UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

SECURITY FEATURES USING A DISTRIBUTED FILE SYSTEM

Rui Miguel Coelho Martins

MESTRADO EM SEGURANÇA INFORMÁTICA

Dezembro 2011

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

SECURITY FEATURES USING A DISTRIBUTED FILE SYSTEM

Rui Miguel Coelho Martins

Orientador

Greg Ganger

Alysson Bessani

MESTRADO EM SEGURANÇA INFORMÁTICA Dezembro 2011

Resumo

Informação sensível como por exemplo dados provenientes the firewalls ou sistemas de detecção de intrusões, é preciso que seja armazenada durante longos períodos de tempo por razões legais ou para fins de análise forense. Com o crescimento das fontes geradores deste tipo de dados dentro de uma empresa, torna-se imperioso encontrar uma solução que cumpra os requisitos de escalabilidade, segurança, disponibilidade, performance e baixa manutenção com custos controlados. Na sequência desta necessidade, este projecto visa fazer uma análise sobre vários sistemas de ficheiros distribuídos por forma a encontrar uma solução que responda aos requisitos de performance e segurança de uma aplicação interna da Portugal Telecom. Para validar a solução, o projecto inclui a concepção de um protótipo que pretende simular as condições de execução dessa aplicação.

Palavras-chave: sistema de ficheiros, distribuído, performance, segurança

Abstract

Sensitive information such as firewall logs or data from intrusion detection systems, has to be stored for long periods of time for legal reasons or for later forensic analysis. With the growth of the sources generating this type of data within a company, it is imperative to find a solution that meets the requirements of scalability, security, availability, performance and low maintenance while keeping the costs under control. Following this need, this project aims to make an analysis of several distributed file systems in order to find a solution that meets both the performance and security requirements of an internal application of Portugal Telecom. To validate the solution, the project includes the design of a prototype that aims to simulate the execution environment of that application.

Keywords: distributed, file system, performance, security

Acknowledgments

I am thankful for both my advisor Greg Ganger and my co-advisor at FCUL, Alysson Bessani, for the valuable insight for this work. I also appreciate the folks at the PDL for making me feel welcome at CMU.

Lisboa, December 2011

Dedicated to my mother for supporting me 100% no matter what and always being selfless.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Performing the monitorization of a computer network is essential to ensure a smooth operation of the services it supports, but the traffic that goes through it and the equipments that are utilized generate a large amount of control data that is useful to comprehend the status of the systems. Moreover, the deployment of complex security controls that are becoming widespread leads to the generation of a very large amount of events that must be safeguarded for compliance reasons, legal reasons, or even for the purpose of its analysis for evidence of attacks or security flaws.

In order to ensure the value of these events, they have to be processed and correlated as to trigger automatic mechanisms or to produce high level information that can be easily interpreted by the infrastructure operators. Furthermore, the collected events many times have to be stored for a variable period of time depending on what the information is and how it will be used.

As the network grows in size and speed as well as the number of applications that run on that infrastructure, the amount of data that has to be stored increases - most of the times without the operators knowing how much the data size will grow. This poses a challenge in terms of storage that calls for a solution that can be flexible enough so that it can scale with the changes in the environment and, at the same time, do not involve a high monetary cost. Today's storage systems using powerful computing platforms present huge costs for any organization and ISP and, thus, alternative approaches are preferable. In this context, distributed file systems are a fitting solution for this storage requisite. There are, however, many available distributed file systems that differ in several aspects such as design goals, implemented features, technical details, and performance capabilities.

1.1 Motivation

Portugal Telecom [1] - a portuguese telecommunications company - has a security information and event management (SIEM) platform currently in production that processes and stores around 30GB of log data every day. In the file system the logs are separated in two files that grow at a constant rate throughout a 24 hour period - see Figure 1.1.

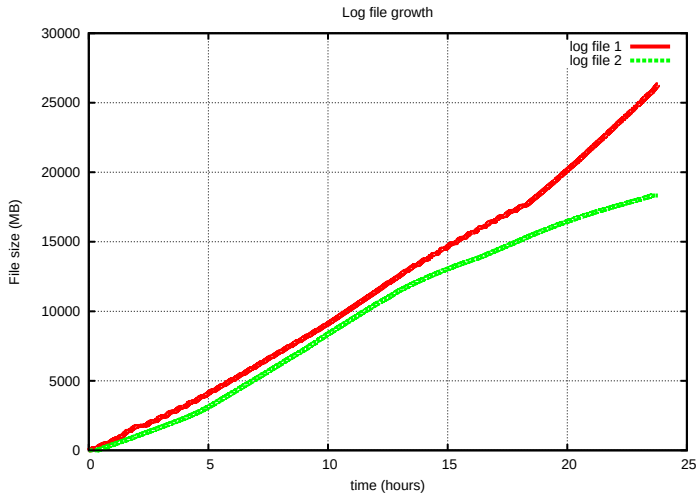

Figure 1.1: Log file growth

After compression and archiving the logs are placed together in a single file of about 5GB. For the reasons mentioned before Portugal Telecom is required to hold the collected data for a long period of time. Therefore, it seeks a long term storage solution that is flexible enough to accommodate the current storage demand without compromising in robustness and availibility. At the same time, it is necessary that the solution offers scalability properties that permit the growth of the storage capacity, seeing that the amount of daily collected events tends to increase as more sources are added to the SIEM. Additionally the cost of the storage is a relevant metric that must not be ignored. Hence, being able to use commodity hardware becomes a essential and, for that reason, distributed storage presents the required features.

Several distributed storage solutions are available but they have different characteristics due to their fundamental goals. In order to evaluate the feasibility of using those systems as a way to meet the requirements of the SIEM platform it is imperative to study and test the options.

[1] `http://www.telecom.pt`

1.2 Objectives and contributions

This project comprises two parts. In the first part I research some open source distributed file systems, analyzing the architectures and design goals behind each one. The second part comprises some benchmark tests with one of the file systems, observing its performance under different setups. The main purpose of the tests is to see if the file system can be used to support an existing application of Portugal Telecom. That application is responsible for storing security data that is collected from several network equipments and business support applications, and because of that it requires a certain degree of security.
The major objectives of this thesis are:

- Find out if there are open source distributed file systems that have the properties that fit the necessities of Portugal Telecom's SIEM application, in terms of storage capacity, performance, security, and scalability;
- Test the file systems as to validate the aspects and features advertised in the available publications about the file systems;
- Find possible shortcomings that might steer away from a solution using these types of solutions.

A research on some available file systems is the starting point for the experimental work which enables a quantitative view on the application of the parallel file system. The experiment was done using some popular file system benchmark tools and consisted in several use cases that are related with the objective of this work. The contributions of this thesis are:

- Research and digest the available documentation on some relevant distributed file systems and describe their design strategies, architectures, and particularities. For that matter, a survey on some open source options is produced so that it is possible to have a quick overview of each one's main features and current status;
- The application of one of the surveyed file systems using a prototype to emulate Portugal Telecom's use case, adding an additional security layer to the system, for the solution does not provide important security features;
- Attest that the usage of a distributed file system is suitable for Portugal Telecom's application with a real data sample.

1.3 Organization

The remainder of this document is organized as follows. Chapter 2 I look into six distributed file systems, describing their architecture and design goals; next in Chapter 3 which is divided in four sections I characterize the application that we are trying to accomodate in a distributed file system and present the results of the benchmark tests I ran. Moreover, I report the design and the outcome of the prototype I setup in order to verify if the file system meets its requirements. Finally Chapter 4 discusses the conclusions of this project.

Chapter 2

A survey on distributed file systems

Several distributed storage options are currently available for free download, and kwnowing their characteristics is essential to find each ones are suited for a particular use case. Among the available distributed file systems different design strategies are employed - some use a single metadata server, others offer the possibility of replicating it whereas some do not even use a metadata node. In addition, some of them choose to follow POSIX [1] semantics while others may assume that by doing this it becomes an hindrance to their goals.
In this chapter I present some of the most relevant distributed file systems that follow an open source model. In particular the analyzed options are GlusterFS, HDFS, Lustre, Sector, and XtreemFS. The analysis is mostly based on the documentation available on the Web, with particular incidence on the published papers and presentations related to each of the file systems. According with the survey a decision between the suitable options is presented on Chapter 3 where I design one possible solution to the challenge imposed by Portugal Telecom's SIEM needs.

2.1 Gluster

GlusterFS is an open source distributed file system that allows the combination of a large number of commodity storage and compute resources into a high performance and centrally managed pool. It advocates three main goals: elasticity, linear scaling, and scale out [2]. Elasticity in this context refers to the ability to adapt to the growth or reduction of data and add or remove resources to a storage system as needed, without disrupting the service. Linear scaling is the property of a system

whose capacity or throughput increases or decreases as more resources are added to it in linear fashion, i.e., adding twice the amount of storage should deliver twice the observed performance: twice the throughput without a significant change in the average response time. This means that it is not sufficient to be able to increase the total storage size. It is required that the system is able to cope with this growth - larger metadata space, the file system must be able to support the total size, the network must meet the increased number of clients accessing the disks, and so forth. The third main goal of GlusterFS is to be a solution which is able to scale out for both performance and capacity, that is, providing horizontal scalability properties. So, an enterprise can increase the performance of system by deploying more nodes and add more capacity by adding more disks, without having to manage a more complex system.

GlusterFS uses a simple strategy in the way it stores the files. Each file is written on disk in the native format of the underlying file system (e.g. EXT3, EXT4, XFS). This means that files can be read directly from the native file system without the need to use GlusterFS, which is useful if, for some reason, the data needs to be moved out of Gluster. But the cornerstone of GlusterFS is the absence of a metadata node. As the number of files, disks or storage nodes increases, the effort to keep track of of the logical location of data (metadata) grows in such a way that it may become a performance bottleneck. Furthermore, many scale-out systems create a separate index with file names and respective locations that pose a potential single point-of-failure. GlusterFS, on the other hand, does not make use of a separate index of metadata. Instead, the placement and location of files is computed using a deterministic algorithm that is called Elastic Hashing Algorithm (Figure 2.1. This means that each storage server has the intelligence to locate any data without having to perform lookup on an index or query a remote server. Each storage server only needs to apply the algorithm on the filename and pathname. This method results in an performance boost because the metadata operations that otherwise had to be concentrated in the metadata server, can be parallelized, thus ensuring linear performance scaling.

In order to overcome the said limitations of a single metadata node, a distributed approach could be used so that the location of the files is spread among several storage systems. But this approach still suffers from some performance and availability issues. The synchronization mechanisms needed keep the metadata nodes coherent accrue some processing overhead that tends to become worse with the increase of the number of indexed files as well as file operations. Another related problem is the danger of a corrupted metadata node. If the corruption is detected and appropriate procedures are executed to recover the bad node, then the systems is able to continue its

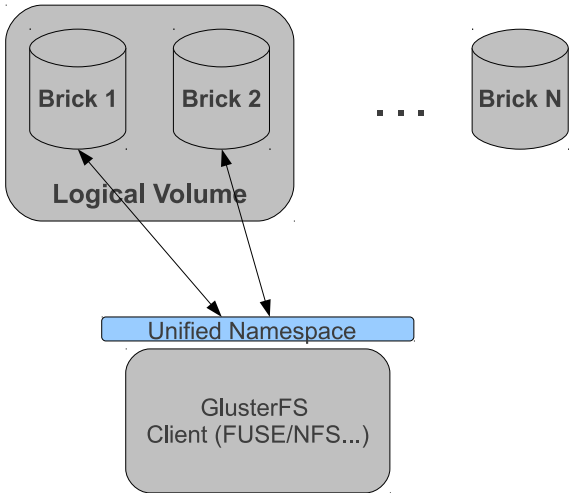

Figure 2.1: GlusterFS does not make use of metadata nodes. The client uses the Elastic Algorithm to locate files in the storage nodes.

regular execution. But, if the node becomes corrupted in such way that the corruption goes undetected then it is possible to reach a scenario where distinct metadata nodes yield different results for the same query, which in practice means the file system is completely broken and unusable.

2.1.1 Elastic Hashing Algorithm

Gluster's Elastic Hashing Algorithm is based on Davies-Meyer hashing algorithm [3] which is construction used on one-way compression functions. The properties of a cryptographic hash function, namely the transformation of a variable-sized input into a fixed length output with a near zero probability of collisions, suit the purpose of the model used by GlusterFS to deal with the location of the files in the server nodes. Since the pair pathname+filename is unique across the entire file system, the Elastic Hashing Algorithm enables any node to independently compute the location of a file by simply applying a mathematical function on that pair. As a result of this approach, there is no contention when multiple entities try to access the same metadata, therefore the performance is faster comparing to the metadata node scenario and also there is no single point-of-failure. Moreover, the risk of out-of-sync metadata is eliminated by the Elastic Hashing Algorithm.

