UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI INGEGNERIA

Corso di laurea in INGEGNERIA INFORMATICA

Tesi di Laurea in Computer Networks

Quality of Service of Cloud Distributed File Systems: an experimental approach

Candidato:
Leo Gioia

Relatore:
Chiar.mo Prof. Ing. Antonio Corradi

Correlatori:
Prof. Ing. Luca Foschini
Ing. Filippo Bosi
Dott. Ing. Stefano Monti

Anno Accademico 2016/2017
Sessione II

Abstract Italian Version

I sistemi informatici moderni vanno sempre più verso soluzioni cloud che permettono alle aziende di gestire in maniera più semplice e veloce i propri servizi. Il cloud, per poter garantire ai propri utilizzatori alti standard di Quality of Service, deve evolversi e modificare la propria struttura interna.

Un aspetto di notevole interesse è lo storage iperconvergente, ovvero la possibilità di riunire più sistemi di storage in un unico oggetto gestito da un apposito software capace di collassare il tutto in un'appliance gestito da un'interfaccia utente semplice. In un ambiente cloud ciò fornisce al fornitore la garanzia di avere un sistema di storage resistente ai guasti e più veloce.

Lo sviluppo di nuove tecnologie come i file system distribuiti porta i sistemi cloud esistenti ad abbandonare le vecchie modalità di gestione in quanto offrono numerose caratteristiche che prima andavano gestite manualmente. La replicazione dei file, ad esempio, viene gestita in modo automatico dai file system distribuiti e ciò garantisce tolleranza ai guasti.

In questo lavoro di tesi, viene fatta, inizialmente, una valutazione delle varie tipologie di file system che possono essere utilizzati in ambiente cloud. Successivamente ci si concentra su quattro differenti file system distribuiti: HDFS, Ceph, GlusterFS e XtreemFS. Per comprendere al meglio le potenzialità dei quattro file system distribuiti si conclude il lavoro con un benchmark personalizzato.

Abstract English Version

Modern IT systems are increasingly moving towards cloud solutions that allow companies to manage their services more quickly and easily. The cloud, in order to guarantee to its users high quality of service, must evolve and change its internal structure.

One aspect of considerable interest is the hyperconvergent storage, which is the possibility of combining multiple storage systems into a single object managed by a special software capable of collapsing everything into an appliance managed by a simple user interface. In a cloud environment, this provides the guarantee of having a fail-safe and faster storage system.

The development of new technologies such as distributed file systems leads existing cloud systems to change old storage systems as they offer many features that were previously handled manually. File replication, for example, is automatically managed by distributed file systems and this guarantees fault tolerance.

In this thesis, is made an evaluation of the various types of file systems that can be used in the cloud environment. Then we focus on four different distributed file systems: HDFS, Ceph, GlusterFS, and XtreemFS. To better understand the potentiality of the four distributed file systems, it’s done a personalized benchmark.

Summary

Introduction

In the last period, the diffusion of cloud technologies increased and more and more companies migrate their systems to cloud platforms. Cloud computing eases service provisioning without having to worry about maintaining the underlying infrastructure.

The infrastructure is not limited to a single server as in the traditional remote hosting model, but services are automatically and elastically deployed over multiple different servers. This higher distribution also ensures that if there is a failure of one host this will not cause a failure in the entire service. Focusing on computing and storage resources, they are not configured by the service provider directly, but assigned, quickly and conveniently, through automated procedures, starting from a pool of shared resources, thus greatly simplifying the user part of the configuration. In brief, with cloud technologies, service providers can manage their applications in a very simple way, by using an Internet browser.

In the Cloud scenario, it is always possible to elastically integrate additional resources by need. For instance, memory, disc space, and bandwidth can be added with few clicks and dismissed when no longer needed. Hence, the system is dynamically scalable and customizable according to the needs of the customer. In particular, these systems are known today as hyper convergent, which refers to an IT infrastructure with a software-based architecture that integrates computing, storage, networking and virtualization resources, making them available atop consumer hardware and with support provided by a single supplier.

This thesis, developed at Imola Informatica S.p.A. in Imola (Bo), focuses on cloud storage support infrastructures. Indeed, to work properly the cloud needs proper storage resources. The storage service itself is a complex one and consists of hardware devices, non-volatile storage media, infrastructures, and the service to manage it all. To

thoroughly analyze these technologies, we studied the behavior of a selection of main distributed file systems cloud solution available nowadays.

In a cloud environment, there is the need to achieve fault tolerance, reliability, availability, etc. Until now, this is reached through the use of file systems that did not offer support for these features and all storage resource management was at the expense of the cloud provider. Using a distributed file system simplifies this model by providing automatic management of storage resources and lowering the costs of managing the cloud provider.

In this thesis we will analyze the various types of distributed file systems present today and evaluate how they can support a cloud environment.

Chapter 1 talks about the storage in cloud computing. Here are shown in detail the differences between the various file system types and the common use cases of the file systems in a cloud environment.

In Chapter 2 there is a description of the State of the Art of local, network and distributed file system. Here are discussed the most common file systems and a general comparison is made.

In Chapter 3 are analyzed in depth four distributed file systems which are considered to be of greater interest to this work: HDFS, Ceph, GlusterFS, and XtreemFS. Here are presented the most important features of these file systems, such as the architecture, the deployment methods, the fault tolerance support, etc. and a specific comparison is made.

