Distributed file system on the SURFnet network
Report

Jeroen Klaver, Roel van der Jagt

July 14, 2010

University of Amsterdam
System and Network Engineering

Abstract

SURFnet (the Dutch NREN) is expecting a storage explosion, because of new services they would like to offer (for example SURFmedia). Also does SURFnet like to offer a system to its participants to exchange storage using the SURFnet network. Both problems need a new and innovative storage solution, working on the high bandwidth but a geographical disperse network. The new system needs to be scalable, durable, high available, have a performance comparable to a traditional SAN/NAS solution, dynamic and cost effective. In this project two available open source tools are selected (from all available tools), Coda and Lustre. Tests show that Coda and Lustre perform good when there are no problems on the network (latency or failure). The performance of Lustre drops when latency is introduced while Coda remains performing at the same level (this because of its cache). Lustre does not have support for replication. Two problems of Coda are: updating big files which causes the whole file to be uploaded, and the hard configuration. Based on the test results (which show the characteristics of the tools) a design recommendation is made for both tools in combination with each other and additional tools .

Contents

1 Introduction

SURFnet is the organisation which develops, implements and maintains the National Research and Education Network (NREN) of the Netherlands. SURFnet also offers services in the field of security, authentication and authorisation, group communication and video to NREN users. For now the ICT facilities make use of traditional SAN/NAS solutions. Since SURFnet is expecting a storage explosion (because of new services), the traditional SAN/NAS environment has to scale up. But SURFnet thinks there is a smarter method to extend the storage in comparison with the traditional SAN/NAS approach. The new storage solution should be efficient and flexible and meet the requirement of SURFnet (section 2.1). Also the regular SAN/NAS based systems have the problem that they can cause a vendor locking. Based of these problems and the available network a wish for a new distributed storage solution based on open standards is born. A flexible system based on distributed, heterogeneous and multi-tiered storage, with the ability to exchange storage between the organisations within the SURFnet community. The goal is to create more storage for own use, and to enable NREN participants to exchange storage for performance and redundancy purposes. SURFnet has already experience with the commercial products. Beside that SURFnet also wants to know what the open source products can offer for a distributed storage solution.

1.1 Research questions

For this project the following main question is defined:

What infrastructure and open source tools provide SURFnet or the participants a scalable and distributed any-kind-storage solution?

The main question above is defined with the following sub-questions:

- Do the tools meet the requirements of SURFnet?
- How intensive is the maintenance and deployment of the tools?
- What conditions does the network infrastructure need to accomplice?
- What is the performance of the tools on SURFnet's network infrastructure?
- What security features are implemented by the tools?

1.2 Approach

The first stage of the project will be a product selection of open source distributed file systems. First a list of products that are theoretical reviewed will be created. Based on the theoretical review the two most promising file systems will be further researched.

The second stage of the project will be testing the selected file systems. This is done to determine how they react on the geographical disperse and the high speed network of SURFnet. A test environment is created based on the theoretical research of the two file systems. The test environment will be made with multiple VMware ESXi servers.

The last stage of the project will be an implementation design for the selected file systems. Based on the test and the theoretical research a design will be made on how the both file system could be used by SURFnet.

1.3 Previous research

The idea of a distributed file system is already old. The first major implementation was Andrew File System (AFS). This file system is still the base for the newer distributed file systems Coda. Also Lustre has a relation with AFS. It is not direct based on it, but the founder of the project worked before on the Intermezzo project, which was an AFS fork as well. Although the implementations are available for some time, a lot of new project started in the past few years. The newer project are less focused on overcoming the network limits and more on performance. More about the different project/tools in section 2.

2 Distributed file systems

The first part of the research is searching and selecting distributed file systems. This is done based on the requirements that were made by SURFnet. First the requirements of the distributed file system are stated. Second a list of available file systems will be created. The file system from the list will be rated and a comparison will be made to see which two will be looked into. The two file systems that have the most potential will be further researched.

2.1 Requirements

The aim of this project, is to determine the feasibility of building a high-capacity, fully distributed, hierarchical storage solution, exploiting the SURFnet infrastructure. The available storage has to be suitable for general purpose. Next to this, SURFnet has defined the following requirements:

Scalable The system must be scalable in terms of capacity and concurrent access. It must be easy to expand the capacity without lost of performance. And it has to be easy to expand the number of concurrent access, without heavy performance impact.

High available Data must always be available for clients, even in case of maintenance or malfunctioning hardware or software. This means that maintenance and configuration of the system must happen "on-the-fly".

Durability When a single software or hardware component fails, no data may be lost. The system must support replication to other (remote) locations. Additional the system can make back-ups to back-up media such as tapes.

Performance at traditional SAN/NAS level The system must have a level of performance comparable to that found in traditional (non-distributed) SAN/NAS environments. By using different kind of storage within the distributed system, it is possible to offer different levels of performance and capacity. By combining different kinds of storage, the requirements (performance an capacity) for the applications can be met.

Dynamic operation Availability, durability and performance must be configurable per application. This prevents the system to run always at the highest supported level, which reduces costs. Preferable the system must be self-configurable and tunable to optimise its own operation.

Cost effective The system must work with commodity hardware, because of cost-effectiveness. This means that individual hardware is not as scalable, high available, durable and fast like high-end hardware. Systems must be able to compensate this. The system also has to be energy efficient and license fees for software must be limited.

Generic interfaces The system must offer a generic interface to applications and clients. It preferably supports the Portable Operating System Interface (for Unix) (POSIX) file system interface. Alternatively, the system may support a block device interface. This because such an interface can be used to implement arbitrary file systems.

Protocols based on open standards The system must be build using protocols based on open standards as much as possible.

Multi-party access The system must support access by multiple, geographically dispersed parities, at the same time.

2.2 Inventory

The inventory was based on the requirements and the scope of this research. At least the file systems have to be open source and support for distribution over the network. Based on these two requirement the following list is made:

- Coda
- Lustre
- GlusterFS
- XtreemFS
- Ceph
- PVFS
- MooseFS
- Hadoop

The file systems that met the first requirements were reviewed based on the information available on the Internet. The results of the review can be found in appendix A. In short: Hadoop is a framework that can be used by applications to store data in a distributed environment which is not SAN/NAS like. The used method looks like the one used by Google File System (GFS). MooseFS, PVFS, Ceph and XtreemFS are heavy developed and are either missing functionality or not mature enough to be used in production. GlusterFS, Lustre and Coda are the three file system that have the majority of the requirements and are mature enough to be used. GlusterFS and Lustre are very comparable in functionality and majority. Although GlusterFS is not chosen because on first sight it does not seem to be possible to keep the storage volume available when one node fails. Also the security is not handled within GlusterFS which could be a issue when multiple participants are using the same storage. The downside of Lustre is the lack of replications, but is does have security features and good documentation. Based on the arguments above Lustre is selected over GlusterFS. The Coda system is selected because it uses a different architecture. This architecture uses more tiers in the file system. This gives it an advantage over GlusterFS and Lustre.

In the section bellow (section 2.3 and 2.4), Coda and Lustre are described in more detail based on the requirements in section 2.1.

2.3 Coda

Coda has been developed at Carnegie Mellon University (CMU) since 1987 by M. Satyanarayanan, and it is still active developed. Coda is used by CMU itself. It started with replicating file servers but develop into a more featured architecture. In 1980's the networks were not that stable and fast as they are these days. This is why the caching client was developed, so less bandwidth was required and clients could continue working even when the connection was broken. These days it can be used for several system, because of this caching client. For example, the client can be used as a web server, distributing Linux distributions.[1]

2.3.1 Architecture

The architecture of Coda has six base components:

System Control Machine (SCM) The SCM is the so called master server. This server is used to manage the Coda environment. During the configuration of the clients and servers, a connection with the System Control Machine (SCM) is made and the configuration is retrieved. The SCM is not redundant, but in case of a failure the whole cluster keeps running. Only no volume or user management can be applied. The SCM can be installed on a separate server, or can be combined with a file server.

File server The file server is used for the file storage itself. For the meta data no separate server is required and is stored on the file server self. A file server can contain multiple volumes and volumes can be replicated between multiple file servers.

Auth server Every server runs the authentication service. This is used to authenticate the user. Based on a token (which the client receives form the SCM during login) the rights for the user on a volume are determined.

Update server The update server is unique in the Coda infrastructure. This is a process running on the SCM. When a configuration, Access Control List (ACL) or user modification is made, the update server distributes it to the servers.

Update client All the servers run the update client, so the server can reach them for update information.

Client Using clients a connection to the Coda infrastructure is made. The client has the capability to cache information local. So there is no need to upload a file direct to the file server, when a file is saved. Or a file does not always has to be retrieved from the file server when a read is executed.

The file servers do not require a specific file system. For the meta data and log information a separate raw partition is required. And if possible a separate disk for each. This to improve performance significant. But in case of a test scenario, both can also be stored as a file on the root file system. The size of the meta data and log partition/file can not be changed after installation. Further configuration data is by default stored in `/vice`. The data store (or volumes, more about volumes later) is by default located in `/vicepa`. If multiple drives are available for the data store they are mounted under `/vicepb`, `/vicepc`, etc. As noted before, for the data store, each file system can be used. The ability of multiple data stores per Coda file server, makes the server more scalable.

To prevent data lost or unreachable data, Coda makes use of replicates. The client contains a list of all replicates. As long as the client can reach one of the replicates, the data is still available. As soon as the replicated are connected again, the servers will automatically synchronise. To store data, a so called volume is created. A volume can be stored on one till eight machines. The replication is configured per volume. The addition of a replica of a volume after creation is possible. Within Coda no capabilities to stripe data over multiple machines are available.

Also the client has no specific file system requirements. The client makes use of a kernel module. Using this module, the mount `/coda` is created. In this folder the Coda root volume is mounted. In that folder the other volumes can be mounted. Using the information retrieved for the System Control Machine (SCM), the client knows were to find which volumes, and where replicates are stored. Also the SCM distributes the Access Control List (ACL) for the volumes. A very nice feature available in the client, is the cache. This makes it possible for often used files to reach traditional SAN/NAS level performance. Also this cache is file system independent. Using some advance technologies, the cache can be used for read and write operations without the change on conflicts between clients and servers. More information about this can be found in [2, A.S. Tanenbaum and M. van Steen, Distributed Systems].

Notice there is no need to install the client software on the application server itself. It is also possible to create an NFS share on the Coda client. iSCSI is not possible, because Coda does not offer a block device. iSCSI can also store its data in a large file, instead of a block device, but Coda has a problem handling large files. More about large files later in this section.

Although the cache is a nice feature, it has some limitations. In case of a large file where only one bit is changed, the whole file has to be uploaded again. Also it is important to notice that files stored on the system can never be larger as the cache size. Global the cache can work in two ways: write behind and write through. In the first case the files are just stored in the cache and later transmitted to the Coda file servers. This works well, but the transmission of the data to the Coda file servers is often slow, this because of the auto regulating bandwidth mechanism in Coda. If this mechanism is disabled, the Coda client switches automatically to a write through system. This has the disadvantage that the process stays locked, until the data is copied to all Coda file servers. When data is written on the client, the will go through following cyclus:

1. Data copied from the process to the Coda client cache;
2. Data copied form the Coda client to the nearest Coda file server;
3. At the Coda file server the data is copied from a buffer to the real data repository;
4. The Coda file servers replicates the data to all replicates at once.

So the data is processed four times before the process is released.

Coda has back-up functionality implemented. For users it is possible to connect to a back-up volume and restore their own files. To create and restore complete volumes, the following procedures are defined:

- Backup
    1. Freeze one of the replicates of a volume and create a read-only clone;
    2. Dump the read-only clone to a local disk;
    3. Backing up the dumped data to a suitable archive media (e.g. tape drive).
- Restore

1. Retrieving appropriate full and incremental dumps from the archive media;
2. Merging the full and incremental dumps to the time line of restoration;
3. Restoring the fully integrated backup to the Coda file system.

For the network used by Coda no special requirements are specified. Because of the caching client, the bandwidth can be limited and the latency might be high. As long as there is no structural capacity shortage, the client can cache all the data send to the cluster and catch up during the night (as long as no cache overflow occurs). If the cache appears to be flooded, the client will slow down the writing process. If this does not solve the problem, the Coda client will lock the process. For reading it may take a while for the first read, but later the client gets served from its cache.