The dynamic nature of a distributed file system - disks are added or replaced, servers are added and/or changed, files need to redistributed, etc. - has to be dealt with. Gluster handles this situations by abstracting the file system with a large number virtual volumes, so that the hashing algorithm assigns files to virtual volumes. Additionally, a separate process assigns a virtual volume to one or more physical devices. As a consequence, the hashing algorithm does not need to be

changed even though the underlying physical storage is prone to modifications. If a file is renamed, it will possibly be assigned to a different virtual volume - due to a new result of the hash algorithm - which in turn can be assigned to a different physical volume. In the time between the file rename operation and the migration of the file is completed, Gluster puts a pointer in the new location indicating the old volume so that new file accesses are forwarded are able to retrieve the file. A background process then deletes the pointer when the file is actually moved. The same strategy is also used if a file needs to be moved due to a disk becoming "hot" or degraded in performance for instance.

2.2 HDFS

Apache Hadoop [1] is a framework devised to support distributed applications that perform computations on large data sets using the MapReduce paradigm. It relies on data partitioning and computations in a large number of hosts so as to take advantage of parallel operations. The Hadoop Distributed File System (HDFS) is the file system component of Hadoop and it is an open source implementation of the Google File System [4]. It stores the file system metadata separated from the application data. The metadata is stored in a single dedicated server called the NameNode, whereas the application data is placed on different servers that are called DataNodes. All the nodes communicate directly by means of TCP-based protocols. Figure 2.2 shows a standard HDFS architecture. [2]

2.2.1 NameNode

The HDFS NameNode holds the namespace tree which is a hierarchy of files and directories that are represented by inodes that record the metadata - things like file permissions, modification and access times, and disk space usage. Moreover, the NameNode mediates client accesses to files. Each file content is split into large blocks - typically 64 or 128MB - and each block is replicated at multiple DataNodes. The mapping of the file to its blocks is maintained in the namespace tree. The size of the block can be modified by the user file-by-file, as is the replication factor of the block which is typically three. The NameNode plays the central role in the ordinary file operations: when a client wants to read a file, he has to communicate with the NameNode first in order to obtain the

[1] http://hadoop.apache.org/
[2] http://www.ibm.com/developerworks/web/library/wa-introhdfs/

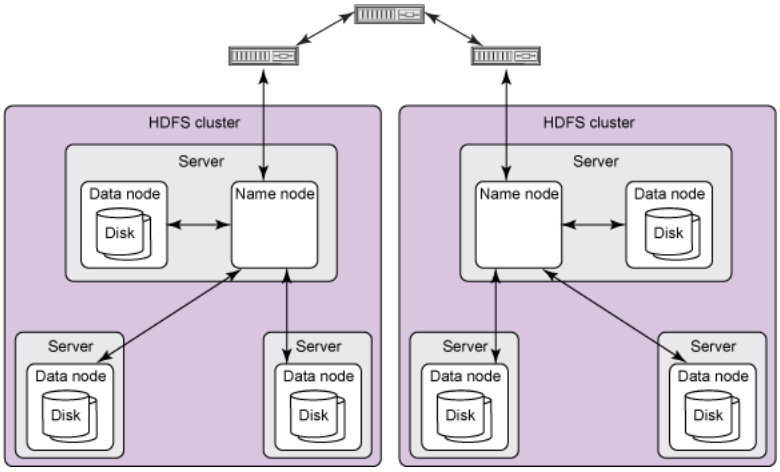

Figure 2.2: Each cluster contains one NameNode. This design facilitates a simplified model for managing each namespace and arbitrating data distribution.

locations of the data blocks that comprise the file, and then read the blocks from the DataNodes that are closer to him. In a write operation the client queries the NameNode for a set of three DataNodes and then he sends the data to the DataNodes in a pipeline fashion. HDFS keeps the entire namespace in memory. The inode data and the list of blocks belonging to each file comprise the metadata of the name system called the image. The persistent record of the image stored in the local host's native file system is called a checkpoint. The NameNode also stores a journal of the image in the local file system. Optionally, the checkpoint and the journal can be copied to other servers as to decrease the chance of losing this crucial data. The NameNode replays the journal to restore the namespace when it is being restarted.

2.2.2 DataNodes

Data blocks are stored in the DataNodes together with the block's metadata that comprises the checksums of the block data and the generation stamp. Unlike traditional file systems, HDFS only allocates the amount of local disk necessary to accommodate the actual data size instead of incurring on the overhead of allocating an entire data block even if the block is only partially used. Each DataNode is assigned the cluster's namespace ID when it is initialized for the first time. Moreover, DataNodes have a unique storage ID that serves the purpose of identifying that specific node even if that DataNode is restarted with a different IP address or port. When a DataNode starts it connects to the NameNode and perform a handshake whose purpose is to check if the DataNode is running the right software version and to verify it its namespace ID is correct. If one of

the verifications fails, then the DataNode shuts down. These restrictions are needed to ensure that one DataNode does not cause data loss or corruption.
When the DataNode first registers with the NameNode, it sends its block report which contains the block id, the generation stamp and the length of each block the DataNode holds. Subsequently, a new block report is sent every hour so that the NameNode can keep track of the replication status of all the blocks in the cluster. DataNodes also need to send heartbeats to the NameNode every three seconds to let it know the block replicas are still available. If the NameNode does not receive any heartbeat from a DataNode in ten minutes, then that DataNode is considered to be down and new block replicas are created and stored in another DataNode. Additionally, the heartbeat messages transport control information like the total size of the storage, the amount available, and the number of operations in progress. This enables the NameNode to make decisions on space allocation and load balancing.

2.2.3 HDFS Client

When a client reads a file it asks the NameNode for the list of DataNodes that have replicas of the blocks of the file. Then it contacts directly the DataNodes and requests the transfer of the block. For the write operation, the client again contacts the NameNode requesting a list of DataNodes that should store replicas of the first block. Using that list, the client makes a pipeline and sends the data. After the first block is filled, the client repeats the process requesting a new list of DataNodes for the next block.
HDFS allows three distinct interfaces - a Java client API; a C language wrapper for this Java API; an HTTP browser interface. Currently the file system does not support appending-writes to files and POSIX requirements are not strictly followed as to enable streaming access to the file system data.

2.3 Ceph

Ceph [5] is an object-based distributed file system that follows the assumption that large scale systems are inherently dynamic, with node failures being the norm rather than the exception and workloads shift over time. It tries to mitigate bottlenecks caused by operations on metadata by using a cluster of metadata nodes which are called metadata servers (MDSs).

Ceph aims to achieve its scalability, performance, reliability and availability goals through three fundamental design features: decoupled data and metadata, dynamic distributed metadata management, and reliable autonomic distributed object storage. Decoupling data from metadata allows clients to use the metadata cluster for typical operations such as open and rename, without impacting other clients that are actually performing operations on the data that is stored in the OSDs. The usage of object-base storage delegates low-level block allocation decisions to individual devices. Ceph does not use allocation lists to index how files are split in blocks in the file system. Instead, each file is striped onto objects whose names can be predicted and then a special function called CRUSH (Controlled Replication Under Scalable Hashing) maps the objects into the storage devices. As a consequence, the layout of the files can be calculated by any party without the need to do any look up, thus reducing the work load of the MDSs.

The second design pillar of Ceph, dynamic distributed metadata management, consists of a metadata cluster architecture based on a dynamic subtree partitioning strategy [6] as to avoid metadata access hotspots and to provide robustness against non-byzantine failures. The third design feature, reliable autonomic distributed object storage, rests on the intelligence (memory and CPU) that the cluster of OSDs have and so, things like data migration, replication, failure detection, and failure recovery are delegated the nodes that store data. Moreover, at high level the OSDs provide a single logical object store and namespace to clients and metadata servers.

2.3.1 Ceph Client

Ceph exposes a POSIX file system interface that is used by the client. Moreover, Ceph supports some POSIX I/O extensions for high-performance computing including the O_LAZY flag for the open system call, allowing the applications to manage their own consistency. The file system can be mounted using either a kernel-space client (since version 2.6.34 of the Linux kernel), or a FUSE client. A client that wants to read a file sends a message to the MDS cluster which reads the directory containing the file, in order to translate the file name into an inode number, and then delivers the client a capability for reading the file. Next, the client uses the inode number together with the striping strategy to compute the object identifiers (OIDs). An OID is mapped to a placement group, a group of OSDs that stores the object and all its replicas. Knowing which OSDs contain the data, the client can then read the file data directly from the OSD cluster. Closing the file consists of giving up the capability issued earlier. The capability specifies the operations permitted - currently it defines the client's ability to read, write, cache reads, and buffer writes. It contains the inode

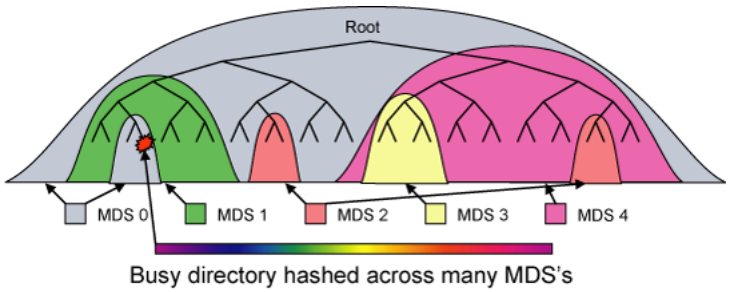

Figure 2.3: Ceph dynamically maps subtrees of the directory hierarchy to metadata servers based on the current workload. Individual directories are hashed across multiple nodes only when they become hot spots.

number, the replication factor, and information about striping strategy of a file, which can be file-specific and is set at file creation time. The write operation is analogous to the read: the client requests a capability to write a file, then it pushes the object contents to the OSD cluster, and after it needs to inform the MDS of the new file size.

2.3.2 Metadata Service

By having small metadata items that consist mostly of file names and inodes (no allocation tables for block mapping), medatata can be moved around quickly with a low impact on the MDS cluster, hence load-balancing can be done continually using dynamic subtree partitioning.
Each directory is stored as an object. Inodes are embedded in directories and stored with the corresponding directory entry (file name). This organization optimizes the common access pattern of listing a directory and retrieving metadata for each file.
The file system's namespace is partitioned across the cluster of MDS nodes along directory subtrees, as shown in Figure 2.3. Subtrees are migrated between MDSs as workload changes. To alleviate hot spots, heavily read directories are replicated on multiple MDSs so that the number of clients knowing about a particular replica can be bounded. Large or heavily written directories are fragmented for load distribution and storage.

2.3.3 Reliable Autonomic Distributed Object Storage (RADOS)

Ceph's clients see the object storage cluster as a single logical object store with a flat namespace, called Reliable Autonomic Distributed Object Storage (RADOS). RADOS sends the clients and the MDSs a cluster map containing information about the OSDs in the system, their status, and the

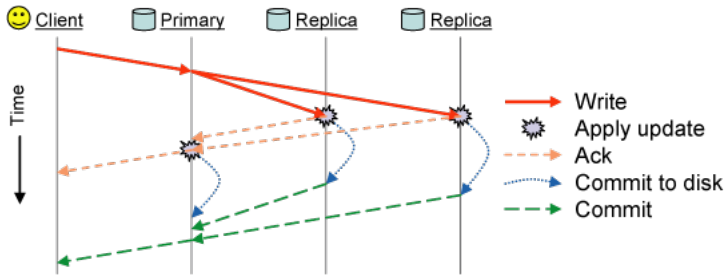

Figure 2.4: Ceph write semantics

CRUSH mapping function. This cluster map is managed by a distributed monitor service which consists of set o monitors that maintain consistency using the Paxos algorithm [7]. The OSDs that are responsible for detecting changes in the cluster (e.g., failed OSDs or new nodes added) and then notify the monitor service. The monitor propagates this information throughout the cluster.

Unlike other distributed file systems, replication in Ceph is managed by the OSDs instead of clients. RADOS manages data replication using a primary-copy strategy. Replicas are stored in placement groups that include a primary OSD and a set of backups. When the primary node of the placement group receives a request, it forwards it to the other OSDs of its placement group. In the case of a write operation, the primary waits for the other OSDs to apply the write before it applies it locally. The client only receives the acknowledgement of the operation from the primary OSD and after the data has been committed to disk the primary sends a second message to client so that he can discard that data from its buffer cache (see Figure 2.4).

An OSD monitors the state of other OSDs in the same placement groups using heartbeat messages, which are usually piggybacked on replication traffic.

