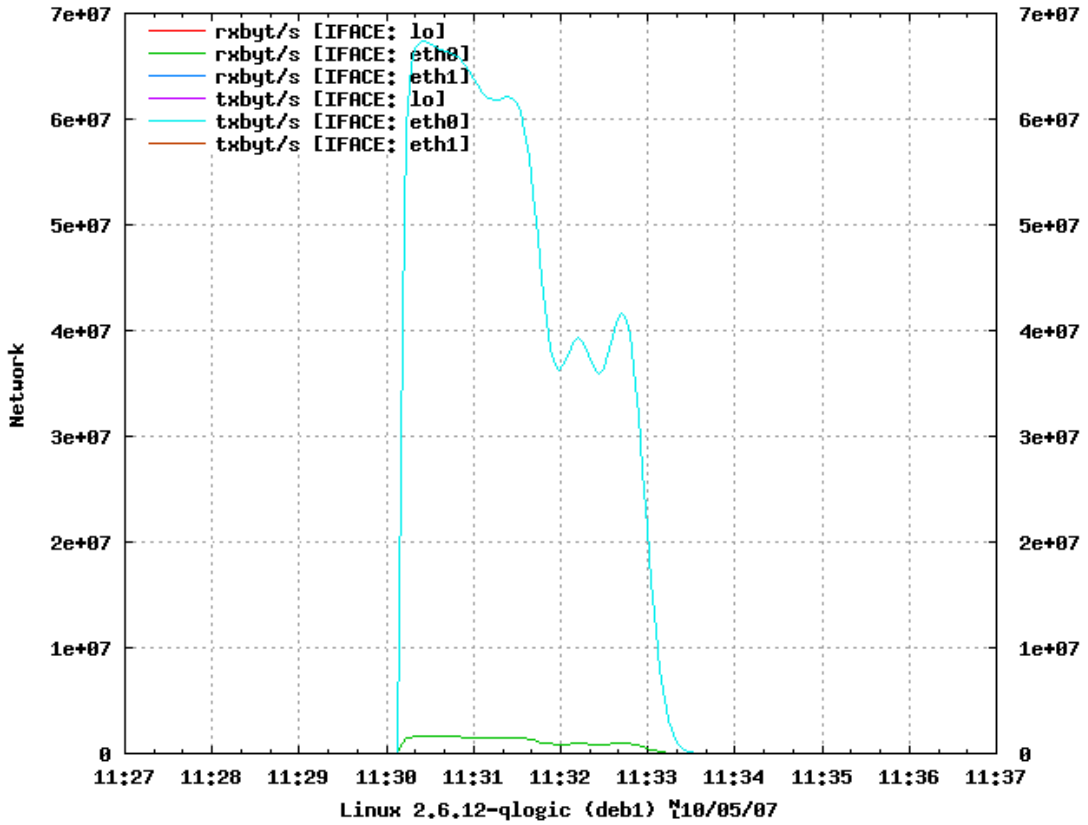


Ilustración V.28: Stripetest 50MB deb1

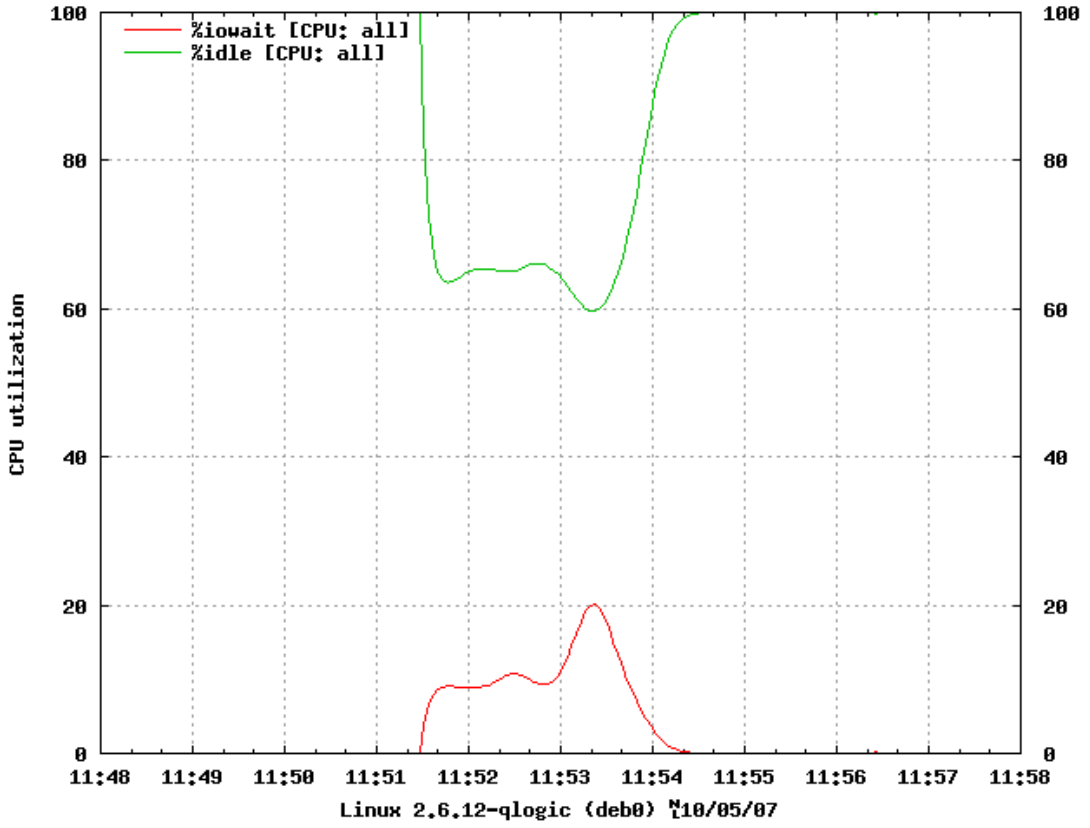