Chapter 4 is focused on distributed file system benchmark. Here are discussed the evaluation criteria to be taken into account for a proper analysis and the tools needed to make the benchmark. Finally, a comparative table of the various instruments is shown, which justifies the choice of used tools.

Chapter 5 shows the benchmark plane and the experimental results. Here is illustrated all the tests for a distributed file system and the results obtained by HDFS, Ceph, GlusterFS, and XtreemFS.

Chapter 1: Storage in Cloud Computing

During the last ten years, many things have changed in the computer world. The diffusion of broadband connection and the possibility to use virtualization and containerization have led to the spread of cloud technologies.

Cloud computing is a paradigm for delivering IT resources such as storage, processing or data transmission, characterized by availability on demand across the Internet from a set of pre-existing and configurable resources.

The resources are not fully configured and implemented by the vendor but are assigned, quickly and conveniently, through automated procedures, starting from a pool of shared resources with other users, leaving the user part of the configuration. When the user releases the resource, it is similarly reconfigured in the initial state and made available in the shared pool of resources.

By leveraging cloud computing technology, users who are connected to a cloud provider can do all these tasks, even with a simple internet browser. They can, for example, use remote software that is not directly installed on their computer and save data on online mass storage predisposed by the provider.

In Figure 1, below, is shown the logic diagram of a simple cloud computing architectures.

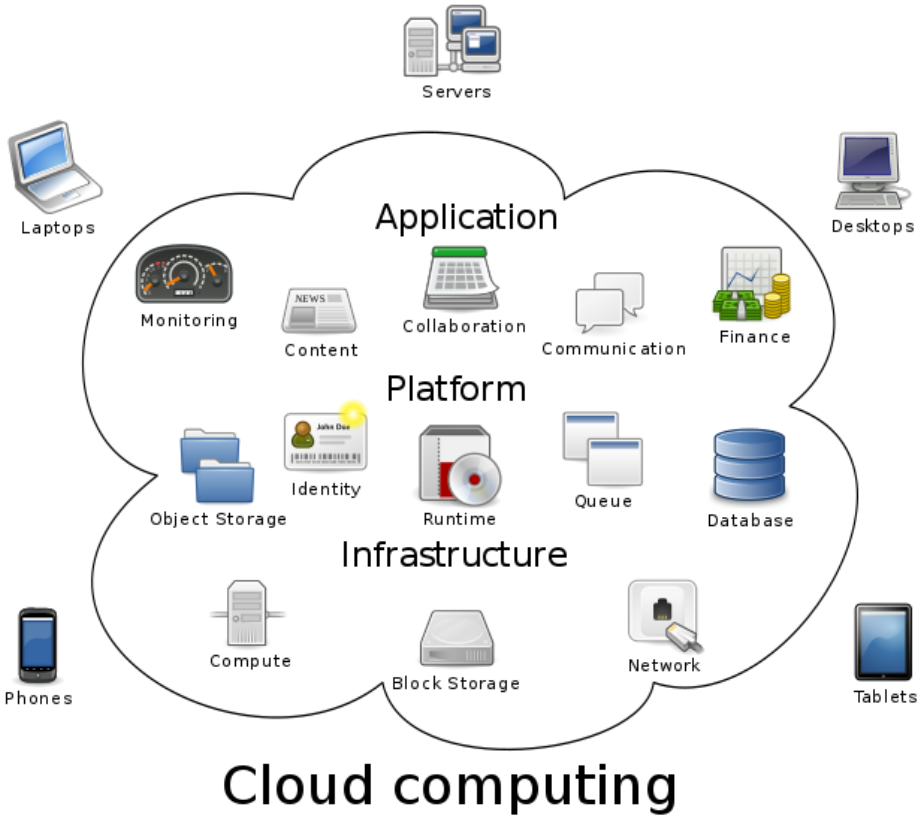

Figure 1: Logical Diagram of a Cloud Computing Network

Three basic types of cloud computing services can be distinguished:

- SaaS (Software as a Service): consists of the use of programs installed on a remote server, that is, out of the physical computer or local LAN, often through a web server;
- PaaS (Platform as a Service): instead of one or more individual programs, a software platform that can be made up of several services, programs, libraries, etc. is remotely executed. This

service is typical of some platforms used to develop other programs, such as Openshift, Cloud Foundry and PaaS services offered by AWS and Microsoft Azure;

- IaaS (Infrastructure as a Service): In addition to remote virtual resources, hardware resources such as servers, network capabilities, storage systems, archives, and backups are also available. The IaaS feature is that resources are instantiated on demand or demand when a platform needs it. For example, an OpenStack user can request the creation of a new virtual network when he needs it.

In the case of remote storage, the creation of a backup copy is automatic and the operation is transferred all over the Internet while the data is stored in server farms generally located in the home country of the service provider.

Cloud computing makes resources available to the user as if they were implemented by "standard" systems. The actual implementation of resources is not defined in detail; indeed, the idea is that implementation is a heterogeneous and distributed set of resources whose features are not known to the user.

This work focuses only on a part of the world of cloud computing: the storage. It represents hardware devices, storage media, infrastructures and non-volatile storage software of large amounts of electronic information. The concept of storage is tied to the concept of the file system.

File System [1], in a computer, is the way in which files are named and where they are placed logically for storage and retrieval. Typically, all the operating systems, such as DOS, Windows, Macintosh or Unix-based operating system, have file systems where files are stored somewhere in a hierarchical structure. A file is placed in a directory or subdirectory at the desired place in the tree structure.