2.3.4 Controlled Replication Under Scalable Hashing (CRUSH)

The distribution of the data among the data cluster is done in fashion that seeks to avoid imbalance or load asymmetries. That is, nodes recently added to the cluster should immediately start being used and also the workload (storage and bandwidth) should be distributed in a fair way. Ceph approaches this question by distributing new data randomly; migrating a random subsample of existing data to new devices; uniformly redistributing data from removed devices.

Ceph makes use of a hash function called Controlled Replication Under Scalable Hashing (CRUSH) to map a placement group ID to a list of OSDs, using a hierarchically structured cluster map and placement rules as additional input. The first OSD in the list is the primary and the length of the

list corresponds to the replication factor. Placement rules are defined by an administrator and are assigned to cluster maps. Moreover, this rules permit the definition of failure domains such as racks or shelves. So, using placement rules it is possible to prevent two replicas from being placed in the same failure domain, thus achieving a higher level of data safety.

2.4 Lustre

Lustre used the architecture first developed at Carnegie Mellon University as a research under the CMU NASD project [8]. Its was commercialized by Cluster File Systems which was bought by Sun Microsystems in 2007 aiming the HPC market. In 2010 Oracle acquired Sun Microsystems and since then it stopped the development of Lustre 1.8 and only supports the current 2.0 release on Oracle hardware or approved third party hardware vendors.

Lustre is an object-base file system that encompasses three major components: metadata servers (MDSs) that control metadata targets (MDTs), a set of object storage servers (OSSs) where each OSS controls a set of object storage targets (OSTs), and clients that interact with the file system [9]. The storage attached to the servers (MDS and OSS) is partitioned, organized with logical volume management (LVM) and formatted as file systems. The Lustre OSS and MDS servers read and write data in the format imposed by these file systems.

Each OSS can be responsible for multiple OSTs, one for each volume and I/O traffic is load balanced against servers and targets. The capacity of a Lustre file system is the sum of the capacities provided by the targets. It is possible to use different storage strategies for the OSS, other than internal drives. One could use a storage array attached over Fibre Channel (FC) or Serial Attached SCSI (SAS) connections. Moreover, it is desirable to use hardware or software RAID. In a Lustre file system, storage is only attached to the server nodes, not to the client nodes. If failover capability is desired, storage must be attached to multiple servers. In all cases, the use of storage area networks (SANs) with expensive switches can be avoided because point-to-point connections between the servers and the storage arrays will normally provide the simplest and best attachments. The same approach can be used for the MDS servers, i.e, MDTs can be layed out in a way to provide increased capacity, fault tolerance, and/or performance.

Lustre can be mounted using a Linux kernel module and the clients can interact with the file system using standard POSIX semantics - open, read, write, close, etc. The software that composes the Lustre client is an interface between the Linux Virtual File System and the Lustre servers. For

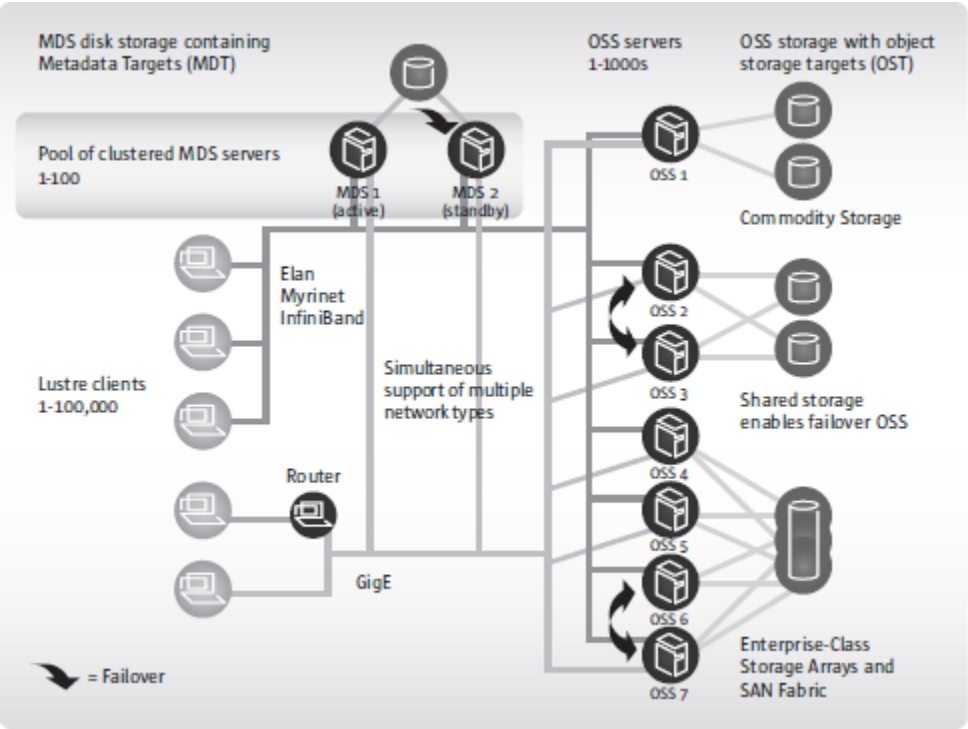

Figure 2.5: Lustre architecture

every target there is a client counterpart, that is, the MDT deals with a Metadata Client (MDC) and the OST deals with a Object Storage Client (OSC). These components combined provide a single coherent namespace to the client mounting the file system.

The MDTs hold inodes that point to objects that are implemented as native files in the OSTs file systems. A client writing a file in Lustre issues a file open request to the MDT through the Metadata Client component. The MDT retrieves corresponding inode and sends the list of objects that compose the file - the MDT knows the layout of the file in Lustre. Using the list of pointers returned by the MST, the client sends parallel write requests to the relevant OSTs using the corresponding OSCs.

2.4.1 Lustre Networking (LNET)

The disk storage behind the MDS and OSS servers in a Lustre file system is connected to these servers using traditional SAN technologies. But this SAN does not extend to the clients and so, Lustre Networking (LNET) is usd to connect the servers and the clients. LNET is a message passing API that handles metadata and file I/O data and supports multiple, heterogeneous interfaces on clients and servers. Its most relevant features include Remote Direct Memory Access (RDMA)

should the underlying networks support it; support for different networks like IP and Infiniband; high-availability and transparent recovery; simultaneous availability of multiple network types with routing between them.

2.5 Sector

Sector [10] is a distributed file system that is meant to provide storage services over a wide geographic scope like the Internet. Its primary usage is the support of Sphere which is a computing framework that provides an API for writing distributed data-intensive applications.
The assumptions on Sector are that the nodes are connected with high-speed networks: 1 Gb/s connecting nodes within the same rack; 10 Gb/s network between two racks in the same data center; 10 Gb/s connecting two distinct data centers. In addition, Sector assumes that the datasets it stores are divided into one or more files, which are called Sector slices. The slices are replicated and distributed over the data nodes.
The architecture of a Sector system comprises a security server, a master and a set of slave nodes - Figure 2.6. The security server is responsible for the management of user accounts, user passwords and file access information. It also keeps a list of IP addresses of the authorized slave nodes, so it acts as a firewall that rejects connections from slave nodes that are not allowed to join the system. The master server is the metadata node but it also acts as gateway between the clients and the security server. Furthermore, the master node controls the slaves keeping track of the hosts that are running and mediating the communication with the security server. The slave nodes are the servers that store the files managed by the system and process the data requested by the Sector clients.

2.5.1 File system management

Sector slices are arranged in the native file system of the slaves without being broken down, i.e., each slice corresponds to a single file in the slave node's file system. This design strategy enables easy metadata recovery by simply scanning the data directories on the slave nodes. It also makes it possible for a user to send and receive files from the system by connecting to a single slave node - the alternative, that is used by storage systems that manage files at block level, requires that the user connects to (usually) many nodes in order to obtain all the blocks that make up a file. The

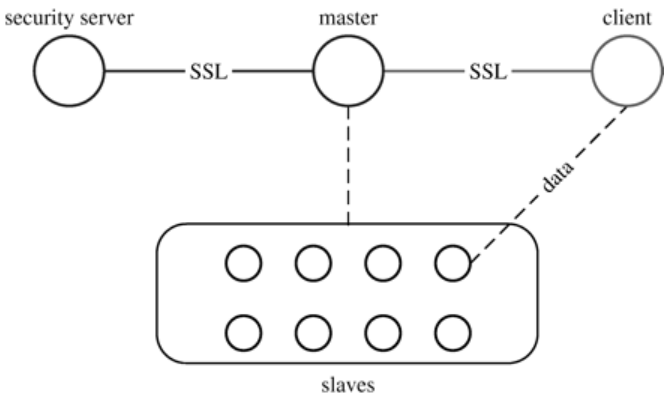

Figure 2.6: Sector architecture

drawback of this approach is that it forces the user to divide large datasets into chunks that can fit in the native file system of the slaves.

The master keeps the metadata index required by Sector and responds to file system queries, such as file lookup and directory services. Moreover, it manages the state of the whole storage system by keeping track of the disk usage in each slave and system topology, so that it can choose the slaves that are likely to yield better performance. The topology of the system is specified in a configuration file on the master server and it is assumed to be hierarchical, e.g. computer nodes on racks within multiple data centers.

The replication of the slices is managed by the master node. It periodically checks if the number of copies of each file is above a defined threshold. If that is not the case then the master chooses a slave to make a new copy of the file. The location of the copy is decided according with the topology of the slave's network. When a client requests a file, the master chooses a slave that is close to the client and is not busy with other services. Sector supports the FUSE interface, presenting a mountable file system that is accessible via standard command-line tools.

2.5.2 Security

The usage of a separate security server makes it possible to deploy several kinds of security features. For instance, it could be used to run Kerberos or other protocols in order to provide authentication to the users. A new user connecting to the system establishes an SSL connection with the master server. The user name and password are sent to the master, which queries the security server for the validity of the credentials, using a new SSL connection. In case the security server validates the credentials (including the IP address of the client), it generates a session ID and a list

of the client's file access privileges. This information is sent back to the master node. So, when a client requests access to a file the master node checks if the user has the necessary privileges and chooses a slave node to serve the file if the user has the right access privileges. Finally a connection between the slave and client is established - no encryption is used in this connection. The slaves only accept commands from the master, and the slave-slave and client-slave data transfers are coordinated by the master server. The security server keeps a list of IP addresses and/or IP address ranges of authorized slave nodes. Slave connections from outside the authorized IP addresses are rejected.

2.6 XtreemFS

XtreemFS [11] is an object-based file system aimed at Grid environments, as traditional file systems lack the properties that enable reliable file access over wide-area networks across multiple organizations. Grid data management systems provide their clients with access to data that is stored at remote storage and file systems. Typically these systems index the files contained in the storage devices and provide a unified interface for accessing the files. The clients connect to a daemon, download the files they want to the local file system, change the files as they wish and then upload the modified file to the remote storage. This is simple scheme that provides the basic functionalities needed while allowing the usage of heterogeneous storage resources. However it presents several limitations namely the lack of control on the whole system - the daemon only serves as a mediator to its own files and is not aware of the system's state, and also it does not know if and how many replicas there are for each file. So, with this approach the Grid data system cannot make guarantees about the consistency of the file's replicas. XtreemFS' design follows the same pattern as other object-based file systems. It uses OSDs to store the data and metadata servers to keep track of the layout of the data in the OSDs. XtreemFS has been specifically design to support file replication and federated setups in Grid environments. XtreemFS organizes its file systems in file system volumes, each of which represents a mountable file system with a separate directory structure and configuration. A volume's files and directories share certain default policies for replication and access.

2.6.1 Metadata and Replica Catalogs (MRC)

XtreemFS may use more than one metadata servers which are called Metadata and Replica Catalogs (MRCs). Each MRC can host multiple volumes and act as a metadata server for clients. Before being able to access a file, clients must authenticate to the MRC by presenting the user's credentials which can be an X.509 certificate.

An MRC stores the volume metadata that includes standard file information - size, access times, etc. - along with the striping strategy and list of OSDs that contain replicas of the file. This metadata can be replicated across other MRCs and therefore a volume can be hosted by multiple MRCs, which increases the volume availability and access performance.

2.6.2 Directory Service

In addition to the OSDs and MRCs, XtreemFS features a directory server providing a service that connects all of the components of the file system. This service allows the clients to discover the MRCs that manage a certain volume but it registers all the servers as well. Moreover the directory service can act as a shared store between MRCs and OSDs.

The authentication token issued to an authorized client is signed and the verification of the signature by the OSDs and MRCs is supported by the directory service in the sense that it stores a shared secret that is used for the signature-related operations. Furthermore, XtreemFS relies on absolute timeouts for some of the algorithms in other components, so the directory service is also used as the central time source for the whole system.