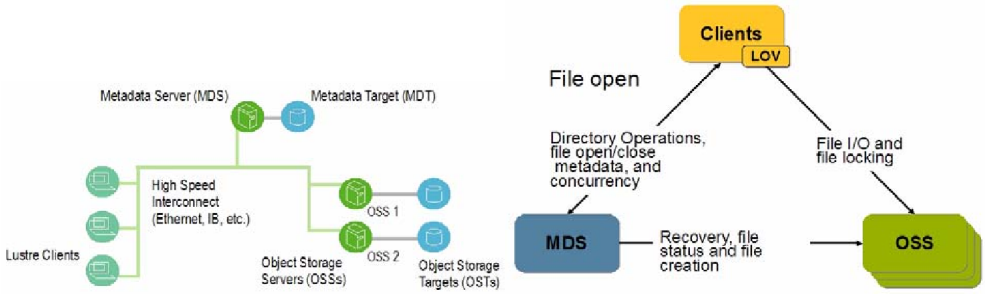
The Coda client makes the system scale well, because the amount of clients is unlimited, and they will unload the Coda file servers. This makes the need for striping file servers unnecessary. Also the amount of file servers is unlimited, and by making use of volumes, the usage of file servers is transparent. The only limitation is the throughput of one client, in case of an application with high requirements. But later in this document (in section 4.2) this problem will be solved. Also is the client a single point of failure, also this will be discussed later. Although because of the scalability, no special and expensive hardware is required.

Adding a new Coda file server is as easy as installing the software, and telling it where it can find the SCM. For the clients it is almost the same thing. But configuring the SCM is a bit harder. There are a lot of parameters which may influence the performance significantly. Most are not well documented and some have unexpected consequences. Also is the system vulnerable for errors during synchronisation. Those errors are easy to fix using the supplied tools, but they do require human interaction.

The Coda client and servers communicate using the Remote Procedure Call version 2 (RPC2) standard. Files are transported using SSH File Transport Protocol (SFTP)(File Transport Protocol (FTP)over Secure Shell (SSH)). Both are open standards and well documented. Using SFTP prevents anyone to listen in to the stream. Next to SFTP Coda makes use of authentication tokens. So a user must authenticate against the SCM. After successful authentication a token is received, which is valid for 24 hours. Based on these tokens, ACLs can be build from volume till file level.

2.4 Lustre

Lustre is a distributed file system first thought of in 1999 by Peter J. Braam. At that time he was a system scientist at CMU. He was already active in a fork of Coda, named InterMezzo. The main goal of the fork was to add performance. In 2002 the development of InterMezzo stopped and Lustre was started. The development of Lustre was done within his own company named Cluster File Systems. Later in 2007 the company was acquired by Sun Microsystems to use the Lustre technology in their file system Zettabyte File System (ZFS). Nowadays Sun has been taken over by Oracle and Lustre remains being developed and supported.



(a) The components in a Lustre system

(b) Simplified Lustre system

Figure 1: The components of the Lustre system. The MDS serves the meta data stored on the MDT. The OSS serves the file data stored on the OSTs. The clients receives both the meta data and the file data.

2.4.1 Architecture

Lustre has five base components:

Meta Data Server (MDS) The Meta Data Server (MDS) makes the meta data stored at the MDT available for the clients. The MDS manages the names and directories within Lustre and provides network request handling for one or more MDT.

Meta Data Target (MDT) The Meta Data Target (MDT) stores the meta data of a MDS. Each file system has one MDT. An MDT can be available to many MDS, although only one should use it. Other MDS can be passive available is the active MDS fails.

Object Storage Server (OSS) The Object Storage Server (OSS) provides the file I/O service and network request handling for the OSTs. Normally one OSS serves between 2 and 8 OST servers.

Object Storage Target (OST) The Object Storage Target (OST) stores the data of one or more OSSs. A Lustre system can have multiple OSTs, each can serve a subset of a file. A logical object volume does the management of the file striping over the different OSTs.

Clients Clients are the computers that mount the file system on their local system.

The file system is based on the ext3 file system. Normally the file system consists of inodes which point to the blocks on the disk where the actual data can be found. On the Lustre file system the inodes points to the objects associated to the file. The objects are chunks of the file and can be distributed on several Object Storage Server (OSS)s. The inode information is stored at the Meta Data Server (MDS) server. So if a clients wants to open a file, it will contact the MDS first and retrieve the inode information. The information contains the object(s) related to the requested file. Second the client will request the object at the different OSSs.

Distribution over the nodes can be done using striping, this can be done per file or directory. Using striping files are distributed along the OSSs, the distribution can be configured at client side. So for example all servers can be used but also a maximum number of OSS. The chunks of the file are placed using round robin until the free space drops under the 20%. Then there will be smart placement of the chunks so a server will not run full while other have a lot of free space.

Replication is not done within a Lustre network. The files are distributed over the storage node and those are assumed to be reliable. To get more reliable nodes a fail-over can be configured. There are two kind of fail overs: active/passive and active/active. The active/passive is commonly used for the meta data server. The active meta data servers is doing the work while the second waits to take over, in case the active servers fails. This can be achieved using Heartbeat, this is later explained in section 4.1.2. The downside of this approach is the need for an extra server which is not used and not helping to add performance. The second approach is to get an active/active fail over, this is normally done for the storage nodes. Both servers have the files and both servers are serving the data. This gives extra performance since a file can be downloaded from multiple nodes. The fail over mechanism only works for the server and not for the storage device. So the fail over servers share the same physical storage.

The backup of Lustre can be done by copying the files from the mounted partition or use "dd" to fully copy the block device. Using "dd" it is possible to copy the whole file system what makes it more easy to restore. In case of an almost empty large partition, the usage of gzip can be useful to compress the bulk of 0's in the backup file. To copy only the files the partition has to be mounted not as a Lustre file system but as a normal file system. Now the files can be copied to another location. Although Logical Volume Manager (LVM) volumes can be used by Lustre this will decrease the performance of the writing. LVM does offer snapshots which are good for backup purpose. In the upcoming version of Lustre there will be an feature to be able to replicate a volume based on transaction logs[3].

Lustre uses Lustre Networking (LNET) for the communication within the Lustre network. LNET runs on top of the network drivers of the Operating System (OS) and can run on multiple interfaces at the same time. The network stack can support multiple types of network interfaces, including Ethernet and InfiniBand. The ability to connect multiple interfaces makes the support for link aggregation and routing more interesting. Within LNET it is possible to define what routes should be taken and what link can work together. Although the link aggregation can only work if the kernel and the other endpoint support it. Normally this endpoint is a switch which needs support for link aggregation[1]. Another feature of LNET is Remote Direct Memory Access (RDMA). This feature grants a network interface direct access to the application memory. This eliminates the need to copy the data to the data buffers from the OS and thereby reducing the CPU work. Although this does not work on Ethernet at the moment.

The scalability of Lustre is good. The MDS is the only components that can have a limiting factor since it can not be used multiple times. To add performance or capacity multiple OSS nodes can be used. This can scale up to

[1]Link aggregation is defined in the standard: 802.3ad Dynamic Link Aggregation

ten thousands of nodes, which is most likely enough for SURFnet.

For security Lustre can be configured to use Kerberos. Kerberos can be used to authenticate the nodes (OSS, MDS and clients) with each other. In this way client can be prevented to connect to nodes that they should not be able to connect to. Also the Remote Procedure Call (RPC) send over the network can be protected against eavesdropping and modification. This adds privacy to the use of the network data and prevents leakage of the data. Beside the security added by Kerberos, Lustre also has an access list to direct who can access what data. This in combination with pools of data that can be configured means that users can set with who he is working together. Other clients can not profit from that until deals have been made.

More information about the architecture can be found in the operations manual of Lustre [4] or a paper from P.J. Braam about Lustre [5].

3 Benchmarks

A test environment for Coda and Lustre will be build to test the performance and behaviour of both architectures. In the test environment the disk performance will be measured and the behaviour during failures will be noted. This based on the requirement for performance comparable to traditional NAS/SAN environments, and high availability (section 2.1).

In this section, first the test environment will be described including the tools used for benchmarking. Second the different stages of testing will be discussed. The first stage contains a null test of the test environment to determine the limitation and characteristics of the test environment. The second stage will be a null test of Coda and Lustre running in the test environment. The third stage consists of tests done to see how the architectures reacts on different conditions like latency and failures.

3.1 Test environment

The test environment is based on the theoretical research done in the earlier section. The test environment is kept simple to get more experience with both tools and because of time restrictions.

Also some practical experience already has been taken into account of setting up the tools. The test environment is based on the theoretical research and practical experience with Coda and Lustre. Based on all experience a simple environment is created to be able to test the characteristics of the architectures.

3.1.1 VMware ESXi environment

The test environment is build with use of four ESXi servers. The hardware of each server is a little different. The machine have the following specifications:

- VMware ESXi 1
  - Dell Poweredge 1750
  - Intel Xeon single core 2.8 GHz
  - 2048 GB RAM
  - 3 * 136 GB SCSI hard disks
  - VMware ESXi version 3.5
- VMware ESXi 2
  - Dell Poweredge 1750
  - Intel Xeon single core 2.8 GHz
  - 2048 GB RAM
  - 3 * 136 GB SCSI hard disks
  - VMware ESXi version 3.5
- VMware ESXi 3
  - Dell Poweredge 1750
  - Intel Xeon single core 2.8 GHz
  - 1024 GB RAM

  - 3 * 136 GB SCSI hard disks
  - VMware ESXi version 3.5
- VMware ESXi 4
  - Dell Poweredge 1850
  - Intel Xeon single core 2.8 GHz
  - 2048 GB RAM
  - 10 GB hard disk RAID 1
  - VMware ESXi version 4.0

On all the virtual nodes the virtual Intel e1000 network interface is used.

3.1.2 Coda

Figure 2 shows the Coda test environment. Two Coda servers are used. One of them is also used as the SCM. One volume is replicated to the both servers. Further two clients are used, although the benchmarks will run on one client. The second client can be used to see how fast changes are propagated (this because of the caching capabilities of Coda). Each of the machines runs on a separate VMWare ESXi server. So each machine has its own VMWare ESXi server. The Coda file servers have each three virtual disks, each on a separate data store. One used for the operating system, one for the data repository and one for the meta and log data. This to prevent interference.

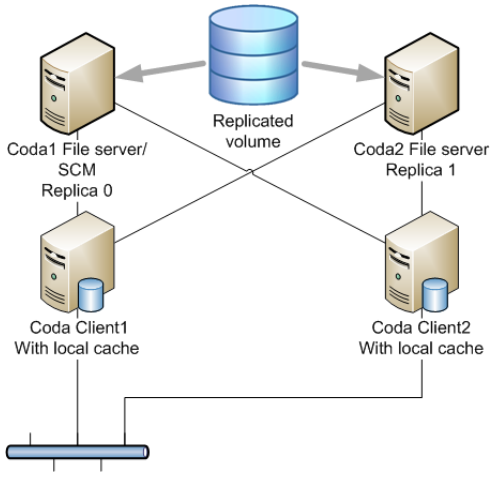


Figure 2: The Coda test setup.

3.1.3 Lustre

The test setup for Lustre is kept basic as can be seen in figure 3. Only two file servers are used since early tests showed that two file servers are already at the limit of the physical disks. The nodes are separated on the four VMWare ESXi servers. Since the server nodes require a separate storage device connected for

the storage of data or meta data, each server node has two hard disk attached. Each hard disk is put on a different data store so the OS disk I/O does not interferences with the (meta) data disk I/O. On the client the mounted directory has the setting to use striping so the files will be distributed over the storage nodes.

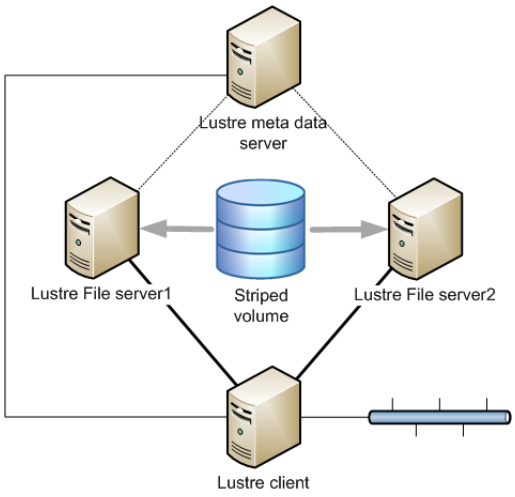


Figure 3: The Lustre test setup.