File systems specify conventions for naming files, including the maximum number of characters in a name, which characters can be used and, in some systems, how long the file name suffix can be. A file system also includes a format for specifying the path to a file through the structure of directories.

Figure 2, below, shows the structure of a simple file system. In this case, there is folders, subfolders, and files.

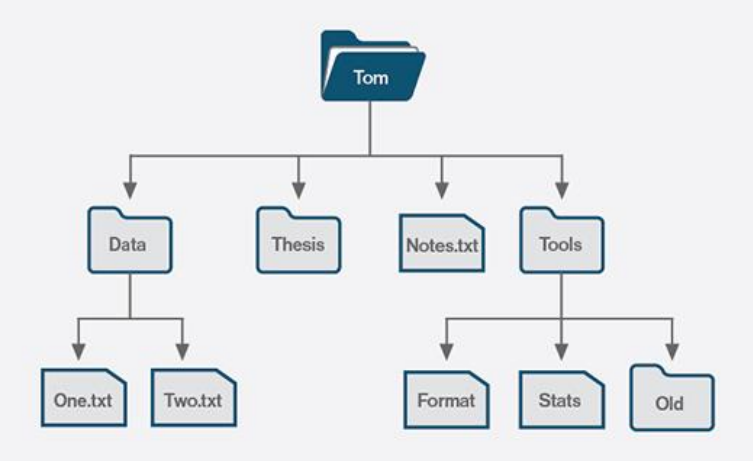

Figure 2: File System Structure

File systems use metadata to store and retrieve files and to have information about them. For example, every file must have information about creation date, modification date, file size, etc.

An important feature of a file system is the file access: file systems can restrict files access to a group of users. Encrypting files is a way to prevent user access. A key is applied to unencrypted text to encrypt it, or the key is used to decrypt encrypted text. Only users with the key can access the file.

There is a difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system where it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong types).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are superblock, inode, data block, directory block, and indirection block. The superblock contains information about the whole filesystem, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

As operating systems have matured, more and more features have been added to their file systems. Some examples of these improvements are more metadata, the ability to index files for faster searches, new storage designs, and more robust error-correction abilities. One of the biggest advances in file systems has been the addition of journaling, which keeps a log of changes that the computer is about to make to each file. This means that if the computer crashes or the power goes out halfway

through the file operation, it will be able to check the log and either finish or rollback the operation quickly without corrupting the file. This makes restarting the computer much faster, as the operating system doesn't have to scan the entire file system to find out if anything is out of sync.

All these continuous changes have led to the development of different file systems depending on the required features. The next paragraph explains some of the differences that characterize the various file system.

1.1 File System Types

Depending on how it is used, a file system can have different features. In personal computers, typically, we find a hard disk that can have some partitions; a local file system is mounted on these partitions, which is a file system that is heavily tied to the machine on which it is running and which cannot be accessed easily from the outside.

In other cases, however, there is a need to have access to a remote file system or to have a portion of this file system synchronized on your personal computer. In these cases, we need a distributed file system that can be either traditional. The most important feature of traditional and non-traditional distributed file systems is that distributed file systems need the network to work.

It is very important to understand the differences between the various file system types to understand the benefits of a distributed file system in a cloud environment. A Local File system is local to the machine in which it is running and in a cloud environment it cannot offer fault tolerance or striping support but is often used as a local support for machines where these features are not needed. A Traditional Distributed File System is generally used in environments where you do not have to sync the data on all clients and do not have high availability requirements because these file systems need the network to work properly. A Distributed File System, however, meets high availability requirements, fault tolerance, striping, etc. and is able to work even if there is no network connection because the changes will then be sent later.

In this work, we begin by describing the typologies of the various file systems and the state of the art of each of them, and then we describe more extensively some distributed file systems considered most significant.

In this section, we will provide a description of three file system types: local file systems, traditional distributed file systems, and distributed file systems.

1.1.1 Local File Systems

A local file system is generally used for saving arbitrary and unstructured data, it is a very general system that typically hosts on-top other systems such as relational databases. As for consistency, there are far less stringent policies than databases. Typically, a local file system does not provide data replication (which makes distributed file systems instead). As a result, it does not have all the advantages of replication such as fault tolerance or striping. Another thing that needs to be taken into account today is the ability to scale the system easily: in a local file system, for example, if you want to increase your storage capacity you have to scale vertically by increasing the capacity of the disk you are using.

A local file system enables applications to store and retrieve files on storage devices located locally on the machine. Files are placed in a hierarchical structure. The file system specifies naming conventions for files and the format for specifying the path to a file in the tree structure.

Each file system consists of one or more drivers and dynamic-link libraries that define the data formats and features of the file system. File systems can exist on many different types of storage devices, including hard disks, jukeboxes, removable optical disks, tape backup units, and memory cards.

All file systems have the following storage components:

- Volumes: a volume is a collection of directories and files;
- Directories: a directory is a hierarchical collection of directories and files;
- Files: a file is a logical grouping of related data.

1.1.2 Traditional Distributed File Systems

Traditional Distributed file systems are called traditional because of their frequent usage. Also, many new solutions are based on these systems. Some of the traditional DFS are commercial (like AFS), and others are free (OpenAFS, NFS, Coda, and Samba).

Reliability is a property which guarantees clients all functionality all the time when clients are connected to the system. In DFS it can be achieved by using several techniques and methods.