Another useful thing of the directory service is that it keeps dynamic information like load and capacity information for OSDs, which MRCs can periodically query as to optimize the resource usage.

2.6.3 Object Storage Devices

When a client accesses data on an OSD for the first time, its request includes information for the OSD about the locations of other replicas of the file, which allows the OSD to coordinate operations in a peer-to-peer manner. No central component is involved in this process, which makes it scalable and fault-tolerant. The operations on file data are coordinated using leases [12] (locks that time out and allow the optimization of consensus algorithms such as Paxos) and so the holder of the data

is able to define the latest version of the data with low communication effort. OSDs keep version numbers of the files objects they store which are used to determine if their local data is current. By leveraging the lease coordination protocol, XtreemFS can guarantee POSIX semantics even in the presence of concurrent accesses to the replicas.

2.7 Final remarks

The file systems outlined in this Chapter use architectures based on I/O services provided by object-based storage devices (OSDs). Other systems based on the Common Internet File System (CIFS) [13] and the Network File System (NFS) [14] follow a client-server model, with a server at a single location storing all the data and multiple clients accessing this data. In the object-based storage paradigm [15], the data primitive referenced by the file system is not a block, but an object that may provide a rich interface enabling application-specific methods for manipulating data. The OSD has the disk, a processor, RAM and a network interface that allows it to manage the local object store and autonomously serve and store data from the network, thus shifting the burden of space management (i.e., allocation and tracking of used and free blocks) away from storage applications (e.g., file systems and databases).

It has been shown that metadata transactions account for over 50% of all file system operations [16]. For this reason, one of the key aspects shared across the systems is the decoupling of file content accesses and metadata transactions to achieve high data throughput. A client contacts a metadata server first to acquire access permission and obtain desired file metadata, such as data location and file attributes, and then directly accesses file content stored on data servers without going through the metadata server. That is the case for HDFS, Ceph, Lustre, Sector, and XtreemFS. However, using a metadata node creates a potential point of failure that can lead to availability problems in case of hardware or software faults and in case of metadata corruption the whole system may become unrecoverable. To address this, Ceph, XtreemFS, Lustre, and Sector employ replication strategies on the metadata node. Each one uses different schemes: Lustre only supports up to two metadata servers per file system in an active/passive failover configuration; both Sector and Ceph enable multiple active metadata nodes that perform load balancing; XtreemFS has metadata and directory services replication using a primary/backup scheme but metadata replication is not considered to be stable. Gluster does not make use of metadata servers. Instead it stores all the metadata together with the file contents inside the storage servers (called bricks) and the client

alone is able to derive where data is located using a custom hash algorithm. Therefore the client does not need to contact an intermediary prior to interacting with the data store nodes.

The interfaces offered by the file systems to applications and clients have slight differences. Mounting the file system so that it can viewed as a local directory tree is a major advantage for administration and operation purposes, for it allows the use of ordinary file system utilites. Further, if it exposes a POSIX-compliant interface then the application can make use of it without requiring modifications. However, POSIX constraints in the scope of parallel file systems are difficult to ensure. For example, with multiple clients modifying the file chunks in parallel, no entity in the file system knows the location of the last block and thus neither metadata servers, storage servers nor clients know the current file size. As a consequence, some file systems choose to sacrifice consistency of concurrent file access or do not fully implement the POSIX interface. All of the analyzed file systems can be mounted using FUSE even though it is not the advised method for some of them. HDFS provides a Java API (a C wrapper is also provided) that should be preferred, but it forces the application to be written as to make use of it. Moreover HDFS follows a write-once approach which means that no write operations can be done after the file is closed. Ceph's client driver has been included in the Linux tree since version 2.6.34 and so it is possible to mount the file system without any additional patch or workaround. Likewise, since Linux 2.6.16 it has been possible to run Lustre clients without any patches. GlusterFS includes a customized FUSE driver and it is one of the advised ways to use it.

In general, high performance is the primary goal for the systems analyzed and, therefore, security has been left as a side matter. Encryption of traffic between client and servers and among servers is present in XtreemFS which supports SSL connections with X.509 certificates. Ceph uses SSL as well in the connections between clients, master nodes, and the security server. It does not encrypt the traffic that goes between clients and slave nodes. The other file systems generally assume by design that all the intervenients are part of the same trusted secure domain. Still, they employ some authorization mechanisms that aim to provide some degree of access control. Gluster does not allow clients connecting from unprivileged ports (above 1024), so only users with root access can connect to the file system. Moreover it permits an authentication scheme that can be a combination of source IP address white listing and/or black listing plus username and password. Sector makes use of a security server as to maintain user credentials, permissions, and ACLs.

In this Chapter it was shown the fundamentals of some distributed file systems and it was possible to find not only some similarities in the way they are built, but also the key differences in terms of design and implementation. The next Chapter exposes the results of an experimentation with one

of the file systems as a proof-of-concept for the application on Portugal Telecom's event storage solution.

Chapter 3

Analysis

In the previous chapter I depicted several parallel file systems and explained their main features. With that analysis it was possible to know the most relevant properties of each file system like the goals it tries to accomplish, whether it uses a shared nothing architecture, the ability to mount it like a local file system, the security attributes it uses, its current limitations, and its development roadmap. That knowledge is necessary to understand if the systems can meet the requirements for Portugal Telecom's SIEM.

In Portugal Telecom the workflow of security data is implemented using standard industry practices. It consists of a security information and event management system and its main features are the following:

- Data aggregation - Aggregate data from many sources, including network, security, servers, databases, applications, providing the ability to consolidate monitored data to help avoid missing crucial events;
- Data correlation - Looking for common attributes, and links events together into meaningful bundles as to provide the ability to perform a variety of correlation techniques to integrate different sources, in order to turn data into useful information;
- Alerting - The automated analysis of correlated events and production of alerts, to notify recipients of immediate issues;
- Dashboards - Take event data and turn it into informational charts to assist in seeing patterns, or identifying activity that is not forming a standard pattern;

- Compliance - Automate the gathering of compliance data, producing reports that adapt to existing security, governance and auditing processes;
- Retention - Long-term storage of historical data to facilitate correlation of data over time, and to provide the retention necessary for compliance requirements.

In the particular case of Portugal Telecom the SIEM in production is *ArcSight ESM* [1]. Currently one instance of Portugal Telecom's SIEM collects about 50GB of daily log data arriving via a syslog server and split across two separate files one with a size of about 30GB and the other with the remaining 16GB of data. The data is separated according with the source that generated the events. The inflow of messages is fairly constant with meaning that throughout the 24 hour period the files grow in a constant rate no matter the time. On average the arrival of messages is in the order of 3000 messages per second where each message is around 180 bytes wide. Hence the throughput the system supports is about 600KB/sec.

Both files are daily compressed to a single file by a cron job so that the events are written in new files every day. Since the contents of those files is in text format the compression reduces the data to about 5GB. So, every day the data stored in the system is incremented by approximately 5GB. So the expected file system usage pattern is mostly for write operations given the objective of making it a long term storage system, where the eventual file reads can be done offline with prior copying for a different location.

In this chapter it is presented the results of the performance tests done using some benchmark tools in order to assess the practicability of having one of the previously described file systems to fulfill the role of the storage system for the scenario in question.

3.1 Testing setup

The tests were performed in a cluster running Debian version 5.0.8 with a 32-bit Linux kernel based on version 2.6.26 and EXT3 as the native file system on a SATA disk. The machines had 2GB of RAM and were equipped with single Intel Xeon CPUs with Hyperthread enabled running at 3.00GHz. Using iperf [2] it was observed that the effective bandwidth for TCP connections of the links connecting the servers was around 600Mbits/sec.

The tools used to conduct the file systems' performance benchmarks were the following: IOzone

[1] http://www.arcsight.com/products/products-esm/
[2] http://sourceforge.net/projects/iperf/

[3]; postmark [4]. IOzone is a file system benchmark tool which generates and measures a variety of file operations. The benchmark tests file I/O performance for the operations like read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read/write, pread/pwrite variants, aio_read, aio_write, mm, etc. Postmark is another file system benchmark tool that tries to emulate the behavior of large mail servers and news servers, which mainly do small file operations.

The tests executed have three goals: 1) assess the performance of the system under different workloads, and to do this it is important to measure both the client and the storage servers; 2) check if the solution has the scalability properties that are advertised, i.e., make sure that adding nodes to the system does not affect the performance negatively; and 3) confirm the applicability of the solution, which is to say that the proposed file system is adequate for the use case described earlier.

For each tool, four sets of tests were executed with different storage strategies: striping with two storage nodes; striping with three storage nodes; replication with two storage nodes; replication with three storage nodes. The purpose of having these scenarios is to get to know if and how the system behaves differently depending on the way data is stored and at the same time find out what are the overall effects of adding servers to the store pool.

3.2 Choosing between DFSs

The choice between the analyzed file systems has to be thought of as weighing the prons and cons of each, not only in terms of technical features but also in terms of philosophy behind them. GlusterFS presents a shared nothing architecture, it is POSIX-compliant and the file system is mountable using FUSE. It also includes self-healing capabilities that enable automatic recovery of inconsistent or faulty files/volumes. Moreover because GlusterFS stores the files in the native files system's format means that management of individual nodes can be done in a more traditional way using commonly used tools. Because of these reasons it is a good candidate for the storage solution that Portugal Telecom seeks.

Although HDFS is mountable using FUSE it does not support append writes, meaning that it is not possible to re-open a file for writing. This constraint alone would not be an obstacle to solve the problem in hands because the intended usage of the file system is based on a continuous write of the file from start until the end. But, in a failure situation where the application or the file system

[3] http://www.iozone.org/

[4] http://www.netapp.com/tech_library/postmark.html

would have to be restarted, the existing files would have to manually be renamed so that new files could be created. An alternative would be changing the application so that it could fit the HDFS model, which is not desirable.

Ceph presents interesting features that seem suitable for the desired solution, especially POSIX semantics and its kernel-integrated client. However Ceph's developers state that it is not yet ready for production usage. Furthermore Ceph was only integrated in the Linux kernel starting from version 2.6.34 whereas the test machines only have version 2.6.26 installed and, because of time constraints, installing a different kernel version should be avoided.

Lustre, on the other hand, is under the control of Oracle and its development path is not very clear especially in terms of its licensing model. Portugal Telecom seeks a solution with active development and support, but Oracle is only supporting Lustre 2.0 in limited hardware, which may rise the costs of the implementation.

Sector distributed file system, enables the usage of multiple active master servers as to enable high availability of the metadata information. Also, it is FUSE compatible and clients automatically switch to good master/slave nodes if the current connected one is down.

XtreemFS has metadata and directory services replication using a primary/backup scheme, but as of the current stable release, the automatic failover is not supported by the client and other services and metadata replication should be considered highly experimental.

From the analysed options, GlusterFS and Sector show the basic features that fulfill Portugal Telecom's requirements - fault tolerance, availability, openness, and active development. Between these two options Gluster was chosen for the subsequent tests because it is less demanding on the configuration - Sector needs a security server and at least a master server in addition to the storage nodes.

3.3 Gluster performance

GlusterFS makes it possible to mount the file system using FUSE and Gluster's software includes a native client which mounts the volumes on the client machines using the FUSE interface. This mounted volume offers a fully POSIX compliant interface. For the benchmark tests a cluster with four servers was used where the first one mounted a remote volume which was stored in the remaining servers. In the next sections it is shown the results of the tests with the benchmark tools and different storage strategies on GlusterFS.

Gluster is designed to support environments with large numbers of clients. Since the I/O of individual clients is often limited, system throughput is generally greater with many clients operating at the same time. So, it should be noted that the tests presented here refer to a particular case where a single client is connected and therefore the results should be interpreted accordingly. Moreover, the tests were executed without changing the configuration between scenarios and for that reason the results are not the optimal because a lot of tunning could be done to better suit each specific test case. Nevertheless the results are important to establish a starting point for future optimizations.

3.3.1 Postmark tests

The postmark test consists of creating a directory structure; generation of an initial pool of random text files; perform a set of transactions on the files, where each transaction consists of a pair of smaller transactions: create or delete file, read or append file; in the end all the files are deleted. For this particular test the number of directories to create was set to 448, the number of files was 200000, and the number of transactions was 50000.

3.3.1.1 Striping

The outcome of the tests using striping shows that Gluster's performance is worse with 3 nodes than it is with only 2 storage nodes. In particular, the creation and deletion of files presented a high performance penalty but not so much in the transactions. With 2 nodes the creation rate was 256 files/sec and the deletion rate was 145 files/sec, whereas with 3 storage nodes the file creation rate was 128 files/sec and the deletion rate was 122 files/sec. As a consequence, the test took 3483 seconds to complete in the former case while it lasted for 6010 seconds in the latter.
Such a high discrepancy can be explained by the nature of the test which creates lots of small files, imposing a high overhead in terms of metadata operations. Therefore, the network latency can become a limiting factor in the result of this type of usage of the file system.
During the execution of this benchmark none of the nodes including the client showed to be limited by CPU capacity or available memory. Figures 3.1 and 3.2 show that Gluster striping makes the first node work more than the others, despite the configuration being the same for all of them.