Ilustración V.29: Stripetest 500MB deb0

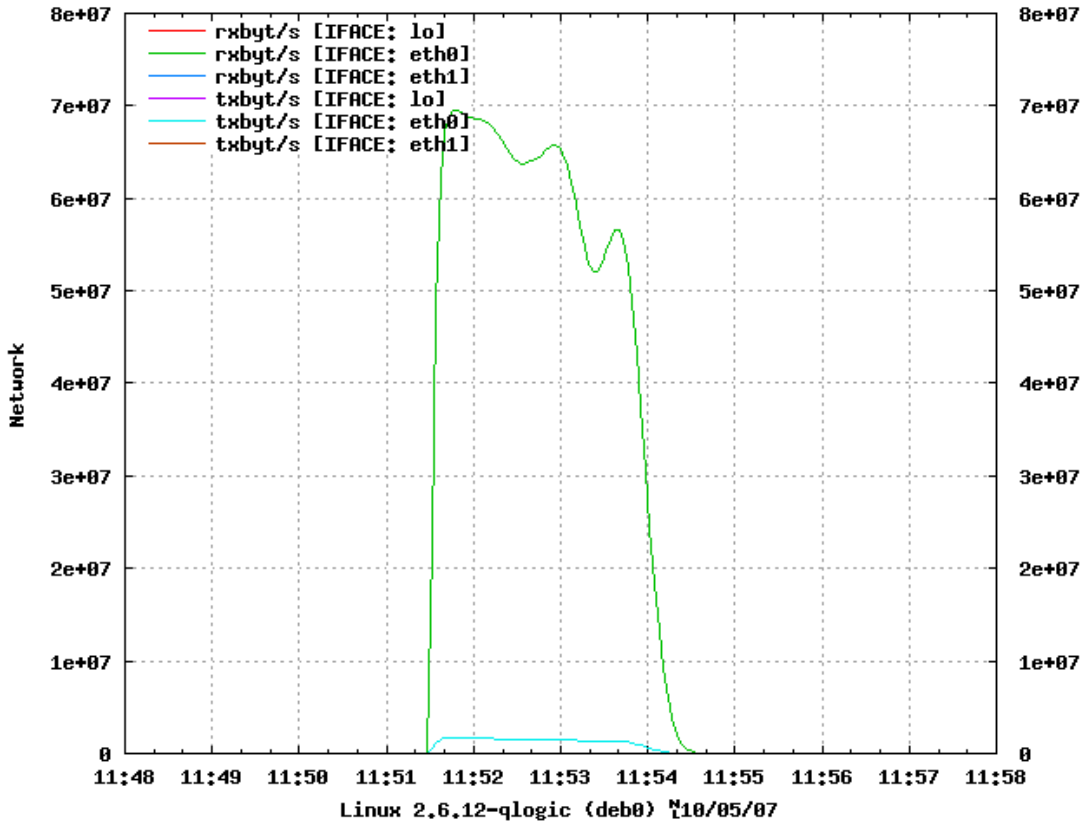