3.1.4 Used benchmark tools

To benchmark the performance of the file systems the following tools will be used: Iozone and Bonnie++. Both are open source hard disk benchmark tools which can be used to test the read and write performance of the hard disk. Both benchmarks tools can perform sequential and random write/read benchmarks. Iozone has more configuration parameters that can be set and is used in the following research [6, S. Carlier and D. Muller, SSD Performance]. The configuration parameters makes it possible to tune the benchmark and save time. Bonnie++ was recommended by SURFnet and has the advantage to store the average CPU load with the benchmark results. This can be useful to see whether the CPU is a bottleneck. Different block sizes are used to see if the architectures have an advantages if using big or small block sizes.

The network is tested using Iperf and ping. Iperf is used to test the network bandwidth using a TCP connection. The test consist of a server and a client that transfer data between them. The test is performed multiple times to get an average but also to see the fluctuations on the network. Ping is used to test the latency between the hosts. A set of 100 ICMP echo messages is send and the latency is stored. This can be used to see the average latency between hosts but also the peaks or network hick-ups which can occur.

The benchmarks are invoked using the following commands including the noted parameters:

Iozone `/usr/bin/iozone -s 2048000 -f /mnt/sda1/tmp2.test -b /home/roel/codaclient1_iozone_1.xls -r 8k -r 256k -r 8m -i 0 -i 1 -i 2 +d`

Bonnie++ `/usr/sbin/bonnie++ -n0 -d/mnt/sda1/ -s1024 -x3 -q -u root > codaclient1_bonnie++_2.csv`

Iperf (client) `/usr/bin/iperf -c 192.87.110.159 -f M > 4_3iperf.txt`

Iperf (server) `/usr/bin/iperf -s`

Ping `/bin/ping -c 100 1.1.1.1 > 4_3ping.txt`

3.2 First stage

In the first stage, a reference test will be executed. This to get insight in the characteristics and the limitations of the used VMWare ESXi test environment. The first test is running the file system benchmark tools on a virtual server on its local storage. Using this test a reference point is made to know what the disk performance of the VMWare ESXi test environment is. Since the VMWare ESXi servers have multiple disks, there will be a virtual disk on each of them.

The second part of this stage is testing the maximum network bandwidth and the average network latency of the test environment. A configuration as displayed in figure 4 is build. The location of the machines will be changed, to get information about all the systems.

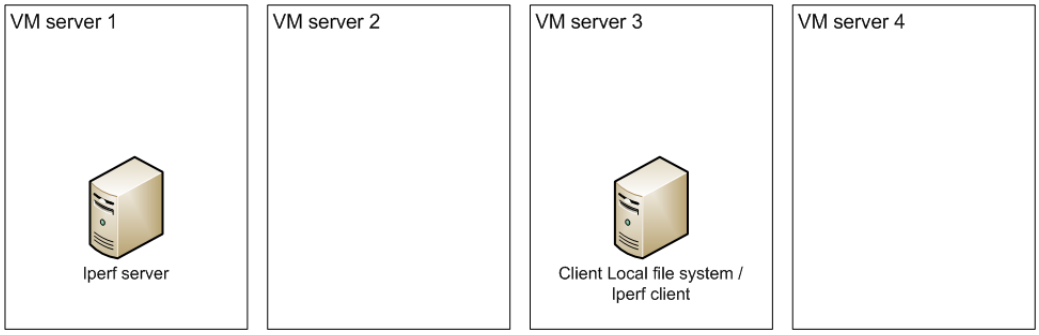


Figure 4: The reference test configuration.

3.2.1 Results

In this section the results of the first test stage are presented. First the hard disk benchmark results are presented. Next the network benchmark results are discussed.

Disk benchmarks
The results of the disk benchmarks from Iozone are displayed in figure 5. For each test the lowest, the maximum and the average values are displayed. The read and write tests are comparable in speed. Also the different block sizes do not make much difference. With the random tests the different block sizes make more differences. This because with a larger block size, data is read in a more sequential way (blocks are sequential on disk). Notice that the random read speed with 8192KByte blocks is faster as the sequential read speed. This

is exceptional. Normal this is the other way around. Probably it has something to do with some smart read ahead functionality within VMWare ESXi.

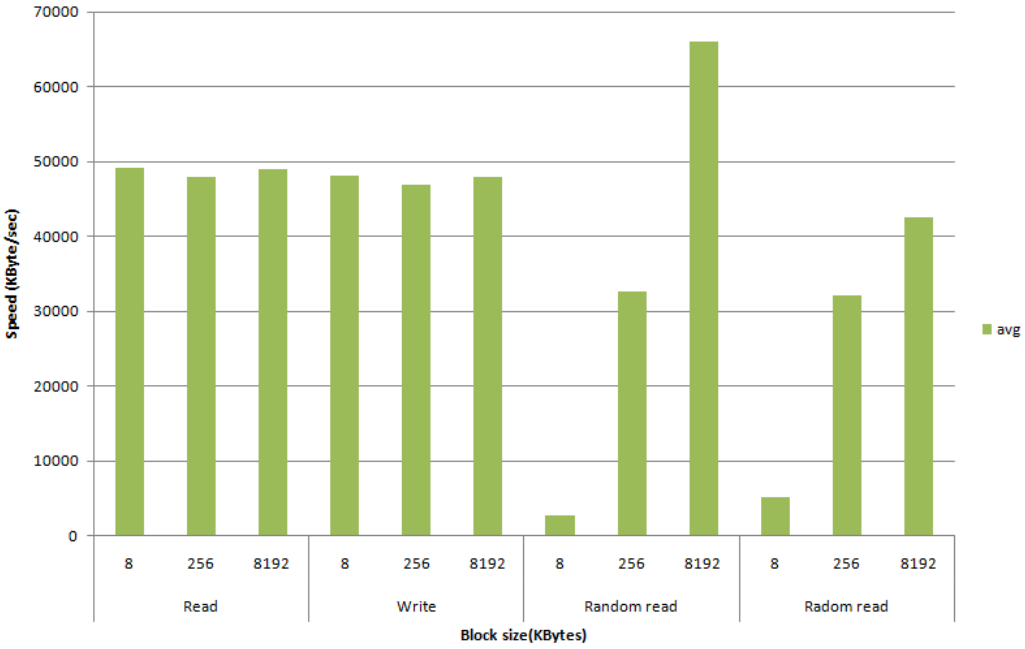


Figure 5: Disk null test results using Iozone.

When looking at the data in figure 6 it can been seen that the first disk of each VMWare ESXi server is slower. This is because ESXi itself and the virtual machine have their OS installed on that disk. Also on the first virtual hard disks snapshots are activated which may cause extra overhead.

The results of the Bonnie++ benchmark can be found in figure 7. The results are lower in comparison with the previous results from the Iozone benchmark. This is all caused by the different test methods. Notice that this does not mind in this case, because we will only use those results to compare the results of Coda and Lustre with the local disk speed. And not to determine the real throughput of the virtual disks. Also here we see the lower performance of the first disks.

The raw data can be found in appendix B.

Network benchmarks

The bandwidth benchmarks show a big difference between benchmarks on the same link. This was not expected since the environment is connected using a dedicated 1 Gbit switch. To make sure the deviation comes not from the switch, a test is ran over a direct cable. This did not take away the deviation. The final bandwidth results can be found in table 1. The numbers are the average of three tests. This to tamper the differences in the results. The raw results can be found in appendix C in table 7.

The latency is bellow the one msec between the servers as can be seen in table 2. This is based on a 64Byte icmp ping. The latency has some peaks which

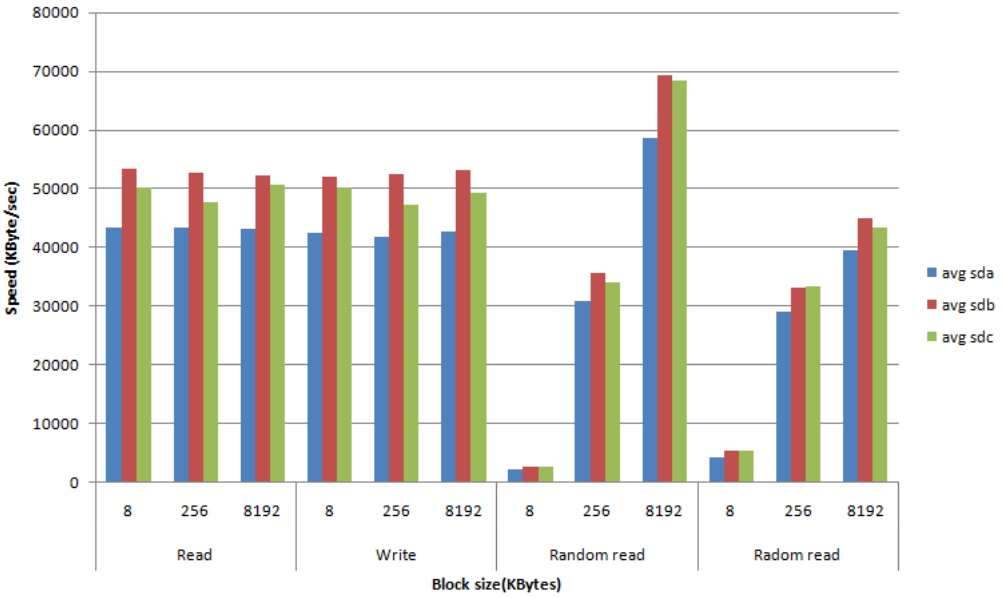


Figure 6: Disk null test results using Iozone, average per data store.

		from 1	2	3	4
to	**1**	x	65.8	67.0	65.1
	2	65.2	x	67.5	72.2
	3	66.9	67.9	x	68.0
	4	82.5	86.5	82.1	x

Table 1: Network bandwidth null test results using Iperf (in MBytes/sec).

show that the networking can be inconsistent like the bandwidth benchmarks already showed. Those can also be seen in the more detailed information in appendix C in table 8.

3.2.2 Conclusion

The results show the limitation of the test environment. The first disk that is in use for VMware ESXi itself performs a little bit slower then the other disks. The networking does not reach it full potential and can vary over time. This is due the use of VMware ESXi.

3.3 Second stage

The second stage is to test Coda and Lustre. This is done to see the impact of the added middleware (Coda and Lustre) on the performance and to act as a reference for stage 3 where the condition of the network will change (higher latency and failures).

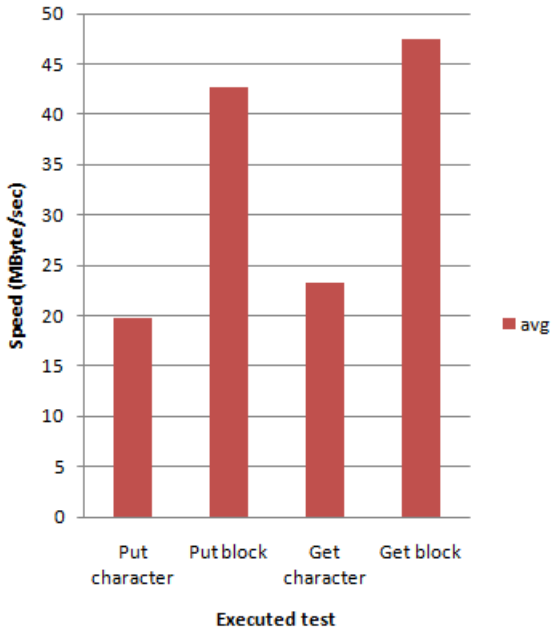


Figure 7: Disk null test results using Bonnie++.

		from			
		1	**2**	**3**	**4**
to	**1**	x	0.642	0.390	0.462
	2	0.424	x	0.727	0.501
	3	0.386	0.438	x	0.483
	4	0.395	0.512	0.479	x

Table 2: Network latency null test results using 64Byte icmp ping (in msec).

3.3.1 Approach

In this stage, the performance of Coda and Lustre will be tested. In this test the tools are set up in the test environment. To limit the performance interference with each other, the servers and clients are put on different VMware ESXi servers and all other virtual machines are shut down. The benchmark will be executed on the client system.

3.3.2 Results

The results of the Iozone benchmarks can be seen in figure 8. The sequential read performance of Lustre is standing out. Since Lustre is using a striping configuration the files can be read from 2 servers at the same time. Coda is also performing slightly better then the null test, this is because of the caching that is done by Coda. The read performance drops when random read is done, especially compared to the good result of the sequential read. The cause of the performance drop is the added time that is needed to request the meta data. The random read of 8192KByte blocks of Coda and the null test are equal or

higher as the sequential. This is not what was expected since normally the random read takes more time. As mentioned before this is probably caused by the used VMWare ESXi environment. The results of the other test does not show any surprises, Coda and Lustre are a little slower. On the other results show that Coda and Lustre perform a little bit less as the null test. This is caused by the overhead of using the middleware. The meta data needs to be requested, the files need to be requested and also the network is in between.