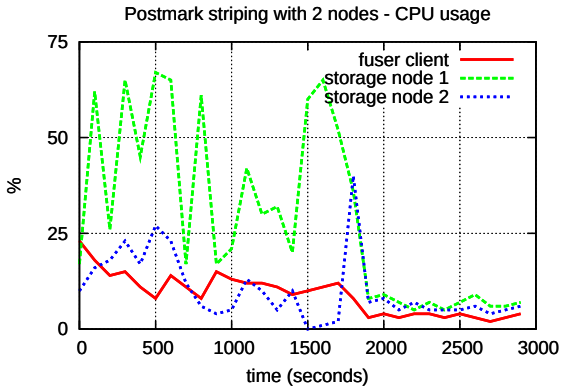

Figure 3.1: Postmark CPU usage with striping on 2 nodes

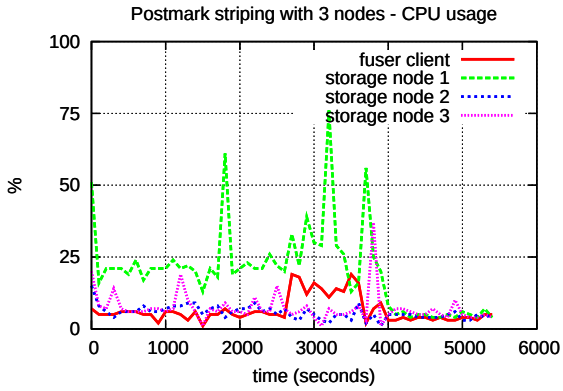

Figure 3.2: Postmark CPU usage with striping on 3 nodes

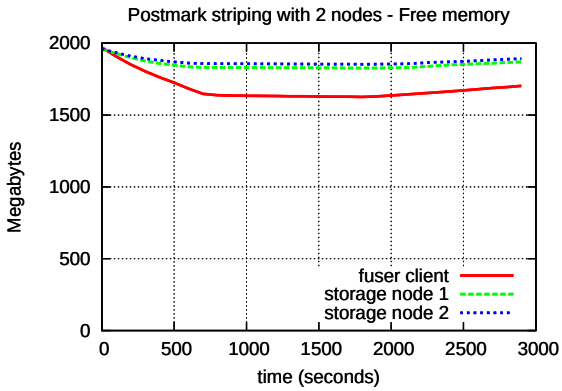

Figure 3.3: Postmark free memory with striping on 2 nodes

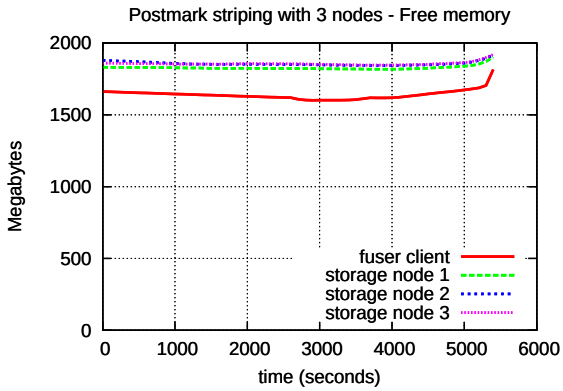

Figure 3.4: Postmark free memory with striping on 3 nodes

3.3.1.2 Replication

Gluster provides the ability of replicating volumes in order to mirror data across clusters. This allows the set up of high availability storage as to deal with node failures.

Unlike the striping case seen before GlusterFS behaved in a more consistent fashion in the scenario where the data is replicated throughout the storage bricks. Particularly, the CPU usage was fairly equal in all the storage servers (Figures 3.5 and 3.6) as opposed to the striping case in which one of the nodes was under more load than the remaining nodes. Furthermore, the overall CPU usage pattern with 2 or 3 nodes is quite similar.

As for the throughput, it is a bit higher with 2 bricks - 161 files created per second and 125 files deleted per second - than it is with 3 bricks - creation rate is 152 files/sec and deletion rate is 98 files/sec.

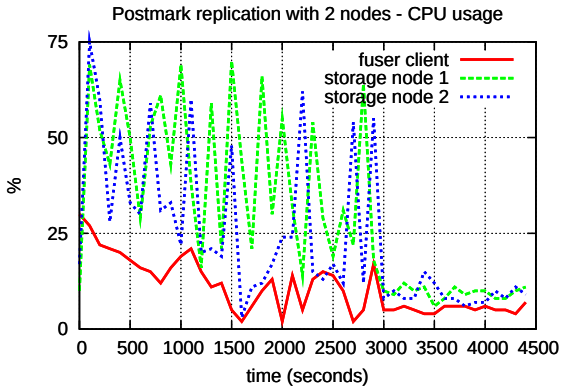

Figure 3.5: Postmark CPU usage with replication on 2 nodes

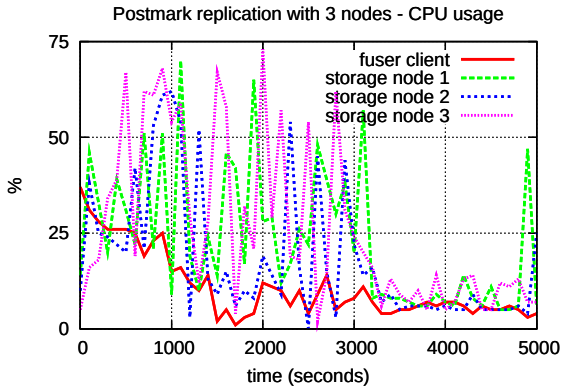

Figure 3.6: Postmark CPU usage with replication on 3 nodes

	Creation rate (files/sec)	Deletion rate (files/sec)
Striping with 2 nodes	256	145
Striping with 3 nodes	128	122
Replication with 2 nodes	161	125
Replication with 3 nodes	152	98

Table 3.1: Postmark results

3.3.2 IOzone tests

IOzone automates a wide range of I/O operations including variations on file and buffer sizes and so it simplifies the process to ascertain the performance limits of the file system. Distributed file systems are not suited for handling small files for they usually cause a negative impact on the efficiency of the whole system, hence IOzone tests are useful to correlate I/O throughput with file size.

3.3.2.1 Striping

Unlike the Postmark test which is more focused on file creation and deletion, IOzone does more data operations and so the system shows a very different behavior. The most significant difference is the CPU usage on the nodes (Figures 3.7 and 3.8) that remains relatively low on the bricks and higher on the node that mounts the file system. The load on the storage nodes was expected to be low because in the IOzone test does not invoke a high amount of metadata operations namely file creations and deletions. On the other hand, the high CPU usage on the client node is caused by the FUSE program which is which is a user-space driver and has a high overhead. An important conclusion of this test execution is the uniformity of the system in terms of resource consumption

independently of the number of storage bricks.

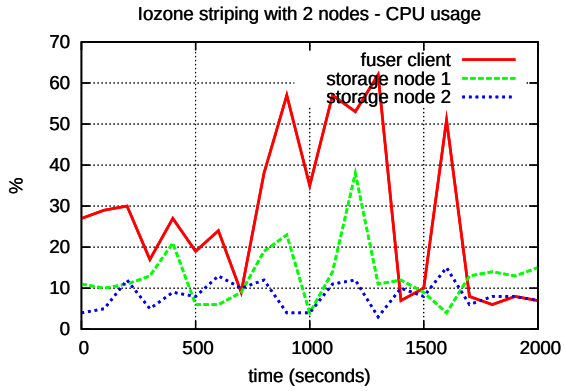

Figure 3.7: IOzone CPU usage with striping on 2 nodes

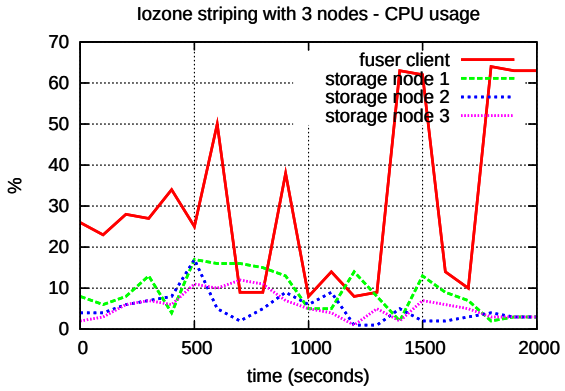

Figure 3.8: IOzone CPU usage with striping on 3 nodes

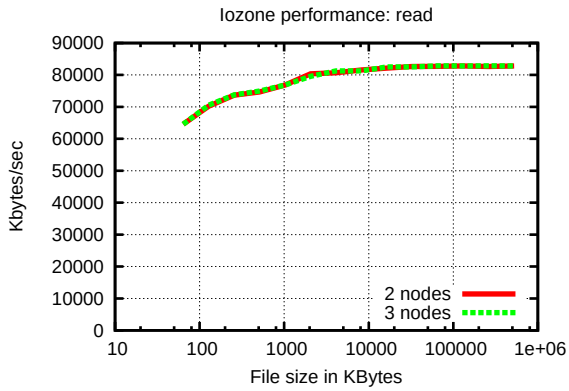

Figure 3.9: IOzone read performance with striping

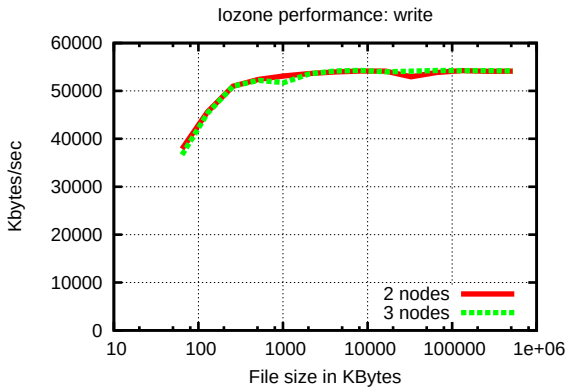

Figure 3.10: IOzone write performance with striping

Gluster's performance with a striping strategy remains constant even with different storage bricks in the cluster. Furthermore, the throughput of the system improves as the file size grows reaching a point of stability when the file is around 16MB, both for read and write operations.

3.3.2.2 Replication

The CPU usage on the nodes using replication of data on Gluster was very similar to the case where striping was used instead. The same pattern is observed - the processor remains mostly unused in the storage nodes and the FUSE client consumes more CPU resources.

The advantage of using replication besides increasing availability of data is the potential gain in performance on reading operations. The Gluster native client can read from different storage nodes

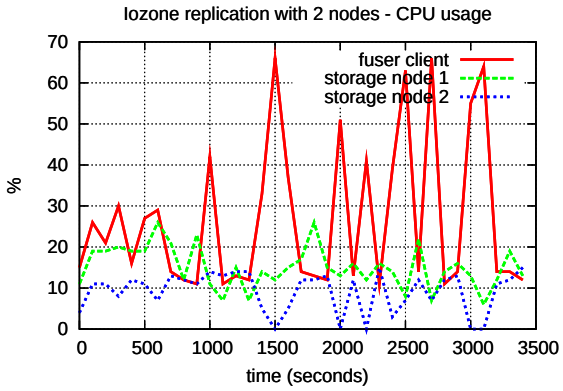

Figure 3.11: IOzone CPU usage with replication on 2 nodes

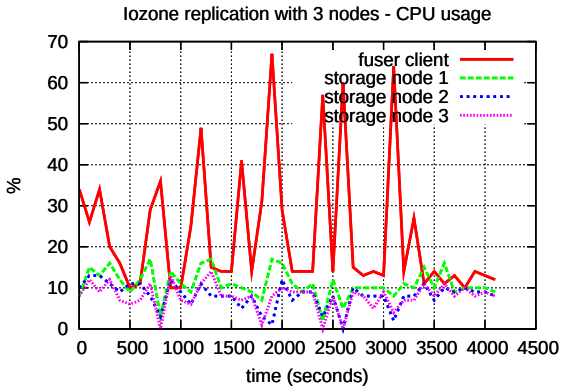

Figure 3.12: IOzone CPU usage with replication on 3 nodes

simultaneously and therefore the reading throughput can be improved by adding more nodes to the cluster.

However the results of IOzone showed reading operations were not faster with one additional storage node; moreover the reading speed in Kbytes/sec is very similar to what was observed in the striping scenario. That is because the number of nodes in this setup is very small and a significant improvement should only happen when dealing a large amount of storage bricks.