For communication between the DFS and a client, is usually used a computer network where several protocols can be used. These protocols can be connectionless or connection-oriented. In the mostly used TCP/IP model, UDP is a connectionless protocol and TCP is a connection-oriented protocol. For achieving reliability, DFS usually use connection-oriented protocols. Each of these protocols uses the port for a different application which the message will be delivered to. TCP and UDP provide the end-to-end connection.

The advantage of this solution is that the client knows almost immediately that a failure occurred and can attempt to set up a new connection. The disadvantage of using connection-oriented protocols is the slowness of these protocols in comparison to connectionless protocols that usually have the smallest overhead than connection-oriented protocols.

Preventing inconsistent state of the file system can be also done by choosing a suitable file system which will be used for storing data. File systems use journal techniques to prevent an inconsistent state. A journal technique or strategy describes when and for what will be the journal used for.

A journal technique has usually four steps:

1. Write a change in the file system into the journal;
2. Apply the change to the file system;
3. Write the end of the operation into the journal;
4. Clear the record from the journal.

The file system can be also damaged when the power supply is down. Preventing this state can be done by using an Uninterruptible Power Supply (UPS). UPS allows the nodes to save all critical data to prevent file system inconsistency in case of power shortage.

Once the communication is solved by using connection-oriented protocol, reliability is the next issue to be solved on the server side. Reliability on the server side in DFS means that the node is still serviceable while there is a fault. HDD crash can be prevented by using RAID. RAID is an acronym for Redundant Array of Inexpensive/Independent Disks. While using RAID, we prevent a failure by using the more hard disk.

Typically, in a distributed file system we need to have some guarantees to recover data after a disk failure. This feature can be achieved by replicating files. Replication in DFS means that the files are stored on several nodes in DFS. When one of the nodes crashes, the file is still available from the other node.

Replication can be done automatically or administratively (the administrator chooses what will be replicated, and a location for replicas). File replication can be done in an optimistic way (all replicas

are writable) or in a pessimistic way (the client can write only to a master replica; other replicas are read-only).

Performance in traditional DFS can be increased by using a file replication and a caching mechanism.

By using replication, choosing the file and the place for replication is very important. The file for replication should be read very often and should not be modified very often. Writing or updating a replicated file is an expensive operation. Choosing a place for a file replica is also very important. The server which is chosen to store the replica should not be overloaded and should have good network connectivity.

A cache in the computer system is a component which stores data that were frequently requested, hence can be potentially used in the future. When the cached data are requested, the response time is shorter than when the data are not in a cache and must be downloaded. The cache can be stored in RAM for fast access.

On the server side, the cache is usually located in RAM, on the client side, the cache can be located in RAM or on the hard disk. Client-side caching is also sometimes called client initiated replication. Client cache can also provide so-called offline cache. This means that the client can access files from the cache after disconnection from the server.

1.1.3 Distributed File Systems

Distributed file systems (DFS) does not directly serve to data processing. They allow users to store and share data. They also allow users to work with these data as simply as if the data were stored on the user's own computer.

Compared to a traditional client-server solution, where the data are stored on one server, important or frequently required data in DFS can be stored on several nodes (node means a computer operating in a DFS). This is called replication. Replication can be used for achieving better system performance and/or for achieving reliability of the system.

The data in a DFS are more protected from a node failure. If one or more nodes fail, other nodes can provide all functionality. This is known as availability and reliability. Availability is the holy grail of the cloud. It embodies the idea of anywhere and anytime access to services, tools and data and is the enabler of visions of a future with companies with no physical offices or of global companies with completely integrated and unified IT systems. Reliability is the ability of a computer-related hardware or software component to consistently perform according to its specifications. In theory, a reliable product is totally free of technical errors. In practice, vendors commonly express product reliability as a percentage.

Files can also be moved among nodes. This is typically invoked by an administrator and it is done for improving a load-balancing among nodes. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent. In DFS, this property is known as transparency.

If the capacity of the nodes is not enough for storing files, new nodes can be added to the existing DFS to increase DFS capacity. This property is known as scalability.

A client usually communicates with the DFS using LAN, which is not a secure environment. Clients must prove their identity, which can be done by authenticating themselves to an identity provider in the system. The data which flow between the client and the node must be resistant to attackers. Another important feature is encryption. It is the method by which plaintext or any other type of data is converted from a readable form to an encoded version that can only be decoded by another entity if they have access to a decryption key. Encryption is one of the most important methods for providing data security, especially for end-to-end protection of data transmitted across networks.

A journal in DFS can be used as a write-ahead commit log for changes to the file system that must remain unchanged. Every transaction made by a client is recorded in the journal, and the journal file is flushed and synchronized before the change is committed back to the client.

In distributed file system it is possible to use checkpoint that is a summarization of all records that were written to the journal and presents a consistent state of the DFS. When any fault appears, the DFS can recover by returning to this checkpoint by applying changes which are stored in the journal. Journal and checkpoint serve for maintaining metadata information. If journal records or/and checkpoint are missing or are corrupted, namespace information is lost. To prevent these incidents, critical information about namespace is stored on several storages.

In these file systems, it is possible to use dynamic file replication that uses statistical information about files and servers. Based on these statistics, it can be decided which files will be replicated and where.

Another way of storing file content is a DFS based on a P2P network [2]. An example of file storage is depicted in Figure 3. Files in this system are split into fragments, which are then stored on clients (peers). Links to these fragments are stored in a Distributed Hash Table (DHT). The DTH also maintains file and directory descriptors. Every client in this system is responsible for files, whose fragment keys are near the peer ID. A fragment key is made from a 160-bit random value and a combination of a pathname and a fragment number as a domain. The file descriptor is responsible for storing the fragment keys.