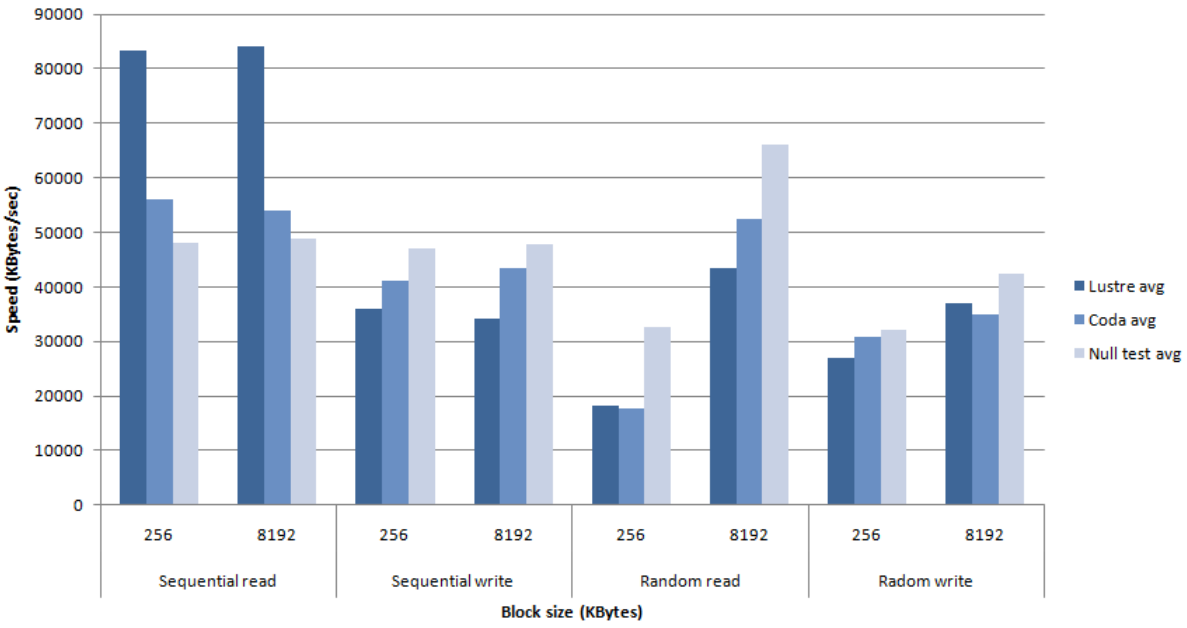


Figure 8: Iozone benchmark of Coda, Lustre and for comparison, the null test.

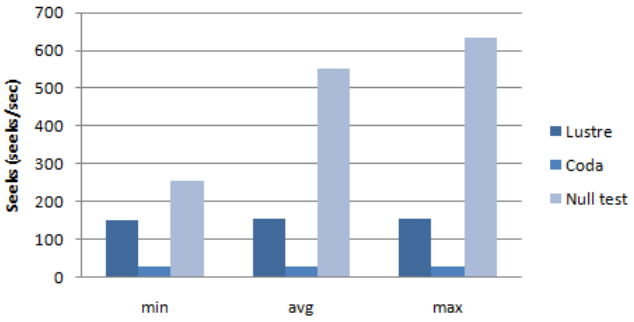


Figure 9: Bonnie++ benchmark of Coda, Lustre and for comparison, the null test.

The results of the Bonnie++ benchmarks, can be seen in figure 9. Only the seek test is used. Actually this is a random read with a single of a single block. This shows a much lower performance of Lustre and an even more dramatic performance of Coda. This is all caused by the middleware. Probably most by

the meta data servers. Al thought the performance is lower, it is more stable. The longer seek time also gives the explanation for the differences for random read in figure 8. Coda reads faster, but Lustre seeks faster.

The other performance results generated by Bonnie++ are not used. This because of the high deviation of the results for Coda as can be seen in appendix D table 9.

Note
Notice the Iozone benchmark of Coda is performed with a 1GByte test file, instead of the 2GByte file. This because of some problems during the Coda benchmark. Coda did not seem to be able to handle the 2GByte file. This is probably related to the used Coda cache size of 2,5GBytes.

3.3.3 Conclusion

Overall the test results show that Coda and Lustre are capable of performing almost as fast as the null test. With sequential reading Lustre has the benefit of reading from two servers at once. Coda has the benefit from its cache when it comes on reading and writing data (both in random as sequential way). Al thought the performance is good, it is still lower as the null test. This is caused by the extra overhead of the middleware.

3.4 Third stage

In the third stage, the test environment will be altered to simulate conditions that can be expected in the SURFnet network. Latency and failures will be introduced. Those parameters are selected based on the requirements (section 2.1).

3.4.1 Approach

The test environments of the second stage will be used with some small modifications to the network. This is done the see how Coda and Lustre react when the following variables will be changed:

Latency Within the SURFnet network the storage nodes and client can be at different geographical location. The distance between the participants in the network introduce latency. How will the systems react if the latency is increased? The main goal of this test is to test whether Coda and Lustre can handle the increasing or changing latencies. The latency that is introduced is 20ms RTT, that is the maximum latency that is allowed on the SURFnet network[7]. In the Lustre setup the latency is added at the file servers. Adding it to the meta data is did not give any major issues. In the coda setup every node has added latency.

Failure Failures can happen within the system. How do Coda and Lustre react when one of the following components of the system is taken out.

- Client
- Meta data server (if separate available)
- Storage server

- Temporary network link outage

The latency variable will be adjusted using the Linux tool called Traffic Control (TC). This tool can introduce latency. This can be done by installing this tool on the Coda and Lustre servers and clients, so there will be no need for separate machines and networks.

3.4.2 Results

Latency
Adding latency is hard to handle for Lustre. The results from Iozone can be seen in figure 10. It shows a significant performance drop. The seeks/sec from Bonnie++ (table 3) show also a reduced amount of seeks. This performance drop can be explained by the fact that Lustre is optimised for a low latency network (latency of one msec or lower). The used protocol makes use of acknowledge messages. By waiting for the acknowledgement before sending the next message, the higher round-trip time has its impact.

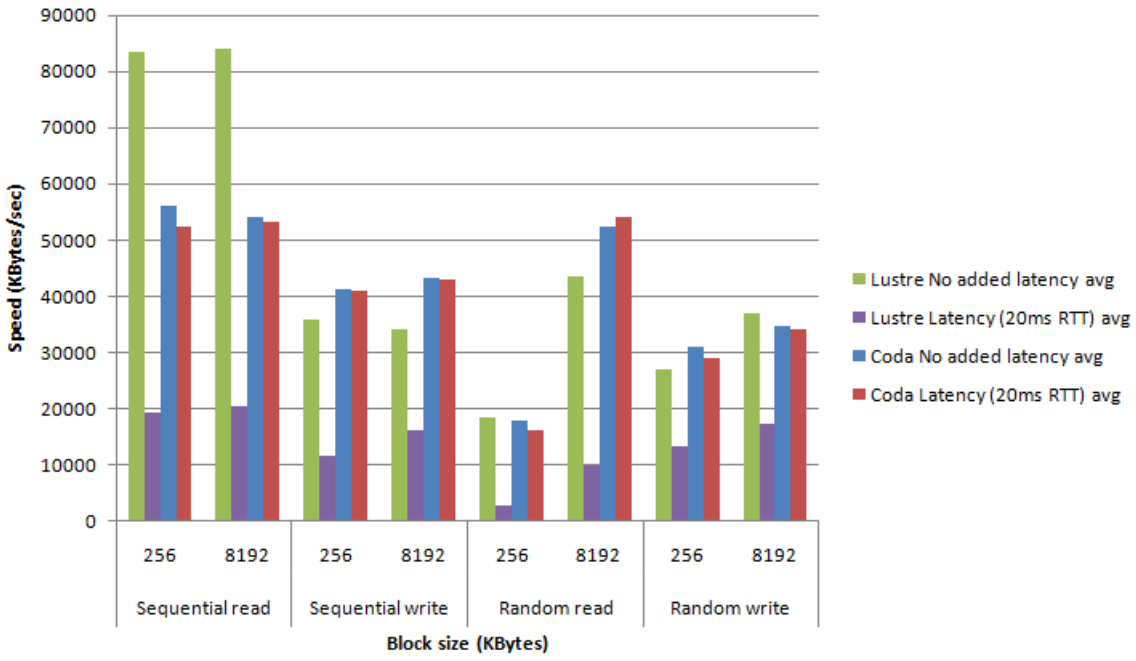


Figure 10: Iozone benchmark of Lustre and Coda with an added latency of 20ms RTT.

In contrast with Lustre, Coda performed very well. There is almost no performance difference as can be seen in figure 10. This can be explained by the usage of the cache. The benchmark actual ran on the local disk (were the cache is stored). Although the synchronisation between the client and the server was slowed down, this did not influence the benchmark.

Just as Lustre, Coda also has higher seek times (so does less seeks/sec) as can be seen in table 3. Despite of the cache, for seeks the Coda client has still to connect to the meta data server, which is on the delayed network.

		min	avg	max
Lustre	**No added latency**	151.4	153.2	155.4
	Latency (20ms RTT)	28.7	29.0	29.2
Coda	**No added latency**	27.6	28.2	28.4
	Latency (20ms RTT)	9.5	9.8	10.0

Table 3: Bonnie++ benchmark on Lustre and Coda with an added latency of 20ms RTT. (seeks/sec)

Failure
Lustre is not designed to handle with failure. Default when a node or link is down it means the files or chunks on that node can not be reached. In practise this means that a copy job will block and will wait until the file becomes available again. After the time out has been reached the copy will fail.

Coda is more designed to handle failures. When the SCM fails, and the client has a valid token, it can continue working. Also it is not possible any more to create new volumes or configure ACLs. In case of a failing file server, this has no impact, as long as the volumes on the file server are replicated. The client will notice the failing file server and switch to one of the other replicates. Also in case of an accidental limited bandwidth or raised latency, the client will connect to the fastest available file server.

3.4.3 Conclusion

Where the differences were not that big in stage 2, they are in stage 3. The added latency has a big impact on the performance of Lustre. Coda could stand the latency better, because of its local cache. And was almost not influenced. Also in the case of failures is Coda better facilitated, because of its replicas.

4 Recommended SURFnet implementation design

This section describes a design on how Coda and Lustre can be used by SURFnet. This is a recommended design on how both file systems can be implemented on the network of SURFnet and what additional tools are required. This is based on the experience collected during the research, and the test results.

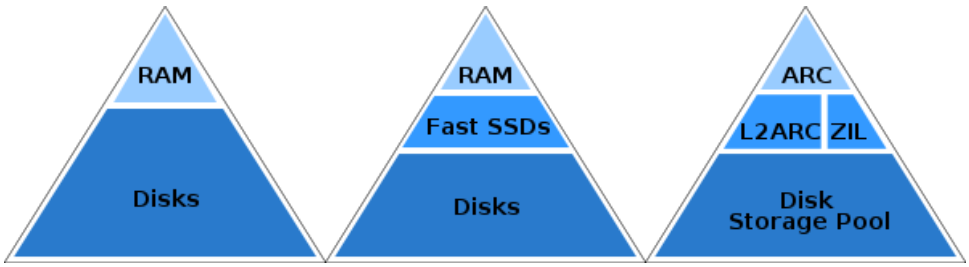
First some information about those additional tools will be given. After this first Coda will be discussed, second Lustre.

4.1 Additional tools

To enhance the architecture of both Coda and Lustre extra tools will be used. The tools and techniques will be explained in this section.

4.1.1 ZFS

Coda uses a cache to store data locally, this is done using the local file system. Here might be an opportunity to add an extra tier to the model. ZFS offers caching on the local file system. In this section two features of ZFS will be discussed: Second level ARC (L2ARC) and ZFS Intent Log (ZIL). L2ARC is an addition to the Adaptive Replacement Cache (ARC) and is used to cache file read operations. ZIL does speed up writes operations by caching those in DRAM instead of writing them direct to hard disks.