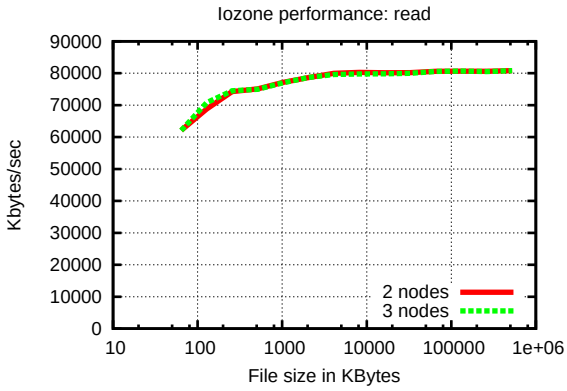

Figure 3.13: IOzone read performance with replication

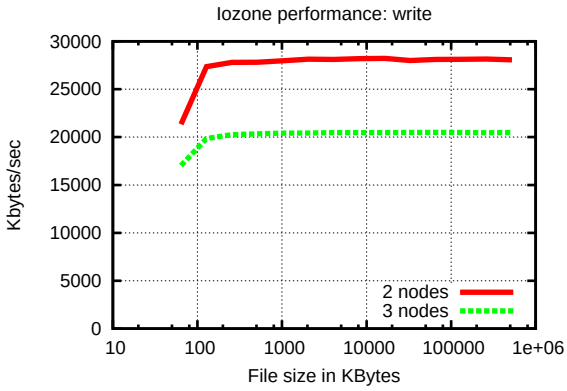

Figure 3.14: IOzone write performance with replication

As for the write operation, it is slower using replication - figure 3.14. Data replication in Gluster is done in a synchronous way and it can be configured in an active-active or active-passive scheme [17]. For the tests presented here the nodes are both active. As a result the performance of write operations decreases linearly as the number of storage bricks increases. The results of the rewrite and randwrite operations (figures 3.15 and 3.16) confirm the behavior observed in the write operation.

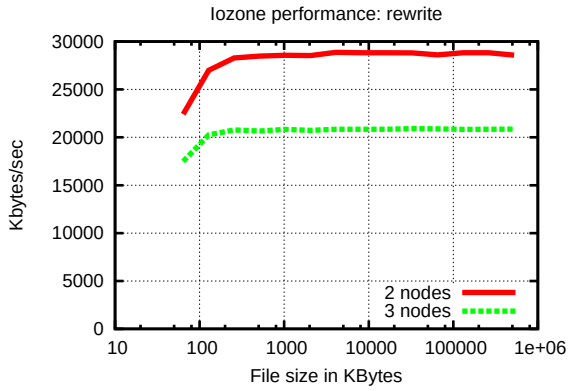

Figure 3.15: IOzone rewrite performance with replication

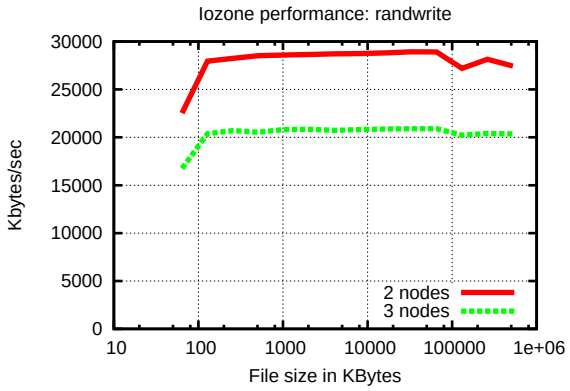

Figure 3.16: IOzone randwrite performance with replication

3.4 Implementing a prototype

The ability to mount the remote file system and the availability of the POSIX interface means that existing applications that interact with a local file system can start using the remote storage without any code modification, just by changing the directory where they are reading and writing data. If this was not the case then the applications would need to be modified as to comply with the available interface. In order to emulate the scenario described earlier that consists of two write streams that build two files, I put two syslog-ng [5] daemons running waiting TCP connections on two different ports, one for each log stream. The syslog instances write the log data they receive in the mount point of the GlusterFS, thus the syslogs are the clients of the file system - figure 3.17.

I developed a log re-player that takes one log file as input and sends it to a syslog server through a TCP connection, while simulating the pace of the arrival of the messages on the original log file taking advantage of the timestamps that each message had. Using this program I was able to reproduce the incoming rate of messages at Portugal Telecom's SIEM, therefore it allowed me to validate the I/O requirements of the system.

3.4.1 Securing the solution

Gluster provides insufficient security mechanisms to protect the information being transfered between nodes and the data being stored. The only protection provided is the following:

- Gluster does not allow clients from an unpriviliged TCP port (under 1024) to connect. This is

[5] http://www.balabit.com/network-security/syslog-ng/opensource-logging-system/

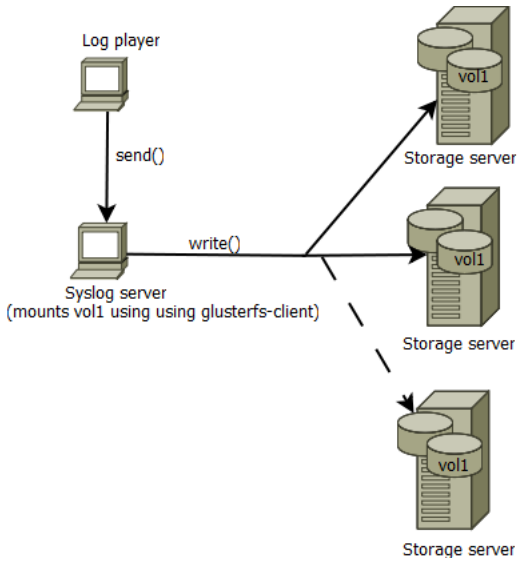

Figure 3.17: The file system client mounts the volume through FUSE

to say that Gluster only allows connections from clients that have root-level privileges on the machine that will mount the remote file system. While this makes some sense, because it prevents regular users from arbitrarily connecting to the storage bricks, it is somehow useless because the FUSE driver already requires root-level access to the client machine. Besides, a malicious user that already has access to an authorized client host could work out some way to escalate his privileges and then he would be able to connect to the remote file system;

- Another method of protection provided is source IP address filtering. Gluster's configuration enables source IP address white listing and/or black listing as to prevent unauthorized hosts from connecting to the system. Again, this security mechanism alone is insufficient to guarantee non-malicious usage of the file system because an adversary can easily spoof his IP address and gain access to the system;

- Gluster makes it possible to add authentication to the clients by setting up a user + password pair and forcing the presentation of the credentials on the connection initiation.

Gluster does not offer the ability to encrypt communications, it only implements features focused on access control.

Given the sensitivity of the data that Portugal Telecom wants to store in the file system, one needs to address the lack of protection of the data in transit in the network from the client to the storage nodes. It is expected that the storage nodes live in a trusted network that should have protection measures in place in order to provide perimeter security. Usually this is achieved by placing a fire-

wall in front of the storage servers mediating the connections from the outside to the inside of the network. So, for this setup it is assumed that no malicious actions can be done by and among the storage bricks.

For the sake of making sure that no malicious third-party can interfere with the data transfered between client and servers, secrecy and integrity of the messages must be ensured. In order to achieve this purpose, I setup a scheme where the connections from the client to the storage nodes go through SSH tunnels which secures the communications, thus preventing a malicious party from being able to obtain the information being conveyed and tampering with it as well.

The way the tunnels were setup using OpenSSH, by generating an asymmetric key pair - in particular 2048 bits RSA key. The client holds the private key and the corresponding public key is added to every storage node's authorized keys. The purpose of using key-based authentication is mostly because it enables passwordless SSH connections which is benefitial in terms of management and automation. Gluster makes it possible to define access credentials on a per brick basis, i.e., different bricks that may or may not be part of the same storage volume can have different usernames and passwords for authentication. Using a single key pair and distributing the same corresponding public key throughout all the storage nodes defeats that purpose. However, one could generate many keys as to use different encryption keys for different storage servers, at the cost of increasing the management complexity.

The default TCP port of the Gluster daemons are numbered from 6996 and incremented by each brick running inside the same server. Using a secure tunnel meant that the Gluster daemon was no longer receiving client connections from remote hosts, instead the client connections were comming from itself and with a TCP source port that was assigned by the client operating system which was not above 1024. Because of this the Gluster daemon rejected the client connections. The way this was circumvented was by changing the source code of Gluster and remove that verification. Additionally the use of the tunnel enabled the implementation of iptables rules on the storage nodes having them only accept connections to the Gluster port from the localhost.

As a result of this implementation it was achieved a more secure solution providing authenticity and secrecy of the data exchanged between the client and the storage nodes with the added benefit of having the storage servers less exposed to remote interactions.

Of course that adding an extra layer to the system has an impact on its performance. The following sections show the results of the benchmarks using the same tools as before in order to provide the means to assess if the added security layer still enables Gluster on the use case described earlier.

3.4.2 Postmark tests

The results of the Postmark test show that the overhead the SSH tunnel imposes for small file operations is not very significant on CPU usage. In fact, the processor usage pattern is very similar to when there is no SSH involved. Indeed, even with SSH involved in the striping scenario, the first storage node is the one that presents the higher CPU load just like for the tests without SSH shown earlier. Moreover, when the test is done over a replication setup, the processor usage is similar among the storage servers as it was for the equivalent case without SSH. Likewise, the memory consumption on the nodes of the cluster remains very close to the observed on the tests done without the encrypted tunnel.

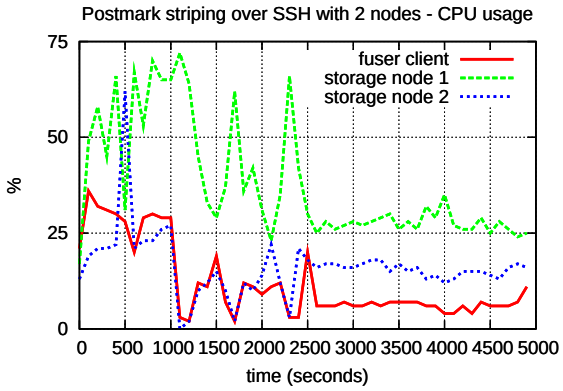

Figure 3.18: Postmark CPU usage with striping over SSH on 2 nodes

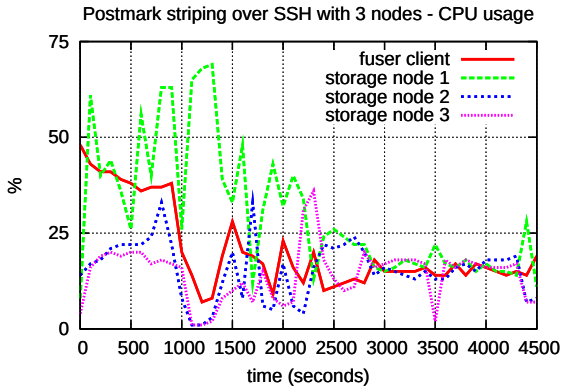

Figure 3.19: Postmark CPU usage with striping over SSH on 3 nodes

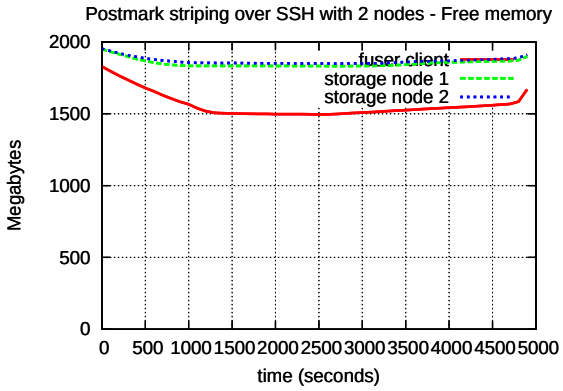

Figure 3.20: Postmark free memory with striping over SSH on 2 nodes

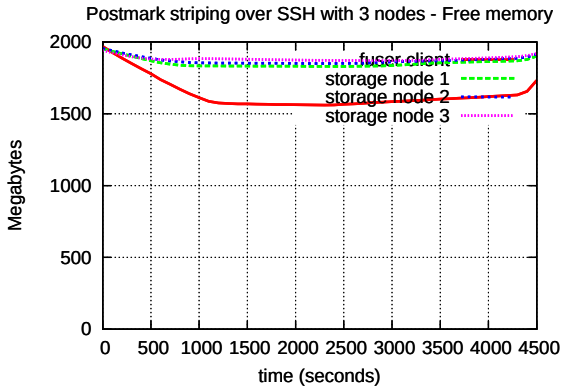

Figure 3.21: Postmark free memory with striping over SSH on 3 nodes

There is, however a general decrease in the performance measured by file creation and deletion rates, especially for the latter - see Table 3.2.