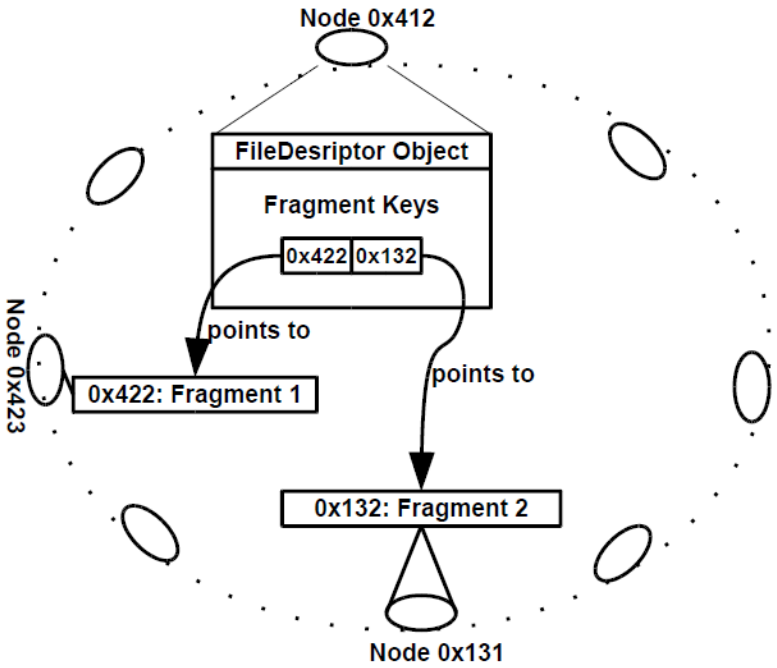

Figure 3: File Storage in P2P overlay

The advantage of using path name calculating hash is in updating file content. If new data is written to the fragments, there is no need to update DHT records. Fragments are still on the same nodes. DHT records must be updated only when the file is moved between

directories. In this circumstance, file fragments must be moved to new nodes according to the new calculated fragment keys.

While achieving reliability in DFS, data consistency is also very important. Data consistency can be achieved by using file locking. There are usually two types of locks: a shared lock for file reading and an exclusive lock for file writing. A shared lock is used when the data can be read simultaneously by many clients, but cannot be written simultaneously. While a file is being read by clients, other clients cannot write into this file. An exclusive lock means that only one client can access a locked file and can write file content.

Cache in the computer system is a component which stores data that can be potentially used in the future. The cache has limited capacity hence exist caching policies which try to predict future requests.

Caching policies need to mark the entity which can be removed from the cache when a new entity comes to the cache. Most of these algorithms are based on statistics made from previous data requests. These policies can be divided into basic policies and sophisticated policies. Basic policies don’t use any statistics, sophisticated policies use statistics.

1.2 Use Cases of Distributed File System

A distributed file system is a file system that allows storage of files in storage devices distributed over a network. This type of file system exceeds the limits of local file systems, where data storage was centrally made, making files available to remote users through a client-server interaction. Using a Distributed File System (DFS) ensures transparent and competing for file management, but does not neglect the security part by providing both authentication and encryption services.

Another point lies in the possibility of file replication on different nodes; thanks to this feature it is not only possible to provide support for parallel readings, but also to support for fault tolerance.

Below there are some of the most meaningful use cases of distributed file systems.

1.2.1 File Sharing

File sharing is one of the possible use cases of distributed file system. The idea behind this service is to share files within a network. Historically, file sharing has spread through the advent of broadband connection, which made it possible to share large files at relatively short times.

Initially, dedicated software such as Napster or Gnutella was used, which, thanks to the construction of an overlay network (a network of computers built at the application level), allowed the exchange of files between computers that did not belong to the same physical network. Napster and Gnutella encounter two different ways of sharing files over the network. Gnutella is an open source, decentralized and free service, with no specific guidelines, but it is not simple to scale up. Napster, on the other hand, is a centralized service, which despite its speed and big

investments has nevertheless been able to convince the industry of its importance. However, many file sharing systems have chosen a middle ground between the two extremes; an example is the kademilia network. But before legal problems arise, the various communities had already developed with OpenNap a viable alternative.

With the advent of distributed file systems, file sharing has been simplified. First, provider side it is possible to establish a sufficient number of replicas to provide high-speed download and fault tolerance support. In addition, the horizontal scalability offered by this technology guarantees that file system size can increase so you can accommodate large files. Secondly, but not least, one must consider the possibility of using client applications that allow automatic file system synchronization or a portion of it on client machines; this means that the experience of using them is facilitated and the rate of adoption increases.

1.2.2 Object Storage

Object Storage, such as Amazon S3, OpenStack Swift and Azure File is an object storage solution with a simple web service interface that allows you to archive and retrieve any amount of data anywhere on the Web. Clients using this type of storage are typically concerned native applications within the cloud as a target file system for backup and recovery features and to provide support for disaster recovery.

The transfer of large volumes of data to and from these file systems is also very simple, thanks to the data migration options within the cloud that many systems offer.

1.2.4 Log Concentration

A log concentration system deals with collecting log files from remote servers (even multi-platform) and saving them at one point. After saving the log files, they must be able to access them in a variety of ways: firstly, access through specific applications that handle log analysis should be accessible to provide statistics to the user; secondly, it should be possible to explore the files in a simple and intuitive manner even by a human being.