(a) The old model (b) The new model (c) Implementation in ZFS

Figure 11: ARC, L2ARC and ZIL in ZFS

ZFS makes use of ARC to cache files in DRAM. ARC sits between the system and the disks, so not all requests have to come from the slow disks. ARC is called primary cache. This because there is also a secondary cache available under the name L2ARC. L2ARC sits between the DRAM and the hard disks. This cache extents the DRAM cache. Often this cache is build with Solid State Drive (SSD) drives. Figure 11 shows the models in relation with ARC and L2ARC. [8]

ZIL is used by ZFS to speed up write operations to the hard disk. Normal some write operations are cached in memory, but important writes are direct written to the disk. Those writes drop the performance. To prevent this, ZIL makes it possible to also prevent those writes to the disk. To eliminate the risk

of data loss in case of a power or system failure, the system keeps a log of the system calls (the actual write operations). This log is saved on SSD drives, so it is saved on persistent storage and can be accessed fast. In case of a system failure, the system calls in the log can be replayed.[9][10]

4.1.2 Heartbeat

Heartbeat can be used to build a high available cluster. This by running two servers with each an own IP address, and one shared IP-address. One of the nodes is the master, and handles the requests. The second node is a slave, and is stand-by for the case the master fails. In case the master fails, the client can detect this in several ways. For example by scanning a specific service on the master, but also a dedicated RS232 communication link can be used. In case of a failure, the client will take over the shared IP address. Clients connecting to this IP address will not notice any change.

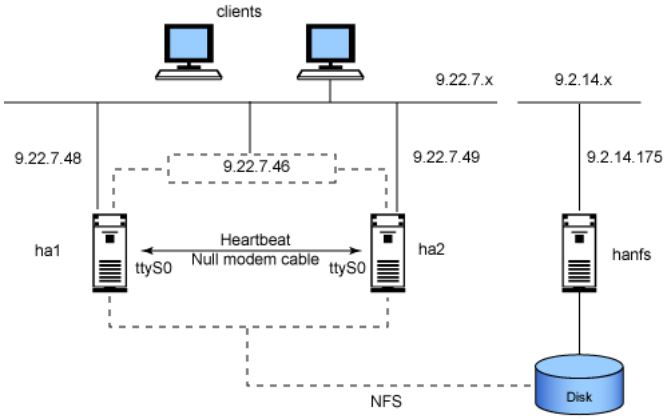


Figure 12: The components and their relations used by Heartbeat. [11]

4.1.3 DRBD

Distributed Replicated Block Device (DRBD) can be used to replicate a partition over a network. This is done by creating a RAID 1 like setup. DRBD is used on top of a normal device or partition. Two nodes need to be set up with a device or partition with the same size. On both nodes a virtual device is created. For the user data seems to be stored in the virtual drive. But when data is written to the virtual device, DRBD will write the data to the local device and also send the data over the network to the second node. The writing can be done synchronous and asynchronous. The difference is the moment DRBD returns a write successful message. With asynchronous this is done when the file is locally written. With synchronous this will be done after the file also has been written on the remote host.

4.2 Coda

The Coda proof of concept environment is based on Coda, ZFS and Heartbeat. Coda is used as the distributed file system, as described in section 2.3. Coda consists global of two kind of nodes, the file servers and the clients. The Coda file servers will run out-of-the-box without any special changes.

The Coda clients will be optimised by making use of ZFS and Heartbeat. ZFS is used on the Coda clients to speed up the caches. This by implementing the ZIL and L2ARC features. The cache is stored not in the default folder, but on a special mount, containing the ZFS file system. How ZIL and L2ARC work can be found in section 4.1.1.

Heartbeat is used to solve the Single Point of Failure (SPOF) caused by the client. The Coda client will be used by a whole system as access point to the distributed file system. If the Coda client fails, nobody behind the client can access the data any more. By creating a redundant Coda client, with heartbeat as fail-over mechanism, this SPOF can be eliminated. In this scenario the cache of the stand-by client will be empty, which will hit the read performance in the beginning, because all data has to be read from the servers. This empty cache will be called a cold cache. To keep the cache warm, it is possible to synchronise the cache of the active and the passive Coda client each minute using ZFS. The performance impact will be limited. This because of the used ZFS caching mechanisms which will cause most I/O being served from the DRAM. More information how Heartbeat works can be found in section 4.1.2.

In figure 13 a graphical presentation of the design of the Coda based distributed file system is presented.

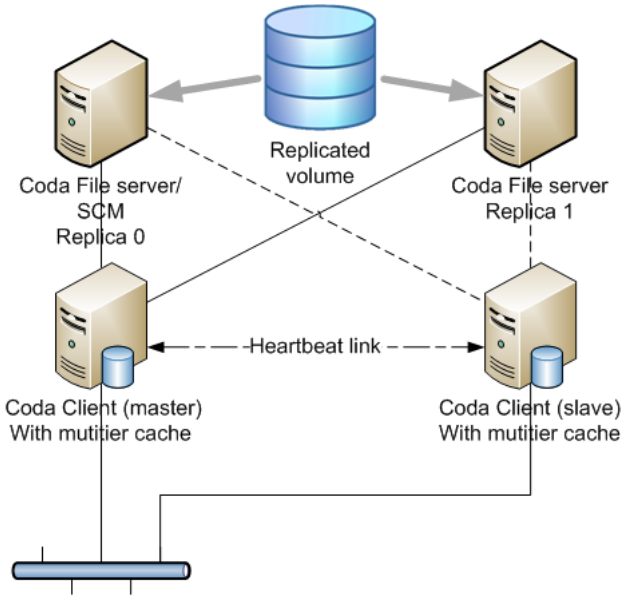


Figure 13: The Coda implementation design.

4.3 Lustre

The Lustre design is how Lustre could be used in the SURFnet environment. The design is shown in figure 14 and consists of the base components (as described in section 2.4.1) and some additional software to overcome some of the limitations of the Lustre architecture. The core of the system are the MDS, the OSS and the client.

The MDS is normally a single point of failure but this can be prevented by using Heartbeat (see section 4.1.2) to create an active/passive fail-over mechanism. Lustre has support for Heartbeat so this will not be a problem for implementing. The downside of using Heartbeat is the need for extra hardware. And this hardware does not add extra performance. Also the hardware is not being used until there is a failure. The positive side is that maintenance can be done on the main MDS server without taking down the whole Lustre system.

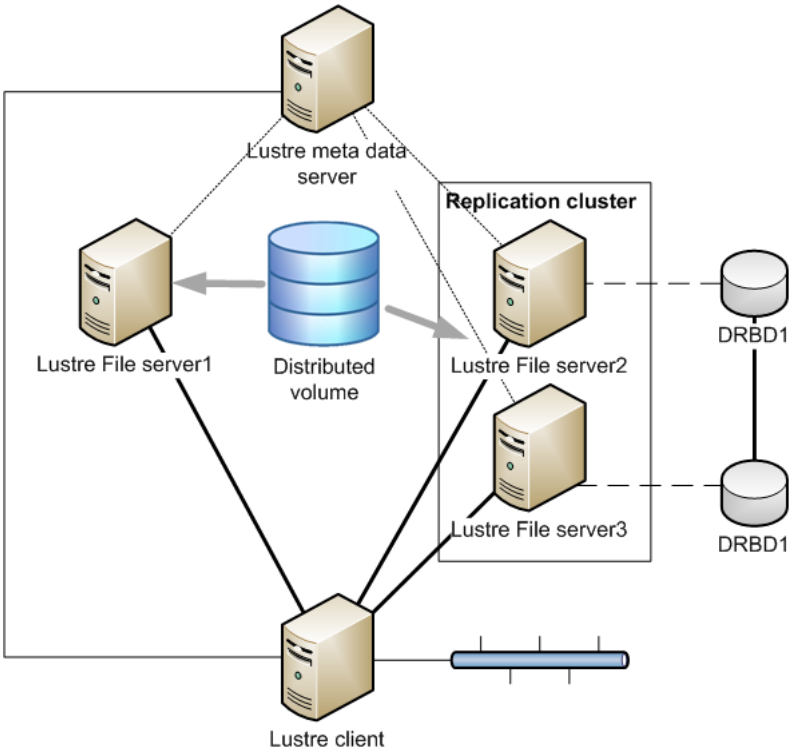


Figure 14: The Lustre implementation design.

Replication of the storage is a big issue in the Lustre environment since the system is only using striping and not any kind of replication. This can be a big problem if a server goes down since all files on that server can not be used anymore. Within Lustre it is possible to have multiple OSSs serving the same Object Storage Target (OST). This will prevents the system to be down, when one node fails. Although this does not solve the problem of an outage at a whole site because then all nodes serving the data are down. To add replication DRBD can be used to have a RAID1 like replication of the storage. Since Lustre

supports active/active fail-over, both sites can then serve the data. Although DRBD adds replication it will also create an overhead which will decreases the performance of the system. So adding the replication has a downside.

Looking back at the test results from stage 3 (section 3.4) the failure is taken away by using DRBD and fail-over. This is an improvement over the test setup. Also can DRBD probably take away (some) of the performance impact caused by the latency. Although this is not confirmed jet, some further research is required.

4.4 Combining the architectures

Coda has trouble with big files and striping. These both features are handled well within Lustre. Lustre on the other hand has trouble with replication and latency, those are well handled within the Coda architecture. This leads to the idea to use both architectures and benefit from the good sides and eliminate the weak sides of both. This idea is visualised in figure 15.

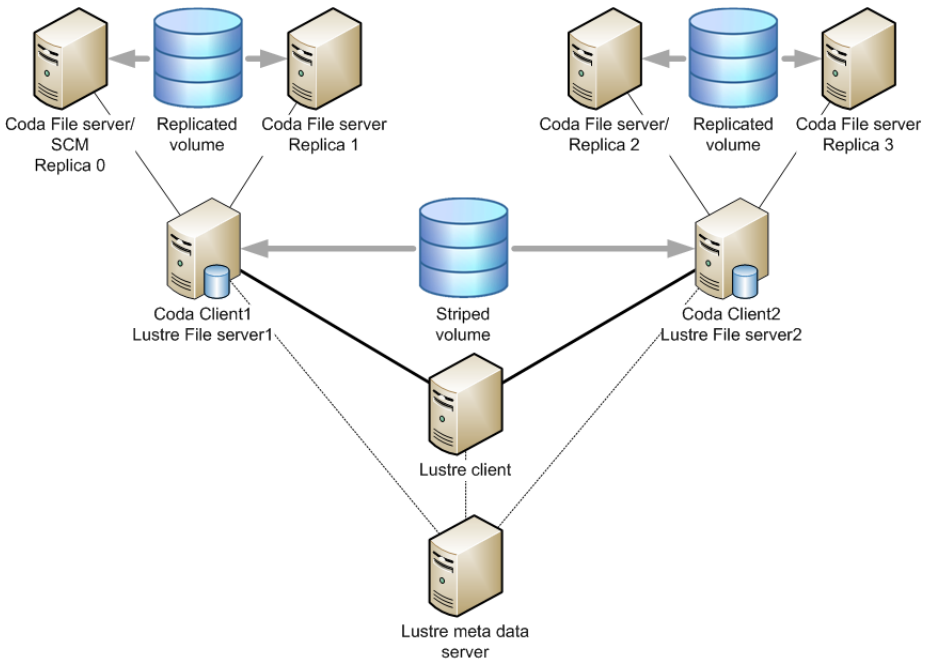


Figure 15: The combination of Coda and Lustre.

The Lustre architecture is used at the local site and can be used to mount the data. Lustre does not store the data on the local disk but rather on a Coda mount. Since Lustre uses chunks and striping to store the data on multiple sites the big file weakness of Coda is solved. The chunks are stored in the local cache of Coda, eliminating the latency that is a problem for Lustre. Once the data has been stored on in the Coda system it can be replicated at multiple servers. So now the striping is done by Lustre and the data is replicated by Coda.

In this setup there is a problem of the connection of Coda and Lustre. Lustre requires a block devices that be available on which it will place it own file system.

The coda client is a mount point which Lustre can not use. This problem needs to be solved with an external tool. Also using both architectures causes an increase of complexity which is not good for the maintenance of the whole system.

5 Conclusion

In this section the main and sub question will be answered. First the sub questions will be discussed, so they can be used to answer the main question.

- *Do the tools meet the requirements of SURFnet?*
  Both Coda and Lustre do not meet some of the requirements. Coda has trouble with updating large files. This causes the whole file to be copied to the server at every update of the file. Also is it hard to maintain. Lustre is missing replication, failure handling and can not handle the added latency well. Other requirements like cost effective, scalability and multi party access are met by both tools. The performance of Lustre is very depending on the latency. The performance of Coda is near to local storage, as long as the local cache can be used. When the network is involved, the performance drops as well, but not as dramatic as with Lustre. The shortcomings of both tools can be solved by using additional tools like: ZFS, DRBD, or heartbeat.