Ilustración V.30: Stripetest 500MB deb0

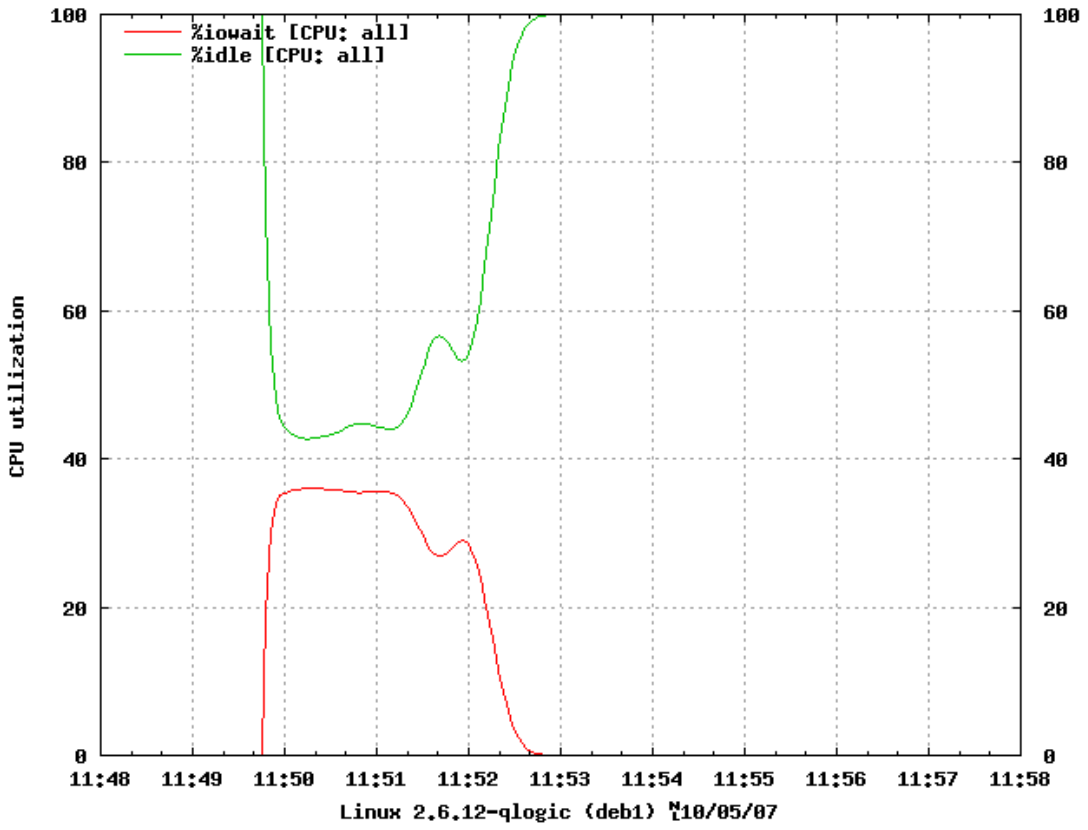


Ilustración V.31: Stripetest 500MB deb1