A distributed file system provides a distributed log concentration support. This means that you have a single 'logical' log data storage point, but you have support for all the QoS features common to distributed file systems. For example, you can replicate log files so that you can have faster access (thanks to the parallelization of the readings) and an automatic fault tolerance support so you can be sure you do not fall into situations where the data logs are lost.

1.2.5 Private Cloud

The use of cloud systems over the last decade has simplified the development, implementation, and maintenance of enterprise-level applications. The cloud, however, is still conceived today by many just as a public one, but also seen in terms of private cloud.

A private cloud provides cloud computing support with public cloud-like benefits, including scalability and self-service, using a proprietary architecture. Unlike public clouds that provide services to multiple organizations, a private cloud is generally devoted to a single organization. The advantage of using a private cloud is to see the ability to interact directly with architecture to control the different environments.

A distributed file system can offer private clouds strong support for shared storage. In other words, it is possible to share the data needed for the proper functioning of the cloud automatically between the various nodes. In this way, developers of cloud solutions can delegate the storage service completely to hyper convergent systems.

Chapter 2: State of the Art

In recent years the need to store large amounts of data has grown exponentially. Whatever type of data you collect, such as media files (images, audio, video, etc.) or academic files, you need to share them among users. Users also need to retrieve their data as quickly as possible. Files can be stored on a local file system or on a distributed file system. The local file system provides data quickly, but it does not have enough capacity to store a large amount of data. On the other hand, a distributed file system offers many advantages, such as reliability, scalability, security, capability, and so on.

A local file system is generally used for saving arbitrary and unstructured data, it is a very general system that typically hosts on-top other systems such as relational databases. A local file system does not provide data replication and striping.

Distributed file systems, on the other hand, offer a number of advantages over local ones such as the ability to replicate files or the ability to strip. Distributed storage systems provide automatic support to increase memory by scaling horizontally and not vertically.

Every distributed file system consists of several nodes: some of these nodes serve for storing file content and others serve for storing metadata about files. File content is usually stored on a local file system. Metadata are usually stored in the database. Every database record must also have a link to the file storage to the file content.

File content and metadata are usually created during file upload process. A user sends the whole file with its attributes to DFS. On the server side, file content and metadata are separated and sent to relevant storage. The whole upload process is depicted in Figure 4.

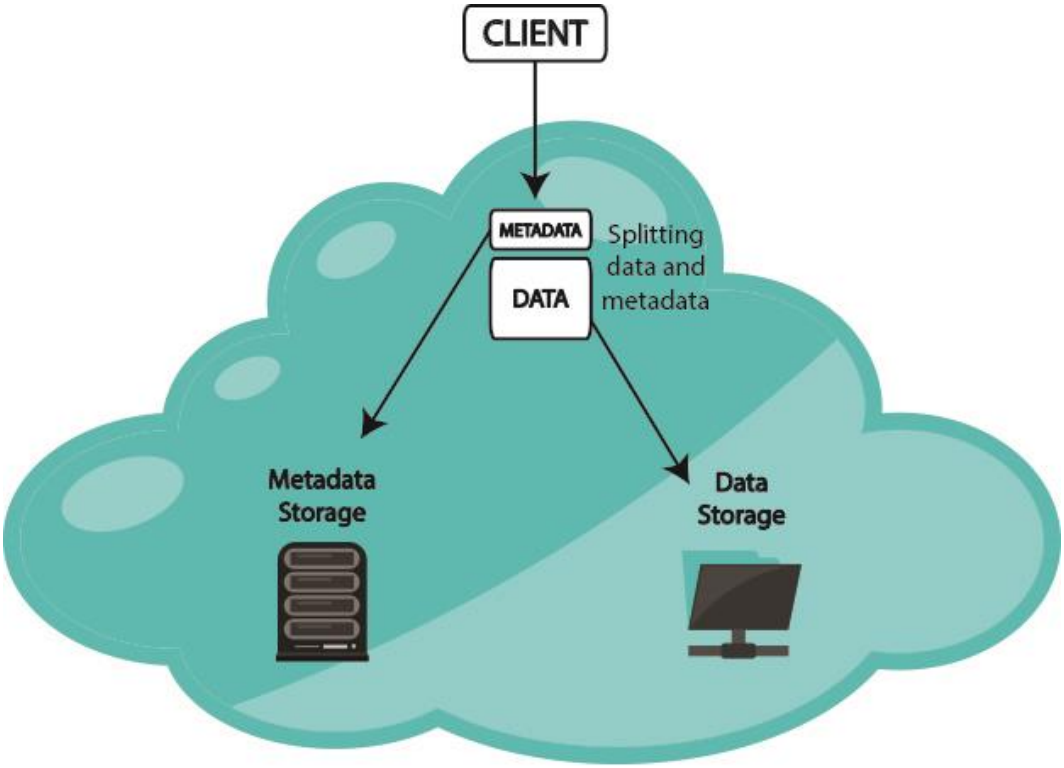

Figure 4: Upload Process

The following sections show, in details, the state of art of local file systems, traditional distributed file systems, and distributed file systems.

2.1 Local File Systems

A local file system enables applications to store and retrieve files on storage devices. Files are placed in a hierarchical structure. The file system specifies naming conventions for files and the format for specifying the path to a file in the tree structure.

Each file system consists of one or more drivers and dynamic-link libraries that define the data formats and features of the file system. File systems can exist on many different types of storage devices, including hard disks, jukeboxes, removable optical disks, tape backup units, and memory cards.