- *How intensive is the maintenance and deployment of the tools?*
  The installation of both Coda and Lustre can be done easy, also adding new nodes to the system is not a problem. The configuration and maintenance of Coda is a problem due the complexity of the system. Lustre has tools available which makes it easier to maintain. Also the documentation of Lustre is good and complete, what is not the case with Coda.

- *What conditions does the network infrastructure need to accomplice?*
  The most important condition is the reliability of the network, since both tools have trouble with connection failures. Coda is capable of handling the latency of a geographical dispersed network because of its read cache and write behind cache. Lustre has trouble with latency and needs a latency that is as low as possible (preferable around one msec). This because Lustre does not have a cache. This forces Lustre to communicate direct over the network. Also is the used protocol not optimised for latency

- *What is the performance of the tools on SURFnet's network infrastructure?*
  The impact of the SURFnet's network is mostly the added latency and the possibility of network failures. Coda does not have problems with the latency due the client cache. Luster on the other hand has trouble with the added latency, this is seen in the big decrease in performance. Network failure is something that is not handled well within both tools. Coda continues working but needs to replicate later after the connection has been re-established. This might require human interaction to solve conflicts. Lustre is not accessible any more, the copy is blocked until the server becomes available again.

- *What security features are implemented by the tools?*
  Both tools have a several security features implemented. Coda makes use of its own authentication mechanism and uses SSH to encrypt the communication channels. Access list can be used to control what user can access what. Lustre can use Kerberos for authentication and encryption. Lustre also uses access list to control the access to the files.

Now the sub questions are answered, the main question can be answered. Looking back at the main research question:

What infrastructure and open source tools provide SURFnet or the participants a scalable and distributed any-kind-storage solution?

At this moment the tools do not meet all the requirements that are needed to implement the wished distributed storage. The architectures have the potential but have trouble of managing the requirements on their own. This can partly be solved by using additional tools.

Coda has two main problems: the configuration and the handling of big files. The first problem can be solved by getting more experience with the system or when better documentation becomes available. The second problems is in the architecture of Coda, this can be hard to solve since it requires updating of the tool. The positive side of Coda is that it can handle the latency well and uses of multi tier storage architecture.

Lustre has a major problem with the latency since the performance of the system drops. Also the lack of replication is something that is really missing in Lustre. The replication can be performed outside of Lustre, by using an additional tool, but that will add extra overhead. Since Lustre already has trouble with latency this can be a problem. The positive side of Lustre is that it is using striping which cause a performance increase on sequential reading.

6 Discussion

The test environment used, was based on VMware ESXi. This added some overhead and made the test environment less predictable. This was visible in the network and the disk performance, and did have impact on the benchmarks. For further research in this subject it would be advised to look at the performance without the virtualisation layer in between.

Another good point for future testing, is to have a better look into the configuration and tuning capabilities of Coda. It can be more configured and tweaked as done during this project. Although the performance in the benchmarks was not bad, the throughput between the client and server was limited. This will be a problem if more data is offered as can be processed. By tuning the client and server communication speed will be increased.

A third point of future research is to extend Coda and Lustre with extra tools like ZFS and DRBD. Due to time limitations, this could not be done within this project but it can have a big impact on the results. For example what kind of impact does ZFS have on Coda or DRBD have on Lustre. Also the possibility to combine Coda and Lustre can be interesting for further research.

At last, an interesting point to look at, are other distributed file system projects. During the project two projects were chosen but also others show some potential. GlusterFS is the most important one, and has almost the same architecture as Lustre. It is interesting to test how GlusterFS handles the latency. Any differences in relation with Lustre, do not come from the architecture but from the implementation.

References

[1] Peter J. Braam. The coda distributed file system. URL http://www.coda.cs.cmu.edu/ljpaper/lj.html.

[2] A.S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms, 2/E*. Prentice Hall, feb 2006.

[3] Lustre replication rsync, June 2010. URL http://wiki.lustre.org/index.php/Lustre_2.0_Features#lustre_rsync.

[4] Lustre operations manual, June 2010. URL http://wiki.lustre.org/images/0/09/821-0035_v1.3.pdf.

[5] The lustre storage architecture.

[6] S. Carlier and D. Muller. Ssd performance. Technical report, University of Amsterdam, feb 2010.

[7] Surfnet service level specification, May 2009. URL http://www.surfnet.nl/Documents/SLS4.0-definitief.pdf.

[8] Zfs l2arc, jul 2008. URL http://blogs.sun.com/brendan/entry/test.

[9] Zfs on-disk specification, 2006. URL http://hub.opensolaris.org/bin/download/Community+Group+zfs/docs/ondiskformat0822.pdf.

[10] Zfs: The lumberjack, November 2005. URL http://blogs.sun.com/perrin/entry/the_lumberjack.

[11] High-availability middleware on linux, part 1: Heartbeat and apache web server, October 2004. URL http://www.ibm.com/developerworks/linux/library/l-halinux/.

[12] Coda file system, June 2010. URL http://www.coda.cs.cmu.edu/.

[13] Coda file system - wiki, June 2010. URL http://en.wikipedia.org/wiki/Coda_(file_system).

[14] Lustre, June 2010. URL http://wiki.lustre.org/index.php/Main_Page.

[15] Lustre - wiki, June 2010. URL http://en.wikipedia.org/wiki/Lustre_(file_system).

[16] Glusterfs, June 2010. URL http://www.gluster.org/.

[17] Xtreemos, June 2010. URL http://www.xtreemos.org/.

[18] Xtreemfs, June 2010. URL http://www.xtreemfs.org/.

[19] Xtreemfs - wiki, June 2010. URL http://en.wikipedia.org/wiki/XtreemFS.

[20] Ceph, June 2010. URL http://ceph.newdream.net/wiki/Main_Page.

[21] Ceph - wiki, June 2010. URL http://http://en.wikipedia.org/wiki/Ceph.

[22] Pvfs, June 2010. URL http://www.pvfs.org/.

[23] Pvfs - wiki, June 2010. URL http://en.wikipedia.org/wiki/Parallel_Virtual_File_System.

[24] Moosefs, June 2010. URL http://www.moosefs.org/.

[25] Moosefs - wiki, June 2010. URL http://en.wikipedia.org/wiki/Moose_File_System.

List of Acronyms

Access Control List (ACL)

An access control list, with respect to a computer file system, is a list of permissions attached to an object. 9, 11, 25, 39, 40, 42

Andrew File System (AFS)

The Andrew File System is a distributed networked file system which uses a set of trusted servers to present a homogeneous, location-transparent file name space to all the client workstations. 6, 41

Adaptive Replacement Cache (ARC)

The primary cache used by ZFS to cache file read operations. 26

Carnegie Mellon University (CMU)

Carnegie Mellon University is a global research university with more than 11,000 students, 84,000 alumni, and 4,000 faculty and staff. Recognised for its world-class arts and technology programs, collaboration across disciplines and innovative leadership in education, Carnegie Mellon is consistently a top-ranked university. 8, 11

Distributed Replicated Block Device (DRBD)

DRBD is a distributed storage system that can be used to get a RAID 1 like replication over the network. 27, 29, 30, 32, 34

European Union (EU)

The European Union is an economic and political union of 27 member states, located primarily in Europe. 44

File Transport Protocol (FTP)

File Transfer Protocol is a standard network protocol used to copy a file from one host to another over a TCP/IP-based network, such as the Internet. 11

Google File System (GFS)

File system used by google for all storage. 8, 48

General Public License (GPL)

The GNU General Public License is the most widely used free software license, originally written by Richard Stallman for the GNU project. 47

Graphic User Interface (GUI)

A Graphic User Interface is a type of user interface that allows people to interact with programs using graphical icons, and visual indicators. The actions are usually performed by direct manipulation of the graphical elements. 46

High Performance Computing (HPC)

High-performance computing uses supercomputers and computer clusters to solve advanced computation problems. 42

Institute of Electrical and Electronics Engineers (IEEE)

The Institute of Electrical and Electronics Engineers is an international non-profit, professional organisation for the advancement of technology related to electricity. 39

Internet Small Computer System Interface (iSCSI)

Internet Small Computer System Interface is an Internet Protocol (IP)-based storage networking standard for linking data storage facilities. 48

Lustre Networking (LNET)

LNET provides the communication infrastructure for Lustre. It can work with mulitple network types including Ethernet and InfiniBand. 13

Second level ARC (L2ARC)

The primary cache used by ZFS to cache file read operations. 26, 28

Logical Volume Manager (LVM)

Logical Volume Manager for the Linux kernel; it manages disk drives and similar mass-storage devices, in particular large ones. The term volume refers to a disk drive or partition. 13

Meta Data Server (MDS)

The MDS is the metadata server handling the requests. 12–14, 29

Meta Data Target (MDT)

The MDT is the storage nodes containing the meta data served by the MDS. 12

Network File System (NFS)

File system used by google for all storage. 48

National Research and Education Network (NREN)

A National Research and Education Network is a specialised internet service provider dedicated to supporting the needs of the research and education communities within a country. 5

Operating System (OS)

An operating system is the software on a computer that manages the way different programs use its hardware. 13, 44

Object Storage Server (OSS)

The Object Storage Servers handles the request for files or chunks that are requested by clients. 12–14, 29

Object Storage Target (OST)

The Object Storage Target contains the files or chunks that are handled by one or more OSS. 12, 29

Portable Operating System Interface (for Unix) (POSIX)

Portable Operating System Interface (for Unix) is the name of a family of related standards specified by the Institute of Electrical and Electronics Engineers (IEEE) to define the application programming interface, along with shell and utilities interfaces for software compatible with variants of the Unix operating system, although the standard can apply to any operating system 7, 42

Parallel Virtual File System (PVFS)

The Parallel Virtual File System is an Open Source parallel file system. This is a type of distributed file system that distributes file data across multiple servers and provides for concurrent access by multiple tasks of a parallel application. 46

Remote Direct Memory Access (RDMA)

RMDA is direct memory access form the memory between 2 hosts across the network. This is done without involving the OS which makes it more efficient to send files. 13

Remote Procedure Call (RPC)

A remote procedure call is an Inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. 14

Remote Procedure Call version 2 (RPC2)

A remote procedure call is an Inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. 11

System Control Machine (SCM)

The management server of the Coda environment. This server is used to create volumes, manage users and ACLs 9, 11, 16, 25

SSH File Transport Protocol (SFTP)

FTP implemented over a reliable and secure SSH version 2 connection. 11, 41

Single Point of Failure (SPOF)

A SPOF is a part of a system which, if it fails, will stop the entire system from working. 28, 39

Solid State Drive (SSD)

A solid-state drive is a data storage device that uses flash memory to store persistent data. 26

Secure Shell (SSH)

Secure Shell is a network protocol that allows data to be exchanged using a secure channel between two networked devices. 11, 32

Secure Socket Layer (SSL)

Secure Socket Layer is a cryptographic protocol that provide security for communications over networks. 44

Traffic Control (TC)

Traffic Control is a tool for Linux able to introduce more latency and limited bandwidth. This by making use of the leaky bucked model. 24

Wide Area Network (WAN)

A wide area network is a network that covers a broad area. 42, 44

Zettabyte File System (ZFS)

Zettabyte File System is a combined file system and logical volume manager designed by Sun Microsystems. The features of ZFS include support for high storage capacities, integration of the concepts of file system and volume management, snapshots and copy-on-write clones, continuous integrity checking and automatic repair, RAID-Z and native NFSv4 ACLs. 11, 26, 28, 32, 34

ZFS Intent Log (ZIL)

The ZFS intent log saves transaction records of system calls that change the file system in memory with enough information to be able to replay them. 26, 28

Appendices

A Tool selection

A.1 Coda

Coda is a distributed file system which was born in 1987 as a fork of AFS. AFS had one big disadvantage, when a file server was replicated, the replicates where read-only. This was solved in Coda. From Coda a new fork called InterMezzo was born. But these days InterMezzo does not exist any more.