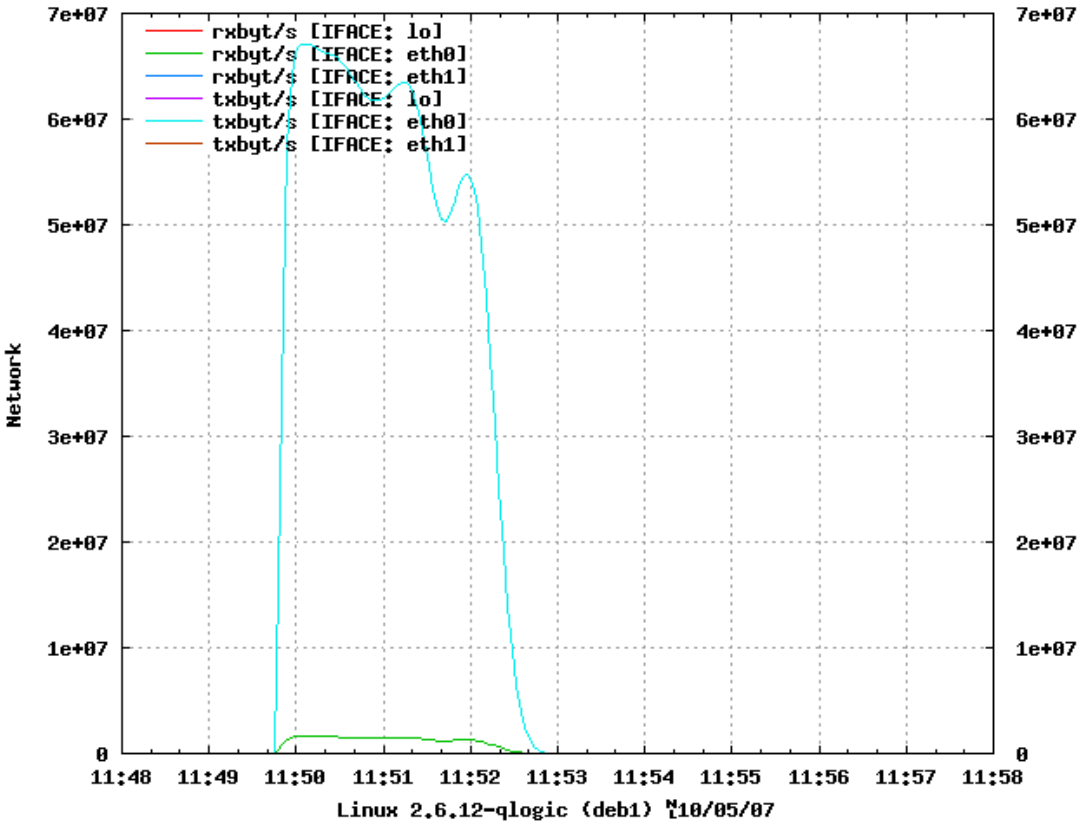


Ilustración V.32: Stripetest 500MB deb1