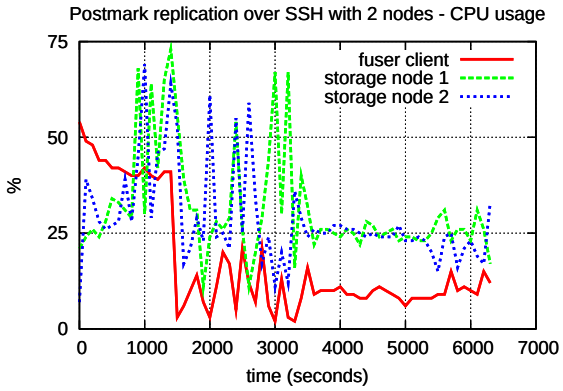

Figure 3.22: Postmark CPU usage with replication over SSH on 2 nodes

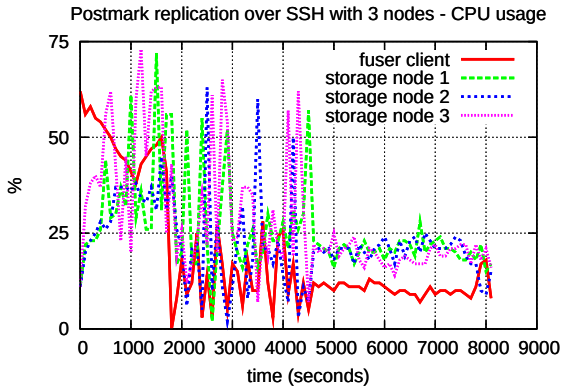

Figure 3.23: Postmark CPU usage with replication over SSH on 3 nodes

	Creation rate (files/sec)	Deletion rate (files/sec)
Striping with 2 nodes	256	145
Striping with 3 nodes	128	122
Striping with 2 nodes + SSH	173	75
Striping with 3 nodes + SSH	168	73
Replication with 2 nodes	161	125
Replication with 3 nodes	152	98
Replication with 2 nodes + SSH	130	67
Replication with 3 nodes + SSH	117	58

Table 3.2: Postmark results comparison

3.4.3 IOzone tests

The IOzone benchmark shows that the Gluster client becomes a performance bottleneck when the connections are tunnelled through SSH. It was expected the increase in CPU usage in this case, for the encryption and generation of the MAC for the network packets occupies the processor. Furthemore, the inherent network overhead of an SSH connection brings the network throughput to lower levels. Every network packet gets additional padding on an SSH tunnel in order to make the packet vary in size, and additionally the MAC takes around 20 bytes depending on the chosen algorithm. As a consequence, the CPU load became higher throughout this test where there is a high amount of traffic being generated.

The bottleneck on the client ends up by taking away the expected extra work the nodes would have to do otherwise. That is, if the FUSE client were able to cope with the work demand without reaching its capacity, then the server nodes would have a much higher CPU usage caused by the expensive cryptographic operations needed to decrypt the traffic. But, as shown in the replication test - Figures 3.24 and 3.25 -, the storage nodes have low processor occupancy ranging from 10% to 30%, which is very close to the numbers obtained in the tests without the SSH tunnel. Changing

the storage scheme to replication confirmed the observations done in the striping test - the client CPU was the limiting factor on the throughput of the system and the storage nodes remained with a relatively low CPU usage (see Figures 3.26 and 3.27). In this case the processor occupancy of the brick nodes took values from 0% to 40% which are slightly above the values presented in the IOzone test using replication without SSH.

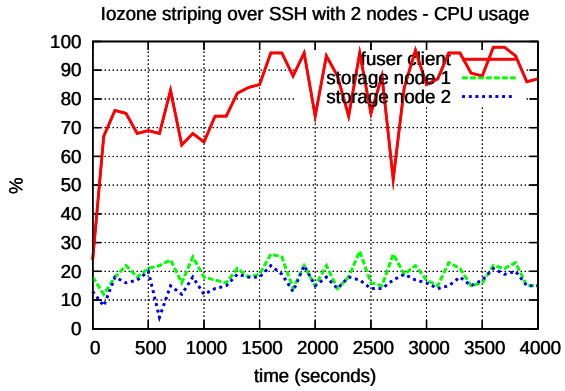

Figure 3.24: IOzone CPU usage with striping over SSH on 2 nodes

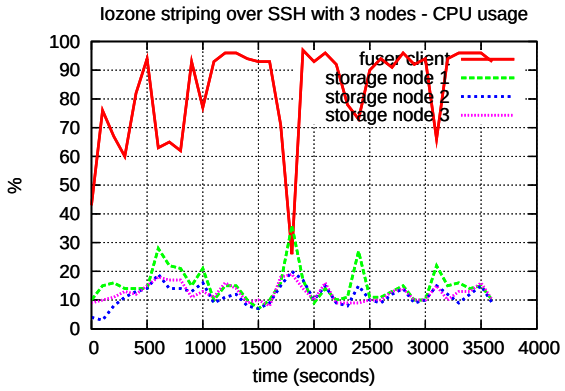

Figure 3.25: IOzone CPU usage with striping over SSH on 3 nodes

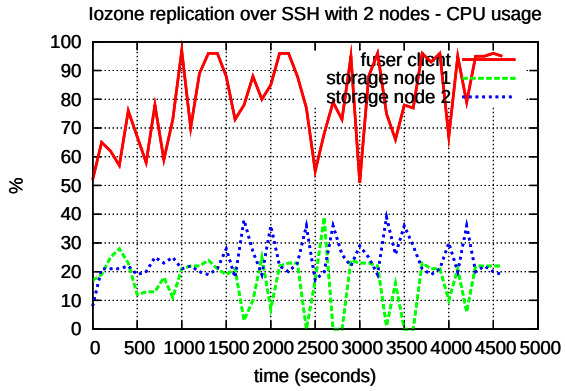

Figure 3.26: IOzone CPU usage with replication over SSH on 2 nodes

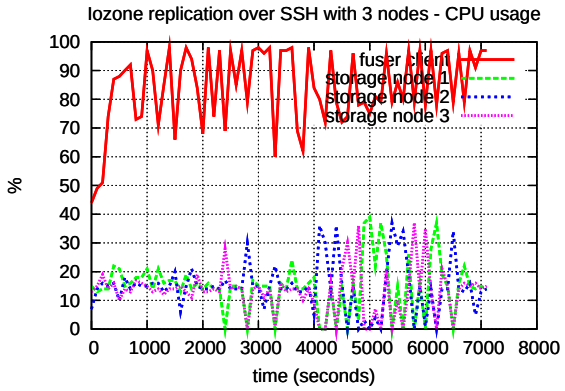

Figure 3.27: IOzone CPU usage with replication over SSH on 3 nodes

The values obtained related to the system's throughput (read and write transactions) reflect the bottleneck on the client. For higher values of file size (X-axis in the charts), the read rate is lower than 50000 Kbytes/sec. This happens with both replication and striping independently of the number of nodes. In the tests without SSH the observed throughput for the read transaction was around 80000 Kbytes/sec for the same scenarios. The write transactions without the encrypted tunnel had distinct behaviors depending on the storage scheme and the number of bricks. Using striping the output was above 50000 Kbytes/sec with two or three nodes whereas with replication the write throughput

was below 30000 Kbytes/sec and 20000 Kbytes/sec with two and three nodes respectively. On the other hand, using SSH, the write operations do not change significantly with the number of storage servers. In this case the measured throughput stays around 30000 Kbytes/sec using a stripe strategy and 15000 Kbytes/sec using a replication scheme.

For smaller values of file size there is a peak of throughput in all the tests involving SSH that must be an effect of CPU cache - when all the data can fit in the CPU cache the results show a very high efficiency. As the file size grows, the effectiveness of cache drops and therefore the output stabilizes in the values that correspond to the actual storage performance.

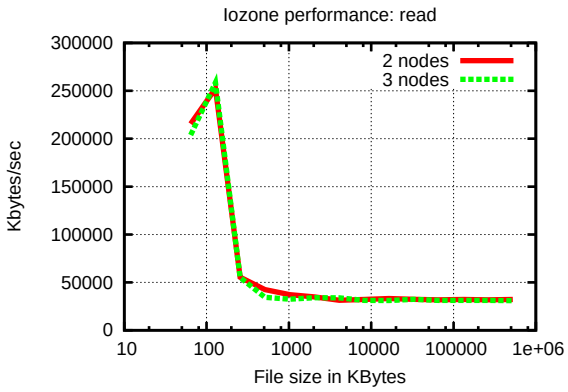

Figure 3.28: IOzone read performance with striping over SSH

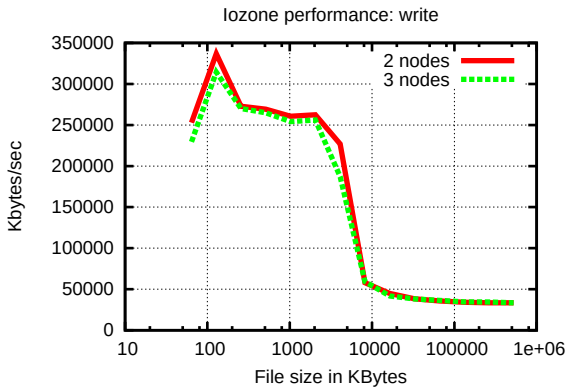

Figure 3.29: IOzone write performance with striping over SSH

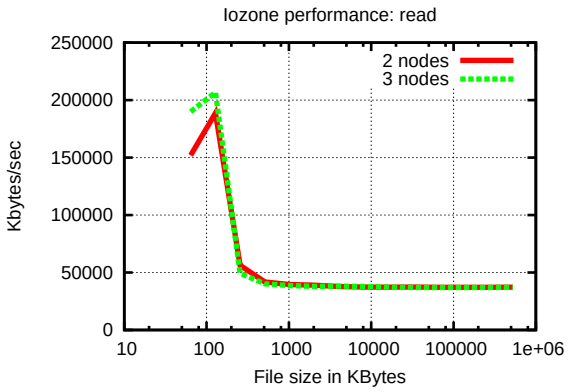

Figure 3.30: IOzone read performance with replication over SSH

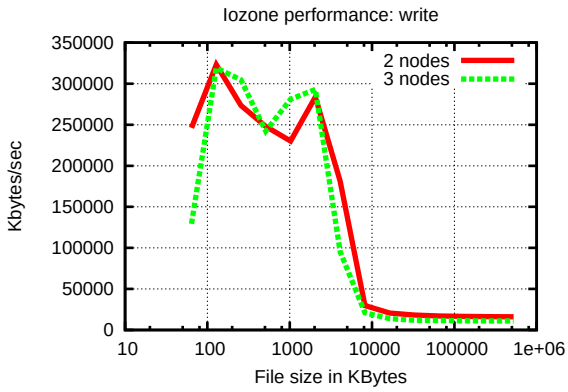

Figure 3.31: IOzone write performance with replication over SSH

3.5 Final remarks

The results obtained with the experiments described in this Chapter served two purposes. The most obvious was the attainment of measurements that allowed the quantification of the performance of GlusterFS under a specific set of scenarios. Those scenarios were chosen to test some of the capabilities of distributed file systems, namely replication of data to achieve higher availability levels and striping to provide for scalability in terms of storage capacity. The second purpose was more qualitative and thus harder to describe. It has to do with the experience of managing such a system from the point of view of a system administrator. For that matter the setup and configuration of the system for the sake of the tests that were performed is relatively simple, because it consisted of changing a single configuration file per node. Now, it should be noted that many aspects were left out of the tests because of time limitations - things like system tunning and different configuration parameters would have an impact on the results, that would have been interesting to assess.

With the results that came out of the tests it was possible to find that a solution using GlusterFS is able to meet the requisites for the application used by Portugal Telecom. It is an application with low throughput requirements (600KB/sec for writing at this time) which is well under the observed values in any test scenario. Furthermore, the system can have its storage capacity increased as needed, by adding new inexpensive storage nodes for the long-term storage needs of the application. For the security enhancement of the system, it was added an encryption layer provided by an SSH tunnel to protect the connections between client and servers. As a result, the throughput of the system generally decreased and it was found that with the SSH tunnel the client CPU becomes a performance bottleneck. That bottleneck happens in stress situations where the client is performing a high number of transactions in a short time frame. For Portugal Telecom's application that is not case, for the rate of arriving events follows a constant pace. Nevertheless it is important to notice that this can change in the future if other event sources are added to the SIEM, and so the solution using the SSH tunnel might not be able to handle an increase in the message rate.