Coda is a system which works with replicated file servers. A file server can be replicated over one or multiple replicates. A client is used to connect to the servers. The client owns a list (received from a so called master system) of the primary file server, and the replicates. The client makes use of caching on its local file system to speed things up. Advantages[12][13]:

- Product is developed for already a long time and is still active developed;
- Active community;
- Support for read/write replicates;
- Number of replicates can be configured upto 8;
- A replica can be added to a volume, even after the creation of the volume.
- Large number of file servers and clients can be used;
- **Caching capabilities at the client;**
- Clients are available for all well-known platforms: Windows, Mac OSX and Linux;
- **Beneath the system all possible file systems are possible;**
- Good documentation available;
- If master server fails the whole system continues running, except for adding or removing volumes, managing users and groups, or changing passwords.

Disadvantages[12][13]:

- All known limitations: `http://coda.wikidev.net/Limitations`;
- Files are processed multiple times, once at the client and at most two times at the file servers (depending on whether distribution is used).
- Complex product, easy to set up, hard to maintain and tune;
- The system makes use of SFTP for the file transport, which gives a performance hit on the CPU;
- **No redundant master (management) server.**

General information:

- Started in 1987 [12]
- Main page: `http://www.coda.cs.cmu.edu/`
- Wiki: `http://coda.wikidev.net/Main_Page`

A.2 Lustre

Lustre was first thought of in 1999 by Peter Braam. He was also working on the Coda fork InterMezzo. InterMezzo fixed in the past some performance issues. The assumption is that these performance issues in Coda are fixed in these days, so the advantage of InterMezzo disappeared. In 2002 he founded the company "Cluster file system" and started working on Lustre and dropped InterMezzo. In 2007 Sun Microsystems acquired the company and Lustre and later Oracle became the owner. Lustre was created to create a distributed file system using commodity hardware. Lustre is often used for High Performance Computing (HPC), it is used on 15 of the top 30 supercomputer.

Advantages[14][15]:

- Portable Operating System Interface (for Unix) (POSIX) compliance (atomic commit),
- ACL/ quotas,
- Adding data dynamic,
- Striping,
- TCP/IP supported,
- Failover of server role,
- **Fully developed.**

Disadvantages[14][15]:

- Not sure if it works on Wide Area Network (WAN);
- **Essence is on performance not on distributed storage;**
- Replication not handled in Lustre

General information:

- Oracle
- Started in 1999, first release in 2003 and bought by Sun at 2007. Active development.
- `http://wiki.lustre.org/index.php/Main_Page`

A.3 GlusterFS

GlusterFS is a parallel and / or replicated file system, initiated and supported by Gluster. At Gluster support can be bought. The development of GlusterFS has started in 2007. GlusterFS is very user friendly. Within a few minutes a master server is running. Storage nodes are deployed with the same ease. Although the easy deployment, the functionality is limited. Actually only JBOD, RAID1 or RAID10 configurations can be build. Flexible redundancy or redundancy over more nodes is not available. Adding more storage later on the storage nodes is unclear. When one of the cluster nodes fails, the storage is not available any more.

Advantages[16]:

- **Nice Gui, as well web based as on the console on the master server;**
- Can create normal (JBOD), stripe or mirrored volumes;
- Support for CIFS, NFS and native by making use of the GlusterFS client;
- Multiple file systems can be deployed on the virtual distributed file system (like ext3, ext4, ZFS)
- Support for Ethernet and Infiniband;
- **Very easy deployment of new cluster nodes;**
- **Very easy central update management using the central management console;**
- **Professional (paid) support available;**
- All nodes can be used to reach the data

Disadvantages[16]:

- Management runs only on one server, no redundancy;
- **New nodes can not be added on the fly;**
- Not clear how to add multiple disks per node.
- Not developed for performance
- **Security is not done in GlusterFS**

General information:

- Developed by Gluster
- Started in 2007
- `http://www.gluster.org/`

A.4 XtreemFS

XtreemFS is part of XtreemOS. In 2006 it got an funding from the European Union (EU). The XtreemOS OS aims at integrating as a single computing platform many different kinds of devices, from mobile ones to large clusters[17]. The project is fully in development so the project is not very usable for production.

Advantages[18][19]:

- Designed for Wide Area Network (WAN) grid;
- Authentication;
- **Secure Socket Layer (SSL) encryption;**
- Striping, checksums;
- Policies for authentication and striping;
- Basic website for status;
- **Caching for high latency links;**
- Linux file right are remembered (user / group ID).

Disadvantages[18][19]:

- **Is new and not fully developed;**
- **Only read only replication possible at the moment;**
- Management runs on 1 server;
- Client needed for mounting;

General information:

- Part of EU project
- Started in 2006 - last release may,2010
- `http://www.Xtreemfs.org`

A.5 Ceph

Ceph is a distributed file system designed for reliability, scalability, and performance. Just like the other systems discussed in this report. Ceph is well featured, but still under heavy development. This makes it at this moment inappropriate for SURFnet.

Advantages[20][21]:

- Snapshot functionality (on directory level);
- **Support for redundant meta data servers;**
- Support for redundant management (monitors) servers;
- **Multiple pools can be defined for replication. By example racks, or physical separated locations.**

Disadvantages[20][21]:

- **Under heavy development, not yet usable for production purpose;**
- Storage nodes (called bricks) have to make use of Btrfs (ext3 is also possible, but not advised);
- Their wiki page is not completely filled yet.

General information:

- Ceph community
- Started at 2007
- `http://ceph.newdream.net/wiki/Main_Page`

A.6 PVFS

Parallel Virtual File System (PVFS) is a file system that is used for being parallel. So distribution is supported but replication not. Also the lack of authentication and the hard possibility for adding and removing server is not feasible for the intended usage.

Advantages[22][23]:

- Can use more network interfaces simultaneously;
- Storage is assigned to a directory;
- Distributed.

Disadvantages[22][23]:

- Special library or kernel module needed for client;
- No Graphic User Interface (GUI)configuration;
- Authentication not seen;
- **No replication in data and metadata;**
- **Changes in the system (adding server) require the system to be fully down.**

General information:

- Research community - Universities / laboratories / Commercial
- Started at least before 2003 - last release in February 2010
- `http://www.pvfs.org/`

A.7 MooseFS

MooseFS is a relative young project which was released in 2008 by a research company. It has some good advantages and the new features are promising but it is also not very developed yet.

Advantages[24][25]:

- Replication factor (N-1 still available);
- Security;
- **On the fly changes possible (mark for removal);**
- **Snapshot possible.**

Disadvantages[24][25]:

- **Non redundant master server;**
- Not fully developed yet;
- Chunk size is hard coded;
- File size limit of 2TiB (possibly extended in new release to 16 EiB);
- Network should be build on 1GB Ethernet.

General information:

- Developed by `http://gemius.com/`
- General Public License (GPL)in 2008 - Last release in April 2010
- Main page: `http://www.moosefs.org`

A.8 Hadoop

Hadoop is a cloud storage framework based on GFS. GFS is a nice example of distributed storage, although not feasible for this project. Google owns one of the largest distributed file systems. Despite GFS is not available for public. That is why open source communities created their own. Hadoop has nice features that would meet the requirements but it does only can be used by the API. The framework can be used by application for storage in the cloud. An other example is CloudStore.

Because of the lack of an Internet Small Computer System Interface (iSCSI), Network File System (NFS), or other interface, those systems are not suitable for file storage and contain only interfaces at application level. They are not build with file storage in mind.

B Raw results - First stage - Disk IO

		Put character (KB/s)	**Put block** (KB/s)	**Get character** (KB/s)	**Get block** (KB/s)	**Seeks** (seeks/sec)
esx1	**sda**	18621	31430	22533	34271	254
	sdb	19208	44066	23928	46865	623
	sdc	18913	39397	24066	41968	591
esx2	**sda**	19081	42480	23284	46314	591
	sdb	18577	52300	23182	59225	615
	sdc	18934	51867	23730	53291	604
esx3	**sda**	17975	37605	21870	51672	633
	sdb	18899	49595	23300	52744	632
	sdc	19077	47353	23395	53362	435
esx4	**sda**	33132	40622	29424	47139	535

Table 4: Raw disk null test results from Bonnie++.

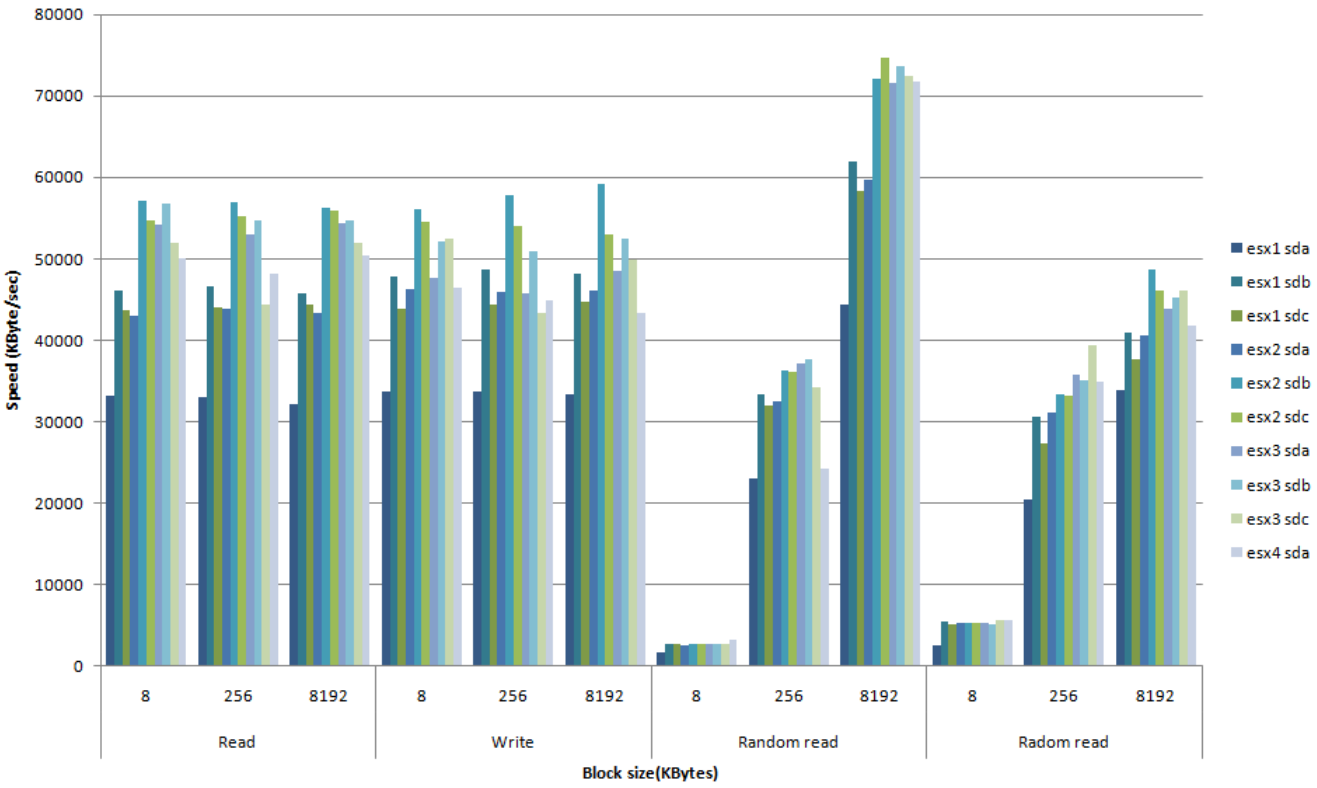


Figure 16: Detailed overview of the average performance for each data store.