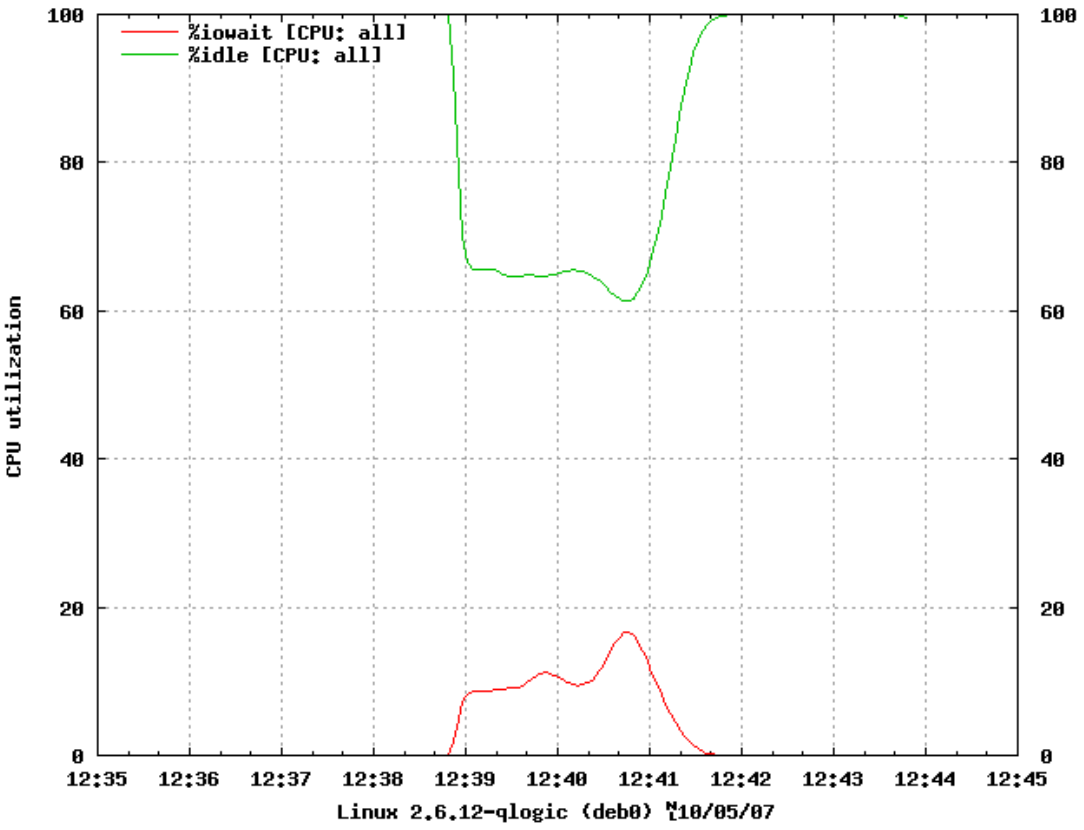


Ilustración V.33: Stripetest 1GB deb0

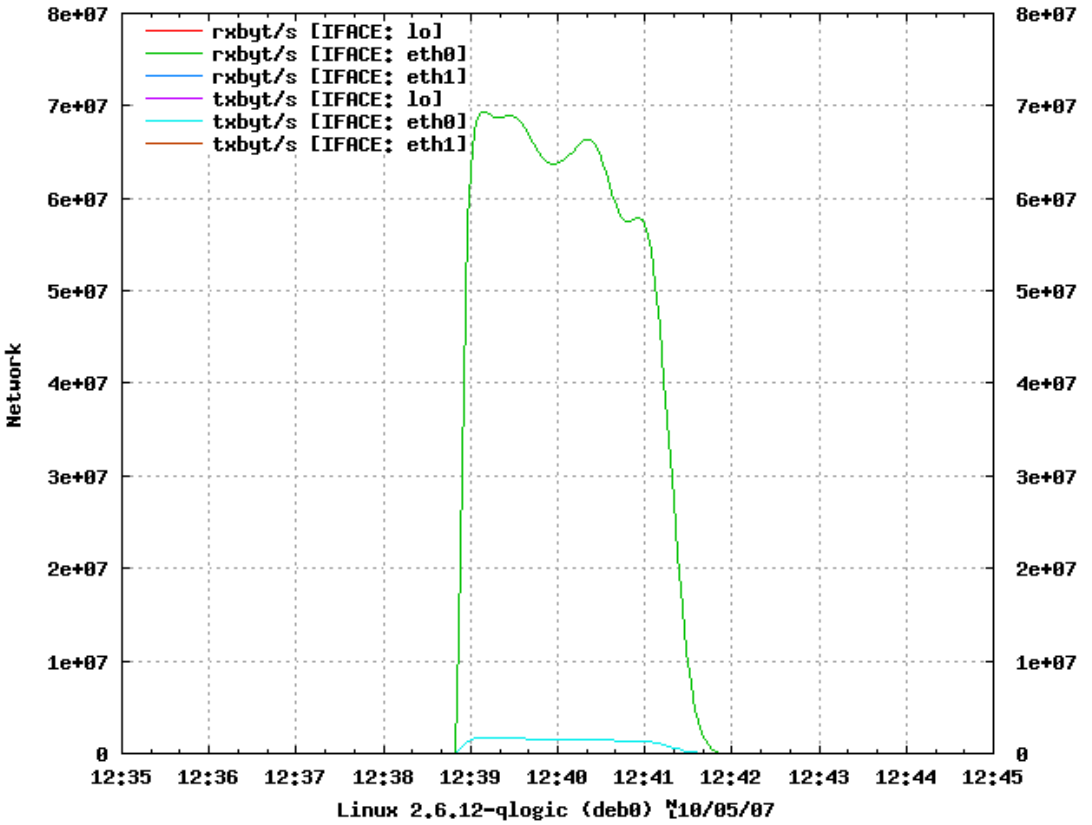