In the next Chapter I present a summary of the work done in this project and show the conclusions that result from it.

Chapter 4

Conclusion and future work

This thesis evaluated different distributed file system for secure storing of event data in Portugal Telecom's application for complex event processing. The idea of using such a file system is to make the application capable of dealing with an increase in the data influx which, in turn, leads to more storage capacity requirements. After the survey on some open source parallel file systems, GlusterFS was chosen as a basis for the implementation of a prototype emulating one possible setup as to apply the file system on Portugal Telecom's SIEM.

4.1 Summary of work

The steady increase in CPU speed in computer systems has revealed a performance bottleneck in the I/O subsystem. Overcoming that restriction has been the target of several distributed file systems by leveraging parallelization and pushing boundaries in terms of storage capacity. Currently there are many distributed file systems with different design goals and design principles which difficult the process of deciding which fits best a specific case. The search for a solution that is efficient, scalable, cheap, easy to manage and secure is not a straightforward task and it can become quite uneasy.

In this project, I have looked into some open source distributed file systems with the objective of selecting one that meets the requirements of Portugal Telecom's ongoing SIEM application. That application in particular does not have high I/O requirements - at least for now - comparing to the most common applications of distributed file systems. However, a DFS can be the answer for other

objectives namely scalability, i.e., the capability of growing or shrinking the storage capacity as needed without having a significant performance penalty and taking advantage of cheaper hardware. The initial cost of the system plus the ongoing costs is an important factor in a corporate environment, oftentimes it is the deciding factor when deciding which way to go. Using other storage systems the norm is to use high-grade hardware that is associated with a high expenses which is justified by high performance and low error rate. Nevertheless, hardware failures happen anyway and so operating expensive storage systems becomes even more costly. The philosophy behind several of the analyzed distributed file systems is rather different. Instead of assuming the hardware is reliable, those systems assume that disks fail all the time and running a storage system should cope with it and not require high maintenance.

Throughout the analysis of the file systems it is become clear that there is a division in the way the applications are supposed to interact with it. Some file systems such as HDFS do not provide native POSIX-like interfaces because it makes it difficult for it to guarantee some crucial properties like consistency while at the same time enabling high performance. Other file systems do allow interacting with it using POSIX primitives, which makes it simpler to put already existing applications on top of it. This was the reason why I primarily tested GlusterFS.

For that matter, I ran some benchmark tests on GlusterFS so as to ascertain how does it behaves under different scenarios. It was possible to observe that Gluster is capable of accommodating the intended application because it performed accordingly. Moreover, the results of the experiments were useful to show that the setup of a storage system of this type is not as simple as plugging in the hardware - planning, testing and tunning is always required.

The final step of this project consisted of setting up a prototype to simulate the environment of Portugal Telecom's SIEM security events storing. This prototype allowed me to validate the capabilities of GlusterFS in terms of performance and security features. In the end I provided and extra layer of security to the prototype and executed some more benchmark tests, that helped to show that in order to achieve performance and security it should be made available natively instead of being an add-on.

The achieved performance numbers can be considered low, even though they are sufficient to accommodate the application. As such, using alternatives other than distributed file systems may also be considered. For instance, a simple NFS with enough storage disks using a software replication for reliability might suit better Portugal Telecom's needs in this case.

4.2 Limitations

During the course of this work I looked into some distributed storage solutions from the point of view of their design and architecture. Although the work was done in a focused way, with the intention of coming up with a solution for the problem in question, it has some limitations and shortcomings that stem from the time constraints. First of all the analysis was confined to some open source systems as it is easier to obtain documentation and technical reports that provide the relevant information related to its concepts and implementation aspects. Second, the performance tests were only done using GlusterFS. It would be better if it was done with more file systems because in that way the results would be richer in the extent that it would be possible to compare different systems with factual data. Testing more file systems would require a lot more time, not only for the tests alone but also for the eventual changes needed to make the file system work - different operating system versions could be mandatory or even modifications in the application.

Third, the setup for the experiments was bounded by the small number of servers. Using a higher number of servers in the tests would provide a better view of the way the systems behave given that the performance results can be different in that case. On the other hand, setting up more servers to test the system would necessarily take a larger amount of time and each change in the configuration is prone to errors which in turn would take even more time to track down and to correct.

Another limitation is related to file system optimizations, tunning, and changes in the configuration - doing it would yield different results. This work does not address it, as the tests were made using a single configuration for the different scenarios. It is clear that tunning distributed file systems by changing parameters such as cache and block size or write strategy leads to changes in the overall performance and therefore it would be better to measure this changes.

Finally, the tools used to perform the benchmarks try to cover a wide range of performance aspects and aim at stressing the file system in order to assess its limits. For the use case in question, the peak performance of the system is not the sole most important feature since the load is pretty much constant. So, things like stability and correctness should be also verified separately but these aspects are at some extent implicitly tested by the used tools.

4.3 Future work

Distributed file systems are a very broad topic and different applications have different requirements. Doing a thorough analysis of the available options calls for a wide range of tests including trying different configurations and tweaking options. For a deeper understanding of the performance and features of the distributed file systems it is necessary to run further tests. As such, analyzing more file systems beyond the ones mentioned in this work could lead to a different solution both in terms of architecture as well as performance.

Within the analyzed solution there are many possible modifications. That said, it is left for future work performing the analysis and tests on the proposed system using an increasing number of storage nodes as to find how does it compare with the analyzed system. Furthermore, it would be beneficial to experiment more combinations of settings and configuration parameters on the file system or even in the operating system itself - for instance, trying a different kernel, switching to 64 bits architecture, and so forth.

On a different level, the benchmark tools used for this project serve a specific purpose and so its results leave some room for improvement. Thus, trying other tools or other methodologies will certainly bring richer results and therefore lead to a better view of the analyzed object.

Bibliography

[1] POSIX.1-2008. The Open Group Base Specifications. Also published as IEEE Std 1003.1-2008, July 2008. URL http://dx.doi.org/10.1109/IEEESTD.2008.4694976. [Online; accessed 09-November-2011]. 2

[2] Inc. Gluster. An introduction to gluster architecture, 2011. URL http://www.gluster.com/products/gluster-file-system-architecture-white-paper/. [Online; accessed 19-May-2011]. 2.1

[3] S.M. Matyas, C.H. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Techn. Disclosure Bull.*, 27(10A):5658–5659, 1985. 2.1.1

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37:29–43, October 2003. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1165389.945450. URL http://doi.acm.org/10.1145/1165389.945450. [Online; accessed 22-May-2011]. 2.2

[5] Sage A. Weil. *CEPH: RELIABLE, SCALABLE, AND HIGH-PERFORMANCE DISTRIBUTED STORAGE*. PhD thesis, UNIVERSITY OF CALIFORNIA - SANTA CRUZ, December 2007. 2.3

[6] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt, and Ethan L. Miller. Dynamic metadata management for petabyte-scale file systems. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 4–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2153-3. doi: http://dx.doi.org/10.1109/SC.2004.22. URL http://dx.doi.org/10.1109/SC.2004.22. [Online; accessed 28-October-2011]. 2.3

[7] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16:133–169, May 1998. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/279227.279229. URL http://doi.acm.org/10.1145/279227.279229. [Online; accessed 02-November-2011]. 2.3.3

[8] Garth A. Gibson, David F. Nagle, William Courtright Ii, and Nat Lanza. Nasd scalable storage systems. In *In Proceedings of the USENIX '99 Extreme Linux Workshop*, 1999. 2.4

[9] Inc. Sun Microsystems. Lustre file system high-performance storage architecture and scalable cluster file syste white paper, December 2007. 2.4

[10] Yunhong Gu and Robert L. Grossman. Sector and sphere: the design and implementation of a high-performance data cloud. *Philosophical Transactions of the Royal Society A: Mathematical,Physical and Engineering Sciences*, 367(1897):2429–2445, 2009. doi: 10.1098/rsta.2009.0053. URL `http://rsta.royalsocietypublishing.org/content/367/1897/2429.abstract`. [Online; accessed 26-May-2011]. 2.5

[11] Felix Hupfeld, Toni Cortes, Björn Kolbeck, Erich Focht, Matthias Hess, Jesus Malo, Jonathan Marti, and Eugenio Cesario. The xtreemfs architecture a case for object-based file systems in grids. URL `http://www.xtreemfs.com/hupfeld_casefs_journal_online.pdf`. [Online; accessed 24-May-2011]. 2.6

[12] Felix Hupfeld, Björn Kolbeck, Jan Stender, Mikael Högqvist, Toni Cortes, Jonathan Marti, and Jesús Malo. Fatlease: scalable fault-tolerant lease negotiation with paxos. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 1–10, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-997-5. doi: http://doi.acm.org/10.1145/1383422.1383424. URL `http://doi.acm.org/10.1145/1383422.1383424`. [Online; accessed 24-May-2011]. 2.6.3

[13] Paul J. Leach and Dilip C. Naik. A common internet file system (cifs/1.0) protocol. Internet-draft, Network Working Group, 1997. URL `http://www.ubiqx.org/cifs/rfc-draft/draft-leach-cifs-v1-spec-02.txt`. [Online; accessed 23-August-2011]. 2.7

[14] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation or the sun network filesystem, 1985. 2.7

[15] M. Mesnier, G. Ganger, and E. Riedel. Object-based storage: pushing more functionality into storage. *Potentials, IEEE*, 24(2):31 – 34, april-may 2005. ISSN 0278-6648. doi: 10.1109/MP.2005.1462464. 2.7

[16] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. Measurement and analysis of large-scale network file system workloads. In *In Proceedings of the 2008 USENIX Annual Technical Conference*, 2008. 2.7

[17] Inc. Gluster. Performance in a gluster system, 2011. 3.3.2.2

[18] XtreemOS consortium. *The XtreemFS Developer Guide*. XtreemOS consortium, version 1.0 edition, 2009. URL http://www.xtreemfs.org. [Online; accessed 24-May-2011].

[19] Brian Bockelman. Using hadoop as a grid storage element. *Journal of Physics: Conference Series*, 180(1):012047, 2009. URL http://stacks.iop.org/1742-6596/180/i=1/a=012047. [Online; accessed 01-July-2011].

[20] P. Carns, R. Ross, and S. Lang. Object storage semantics for replicated concurrent-writer file systems. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1 –10, sept. 2010. doi: 10.1109/CLUSTERWKSP.2010.5613095.

[21] Michael Ewan. Exploring clustered parallel file systems and object storage, 2010. URL http://software.intel.com/en-us/articles/exploring-clustered-parallel-file-systems-and-object-storage/. [Online; accessed 25-May-2011].

[22] J. Jeffrey Hanson. An introduction to the hadoop distributed file system, February 2011. URL http://public.dhe.ibm.com/software/dw/web/wa-introhdfs/wa-introhdfs-pdf.pdf. [Online; accessed 15-June-2011].

[23] Jeffrey B. Layton. Ceph: The distributed file system creature from the object lagoon, April 2010. URL http://www.linux-mag.com/id/7744/. [Online; accessed 26-May-2011].

[24] Oracle. *Lustre File System Operations Manual - Version 1.8*. Oracle, revision 01 edition, December 2010. URL http://wiki.lustre.org/index.php/Lustre_Documentation. [Online; accessed 23-May-2011].

[25] B.C. Reed, M.A. Smith, and D. Diklic. Security considerations when designing a distributed file system using object storage devices. In *Security in Storage Workshop, 2002. Proceedings. First International IEEE*, pages 24 – 34, dec. 2002. doi: 10.1109/SISW.2002.1183507.

[26] Thomas M. Ruwart. Osd: A tutorial on object storage devices. In *19th IEEE Symposium on Mass Storage Systems and Technologies*, 2002.

[27] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, David, and C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39:447–459, 1990.

[28] Konstantin V. Schvachko. Hdfs scalability: the limits to growth. *;login: The USENIX Magazine*, 35(2), April 2010.

[29] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1 –10, may 2010. doi: 10.1109/MSST.2010.5496972.

[30] Jan Stender, Björn Kolbeck, Felix Hupfeld, Eugenio Cesario, Erich Focht, Matthias Hess, Jesús Malo, and Jonathan Martí. Striping without sacrifices: Maintaining posix semantics in a parallel file system. URL `http://www.xtreemfs.org/publications/striping-lasco-camera.pdf`. [Online; accessed 23-May-2011].

[31] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association. ISBN 1-931971-47-1. URL `http://portal.acm.org/citation.cfm?id=1298455.1298485`. [Online; accessed 10-June-2011].

[32] Clusters Brent Welch, Brent Welch, and Garth Gibson. Managing scalability in object storage systems for hpc linux. In *In Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 433–445, 2004. [Online; accessed 20-June-2011].