		Read (KB/s)			**Write** (KB/s)		
		8	**256**	**8192**	**8**	**256**	**8192**
esx1	**sda**	**33183**	**33024**	**32180**	**33727**	**33661**	**33437**
		33893	33906	33497	34117	32540	33231
		33159	32953	31384	33776	34118	33335
		32497	32213	31659	33289	34324	33746
	sdb	**46066**	**46603**	**45775**	**47866**	**48721**	**48097**
		47133	47007	46088	47319	49099	47350
		44312	46480	43929	47614	48492	48729
		46752	46323	47308	48664	48573	48212
	sdc	**43677**	**43986**	**44299**	**43876**	**44442**	**44752**
		43564	43943	44662	44091	44756	44057
		43801	44576	44089	42490	44511	45029
		43666	43440	44147	45048	44058	45171
esx2	**sda**	**43040**	**43920**	**43378**	**46234**	**45881**	**46117**
		48120	49194	48732	51427	50749	50622
		39185	41307	42200	43475	43879	44355
		41815	41260	39201	43799	43015	43375
	sdb	**57140**	**57008**	**56301**	**56173**	**57796**	**59165**
		57582	57583	57741	58911	58158	58200
		56633	56565	54051	54544	55151	59210
		57206	56875	57111	55064	60079	60086
	sdc	**54776**	**55174**	**55858**	**54587**	**54053**	**53024**
		53898	55333	55838	54713	53671	53629
		55157	55278	55749	54576	53825	53774
		55273	54912	55986	54473	54663	51669
esx3	**sda**	**54166**	**53038**	**54290**	**47627**	**45731**	**48459**
		53994	50198	54358	48944	48476	47887
		54484	54371	54026	45405	40595	48460
		54019	54544	54485	48532	48123	49029
	sdb	**56761**	**54694**	**54712**	**52042**	**50920**	**52499**
		56811	51062	56569	53015	49065	50353
		56347	56280	50883	53047	51013	53089
		57125	56739	56685	50064	52683	54054
	sdc	**51983**	**44322**	**51940**	**52448**	**43371**	**49970**
		49105	26297	53614	51480	26336	45828
		53392	53335	53588	53048	52700	52117
		53453	53334	48618	52817	51077	51965
esx4	**sda**	**50140**	**48161**	**50454**	**46441**	**44891**	**43338**
		49863	50469	51137	49148	45252	38021
		49877	45981	49851	47532	42355	44445
		50681	48034	50373	42644	47066	47547

Table 5: Raw test disk null test results from Iozone - Part1. Bold results are average values for that data store.

		Random read (KB/s)			**Radom write** (KB/s)		
		8	**256**	**8192**	**8**	**256**	**8192**
esx1	**sda**	**1628**	**23055**	**44461**	**2499**	**20385**	**33909**
		1576	22102	44068	2409	20218	33106
		1631	23100	42783	2526	20200	35209
		1677	23963	46533	2562	20736	33412
	sdb	**2736**	**33289**	**61995**	**5543**	**30661**	**40856**
		2662	32034	57772	5979	30829	40415
		2775	33767	62857	5257	31208	42252
		2771	34067	65355	5394	29945	39901
	sdc	**2714**	**31971**	**58345**	**5082**	**27340**	**37654**
		2634	30851	56804	5037	26973	38461
		2746	32459	58255	5050	27621	36828
		2763	32602	59975	5158	27427	37673
esx2	**sda**	**2611**	**32546**	**59723**	**5287**	**31068**	**40541**
		2477	32711	63320	5573	32608	45190
		2671	31818	57369	4995	30174	37868
		2685	33108	58480	5293	30423	38566
	sdb	**2717**	**36326**	**72050**	**5322**	**33375**	**48726**
		2521	35369	61714	5036	32825	49202
		2817	36444	77835	5257	34089	51090
		2812	37165	76601	5674	33210	45886
	sdc	**2746**	**36113**	**74726**	**5380**	**33225**	**46164**
		2609	33724	71114	5046	32851	45705
		2812	37204	76166	5522	33005	46150
		2816	37410	76899	5572	33820	46636
esx3	**sda**	**2713**	**37079**	**71532**	**5317**	**35694**	**43833**
		2672	37468	71851	5404	36050	46083
		2722	37690	65746	5162	36004	43019
		2746	36080	76999	5386	35029	42396
	sdb	**2726**	**37716**	**73621**	**5078**	**35166**	**45248**
		2683	35481	65141	4912	33973	46246
		2751	38690	75625	5107	36677	44221
		2745	38977	80097	5216	34848	45278
	sdc	**2796**	**34306**	**72460**	**5705**	**39302**	**46068**
		2664	30457	69441	5479	37727	45437
		2858	37323	73534	5786	40962	46898
		2865	35139	74406	5851	39216	45870
esx4	**sda**	**3271**	**24277**	**71737**	**5718**	**34830**	**41734**
		3186	24202	74690	5978	34456	40885
		3387	24155	70289	5860	34618	43556
		3240	24473	70232	5315	35416	40762

Table 6: Raw test disk null test results from Iozone - Part2. Bold results are average values for that data store.

C Raw results - First stage - Network IO

Connection	test1	test2	test3	Averages
1 to 2	76.7	58.5	60.5	65.2
1 to 3	73.9	62.4	64.5	66.9
1 to 4	87.0	79.4	81.2	82.5
2 to 1	70.6	61.4	65.5	65.8
2 to 3	73.8	64.7	65.1	67.9
2 to 4	89.2	85.3	85.1	86.5
3 to 1	75.2	61.8	64.0	67.0
3 to 2	67.1	68.6	66.7	67.5
3 to 4	88.5	80.1	77.7	82.1
4 to 1	67.8	62.3	65.3	65.1
4 to 2	77.8	70.6	68.3	72.2
4 to 3	75.0	64.0	65.1	68.0

Table 7: Raw network null test results using Iperf on a clear network (in MBytes/sec).

Connection	min	avg	max	mdev
1 to 2	0.294	0.424	0.961	0.074
1 to 3	0.249	0.386	2,369	0.247
1 to 4	0.291	0.395	1,074	0.108
2 to 1	0.312	0.642	13,464	1,499
2 to 3	0.306	0.438	3,673	0.389
2 to 4	0.298	0.512	1,888	0.168
3 to 1	0.252	0.39	1,393	0.116
3 to 2	0.298	0.727	28,743	2,826
3 to 4	0.296	0.479	2,021	0.181
4 to 1	0.27	0.462	1,604	0.15
4 to 2	0.332	0.501	0.954	0.097
4 to 3	0.218	0.483	4,460	0.463

Table 8: Raw network null test results using ping with 100 runs on a clear network (in msec).

D Raw results - Second stage - File system benchmarks

D.1 Bonnie++

		Test	Put character (KB/s)	Put block (KB/s)	Get character (KB/s)	Get block (KB/s)	Seeks (seeks/sec)
Lustre	**run1**	**1**	27000	8257	27353	42552	155.4
		2	25542	8671	27641	64929	153.3
		3	17902	8762	28606	71931	153.4
	run2	**1**	27080	8755	27795	48378	154.4
		2	25310	8678	27096	68098	151.5
		3	24792	8540	27631	48732	151.4
Coda	**run1**	**1**	3454	41827	3604	3804	27.6
		2	3621	43300	31137	3738	28.4
	run2	**1**	3759	40548	3761	3921	28.4
		2	3777	3878	3758	3922	28.3

Table 9: Bonnie++ benchmark on Lustre and coda, raw results.

		Put character (KB/s)	Put block (KB/s)	Get character (KB/s)	Get block (KB/s)
Lustre	**min**	17902	8257	27096	42552
	avg	24604	8610	27687	57437
	max	27080	8762	28606	71931
Coda	**min**	3454	3878	3604	3738
	avg	3653	32388	10565	3846
	max	3777	43300	31137	3922
Null test	**min**	17975	31430	21870	34271
	avg	20242	43671	23871	48685
	max	33132	52300	29424	59225

Table 10: Bonnie++ benchmark on Lustre and coda, summarised results.

	min	avg	max
Lustre	151	153	155
Coda	27.6	28.175	28.4
Null test	254	551	633

Table 11: Bonnie++ disk latency test. The amount of seeks in one second for the null test, Coda and Lustre (seeks/sec).

D.2 Iozone

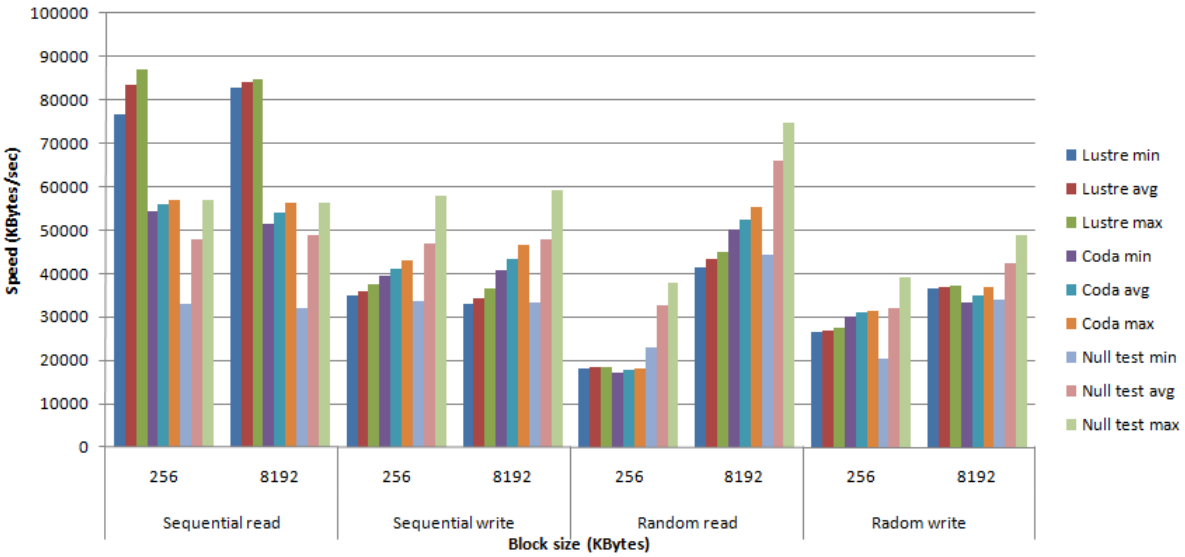


Figure 17: Iozone benchmark of Coda, Lustre and for comparison, the null test in detail.

		Read (KB/s)		**Write** (KB/s)		**Random read** (KB/s)		**Radom write** (KB/s)	
		256	**8192**	**256**	**8192**	**256**	**8192**	**256**	**8192**
Lustre	**Run1**	76808	84672	35540	33211	18139	41508	26430	37195
	Run2	86460	82986	37383	36595	18325	43870	27476	37311
	Run3	87089	84841	34998	32851	18516	45130	26906	36647
	Min	76808	82986	34998	32851	18139	41508	26430	36647
	Avg	83452	84166	35974	34219	18327	43503	26937	37051
	Max	87089	84841	37383	36595	18516	45130	27476	37311
Coda	**Run1**	54239	51418	41219	40712	17059	50188	30083	34308
	Run2	56927	54289	39377	42557	17993	55418	31242	33331
	Run3	56966	56386	42966	46713	18261	51609	31473	36946
	Min	54239	51418	39377	40712	17059	50188	30083	33331
	Avg	56044	54031	41187	43327	17771	52405	30933	34862
	Max	56966	56386	42966	46713	18261	55418	31473	36946

Table 12: Iozone benchmark on Lustre and Coda, raw results.

E Raw results - Third stage - File system benchmarks

E.1 Coda

	Put character (KB/s)	**Put block** (KB/s)	**Get character** (KB/s)	**Get block** (KB/s)	**Seeks** (seeks/sec)
Run1	1285	38058	1280	1299	9.9
Run2	1266	39696	34219	54822	9.5
Run3	1274	1286	1270	1292	10
Run1	1284	1288	1273	1295	9.8
Run2	1281	1300	1277	1304	9.8
Run3	1274	39946	1294	1308	9.9

Table 13: Bonnie++ benchmark on Coda with 20ms (RTT) latency, raw results.

	Read (KB/s)		**Write** (KB/s)		**Random read** (KB/s)		**Random write** (KB/s)	
	256	**8192**	**256**	**8192**	**256**	**8192**	**256**	**8192**
Run1	51717	53354	40412	43246	16224	53881	28801	33932
Run2	53029	54140	40640	44001	16360	53042	29174	35334
Run3	52293	52006	41913	41905	16317	55382	29340	33392

Table 14: Iozone benchmark on Coda with 20ms (RTT) latency, raw results.

E.2 Lustre

	Put character (KB/s)	**Put block** (KB/s)	**Get character** (KB/s)	**Get block** (KB/s)	**Seeks** (seeks/sec)
Run1	16655	9119	15213	4437	28.7
Run2	14304	9086	17513	17418	29.2
Run3	14069	9039	16701	8281	29.1

Table 15: Bonnie++ benchmark on Lustre with 20ms (RTT) latency, raw results.

	Read (KB/s)		**Write** (KB/s)		**Random read** (KB/s)		**Random write** (KB/s)	
	256	**8192**	**256**	**8192**	**256**	**8192**	**256**	**8192**
Run1	15594	18811	14691	16501	2721	9927	13644	18125
Run2	22363	22666	3731	15259	2709	10110	13396	15953
Run3	20121	20269	16053	16692	2713	10016	13335	17752

Table 16: Iozone benchmark on Lustre with 20ms (RTT) latency, raw results.