Ilustración V.34: Stripetest 1GB deb0

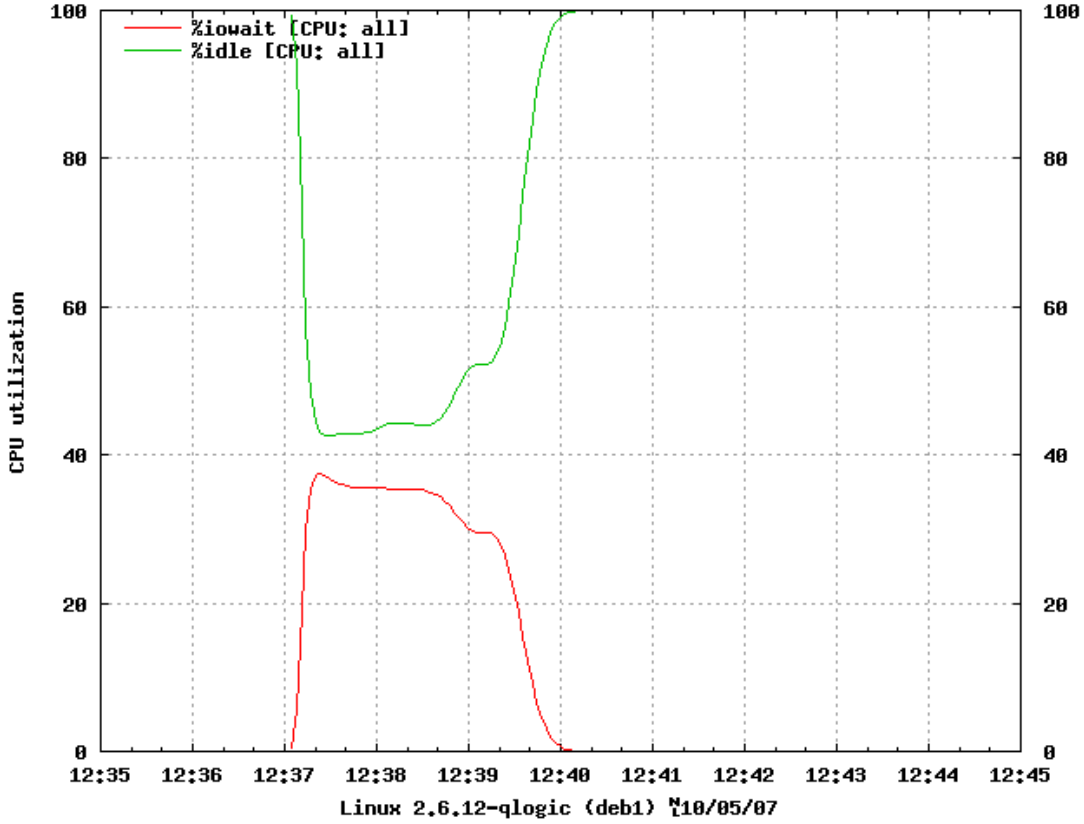