The most important point of this type of file system is that the files are not distributed and file system does not have features such as fault tolerance, replication, striping, etc.

Below there is a list of the most important local file system.

2.1.1 File Allocation Table (FAT)

A File Allocation Table (FAT) [3] is a table that an operating system maintains on a hard disk that provides a map of the clusters (the basic units of logical storage on a hard disk) that a file has been stored in. When you write a new file to a hard disk, the file is stored in one or more clusters that are not necessarily next to each other; they may be rather widely scattered over the disk. The operating system creates a FAT entry for the new file that records where each cluster is located and their sequential order. When you read a file, the operating system reassembles the file from clusters and places it as an entire file where you want to read it. For example, if this is a long Web page, it may very well be stored on more than one cluster on your hard disk.

Figure 5, below, shows how FAT operate to store different files.

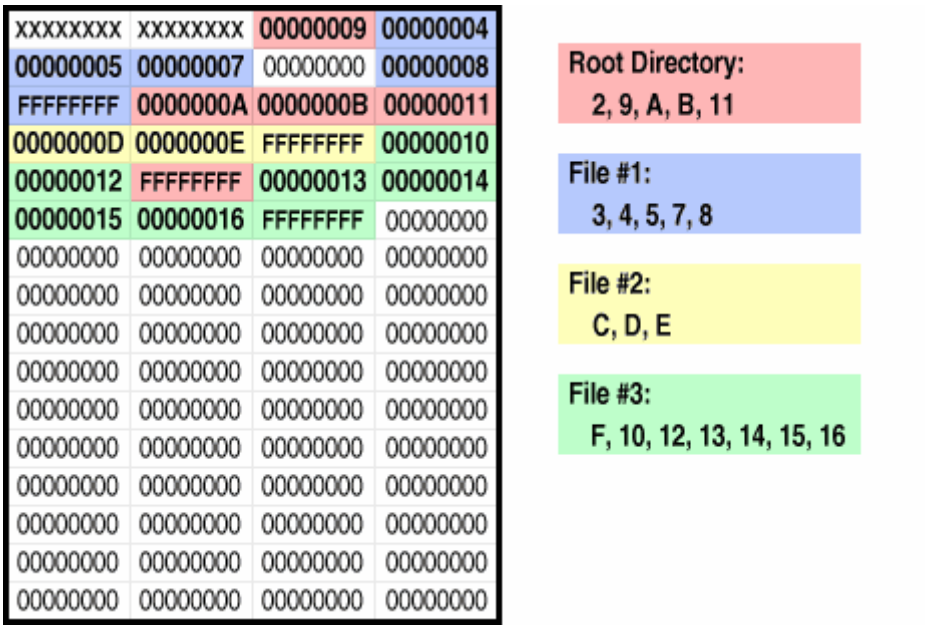

Figure 5: FAT Example

2.1.2 New Technology File System (NTFS)

NTFS (NT file system; sometimes New Technology File System) [4] is the file system that the Windows NT operating system uses for storing and retrieving files on a hard disk. NTFS offers a number of improvements over FAT in terms of performance, extensibility, and security. Notable features of NTFS include the use of a b-tree directory scheme to keep track of file clusters, support for very large files, an access control list (ACL) that lets a server administrator control who can access specific files, etc.

When a hard disk is formatted (initialized), it is divided into partitions or major divisions of the total physical hard disk space. Within each

partition, the operating system keeps track of all the files that are stored by that operating system. Each file is actually stored on the hard disk in one or more clusters or disk spaces of a predefined uniform size.

2.1.3 Extended Filesystem (EXT)

The extended filesystem has been part of the Linux kernel since 0.96c. With its developments through ext2, ext3, and ext4, it is one of the oldest Linux-specific software projects. The Linux kernel is now almost 22 years old. Its faithful companion since 1992 has been the family of extended filesystems. Below, there are the most important features of ext2, ext3, and ext4.

Ext2 is recommended on flash drives, USB drives because it doesn’t need to do the overhead of journaling. The main benefit of ext3 is that it allows journaling that has a dedicated area in the file system, where all the changes are tracked. When the system crashes, the possibility of file system corruption is less because of journaling. Ext4 supports huge individual file size and overall file system size. It is possible to mount an existing ext3 fs as ext4 fs (without having to upgrade it). Several other new features are introduced in ext4: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc. All you need to know is that these new features have improved the performance and reliability of the filesystem when compared to ext3. In ext4, you also have the option of turning the journaling feature “off”.

2.1.4 Oracle ZFS

Oracle ZFS [5] is a combined file system and logical volume manager designed by Sun Microsystems. The features of ZFS include protection against data corruption, support for high storage capacities, efficient data compression, integration of the concepts of filesystem and volume management, snapshots and copy-on-write clones, continuous integrity checking and automatic repair, RAID-Z and native NFSv4 ACLs.

ZFS is unusual, because unlike most other storage systems, it unifies both of these roles and acts as both the volume manager and the file system. Therefore, it has complete knowledge of both the physical disks and volumes (including their condition, status, their logical arrangement into volumes, and also of all the files stored on them). ZFS is designed to ensure (subject to suitable hardware) that data stored on disks cannot be lost due to physical error or misprocessing by the hardware or operating system, or bit rot events and data corruption which may happen over time, and its complete control of the storage system is used to ensure that every step, whether related to file management or disk management, is verified, confirmed, corrected if needed, and optimized, in a way that storage controller cards, and separate volume and file managers cannot achieve.