Ilustración V.35: Stripetest 1GB deb1

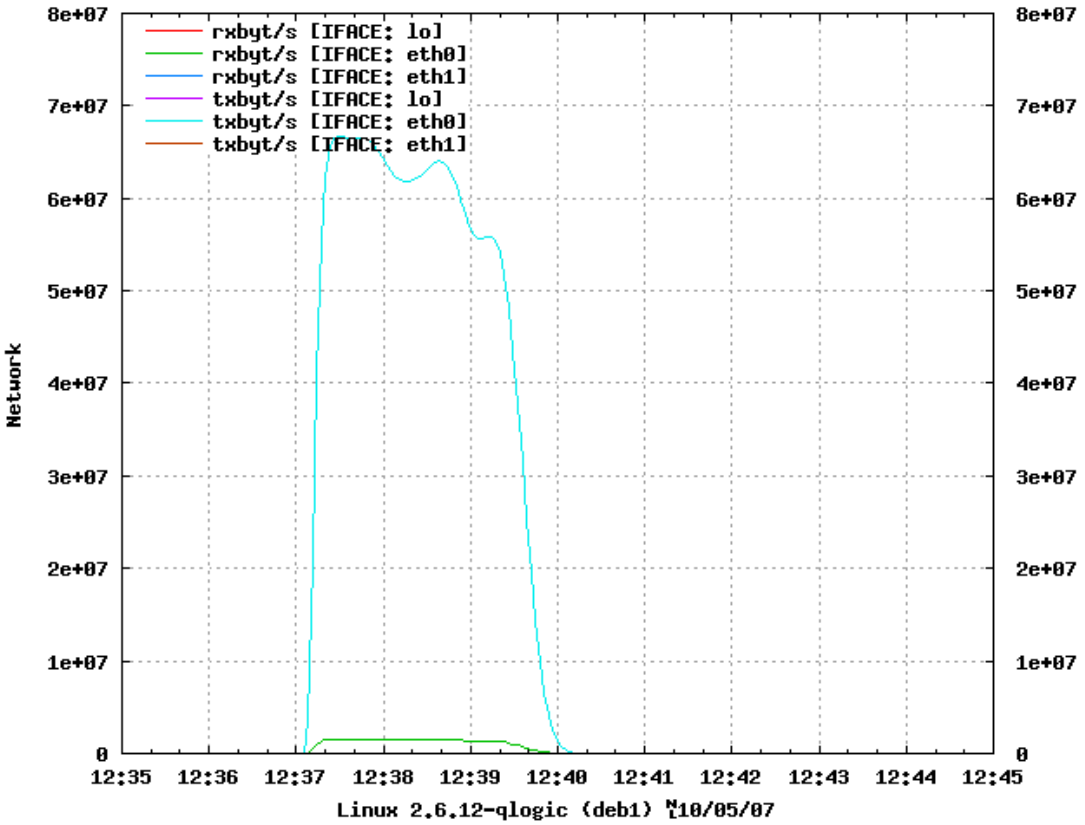


Ilustración V.36: Stripetest 1GB deb1