ZFS also includes a mechanism for snapshots and replication, including snapshot cloning. Very large numbers of snapshots can be taken, without degrading performance, allowing snapshots to be used prior to risky system operations and software changes, or an entire production file system to be fully snapshotted several times an hour, in order to mitigate data loss due to user error or malicious activity. Snapshots can be rolled back "live" or the file system at previous points in time viewed.

Oracle ZFS provides Oracle and third-party licensing as shown on the official documentation.

2.1.5 B-tree FS (BTRFS)

B-tree FS [6] is a modern copy on write (CoW) filesystem for Linux aimed at implementing advanced features while also focusing on fault tolerance, repair, and easy administration. Jointly developed at multiple companies, Btrfs is licensed under the GPL and open for contribution from anyone. Not too many companies have said that they are using Btrfs in production, but we welcome those who can say so on the production users page.

Major features currently implemented are: extent based file storage, space-efficient packing of small files, space-efficient indexed directories, dynamic inode allocation, writable snapshots, read-only snapshots, etc.

2.2 Traditional Distributed File Systems

Traditional Distributed file systems are called traditional not only for their frequent usage but also because these file systems need a network connection to work.

Below there is a list of the most important traditional distributed file system.

2.2.1 Network File System (NFS)

NFS (Network File System) [7] is an “rpc” protocol which was originally created by Sun Microsystems in 1985 and was designed for mounting disk partitions located on remote computers. NFS is based on RPC (Remote Procedure Call) and is supported in almost all operating systems. NFS was made for making client unaware of the location of their files and the remote file system can be mounted into a local directory structure. Clients can then work with their files as if the files were stored on their local file system. On the server side, NFS uses the same directory structure as is shown after mounting to the client.

To mount and unmounts file system client-side is used an automounter daemon. It also provides the ability to mount another file partition if the primary partition is not available at a given moment. List of replicas must be made before automounter daemon is run.

Figure 6 shows the Network File System architecture.

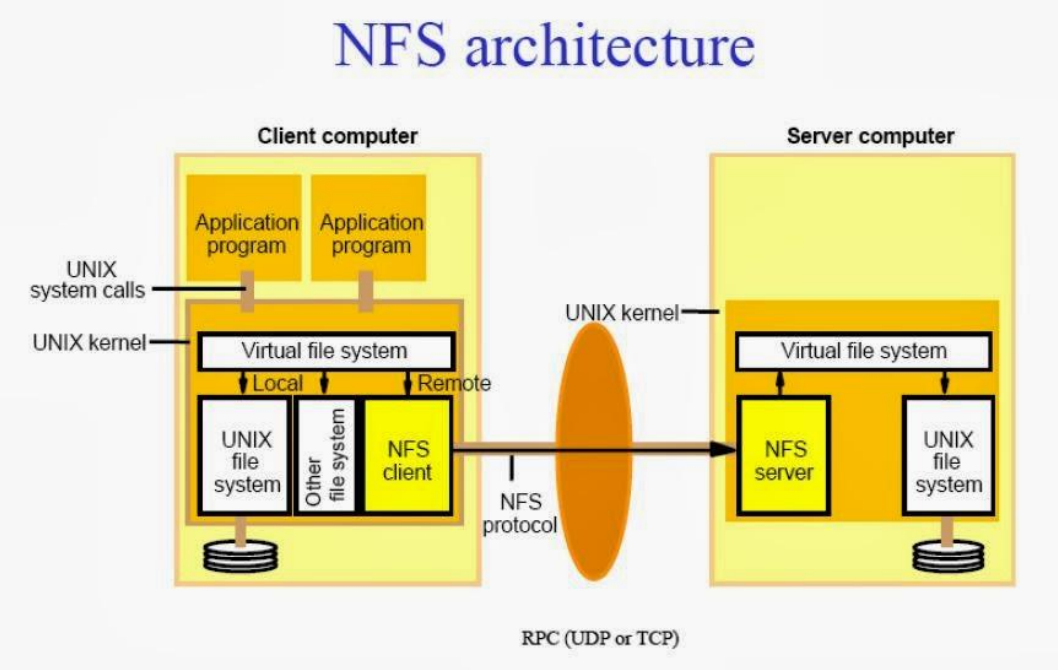

Figure 6: Network File System Architecture

2.2.2 Samba (SMB)

SMB [8] was developed in 1985 by IBM as a protocol for sharing files and printers. In 1998 Microsoft developed a new version of SMB called Common Internet File System (CIFS), which uses TCP/IP for communication.

SMB has been ported to other operating systems where the SMB is called samba. This system is stable, wide-spread and comfortable. It does not use local client-side caching. SMB uses the operating system's file access rights. SMB is wide used in WindowsTM operating systems.

2.2.3 Andrew File System (AFS)

AFS (Andrew File System) [9] was originally created at Carnegie Mellon University; later it became a commercial product supported by IBM. Now it is being developed under a public license (OpenAFS). The design goal of AFS was to create a system for large networks.

The smallest operating entity in AFS is the whole file which is the basic unit of data movement and storage. The whole file was chosen rather than some smaller unit such as physical or logical record.

AFS uses a uniform directory structure on every node. The root directory contains other directories which correspond to the cells. Cells usually represent several servers which are administratively and logically connected. One cell consists of one or more volumes. One volume represents a directory sub-structure, which usually belongs to one user. These volumes can be located on an AFS server. Volumes can be also moved from one AFS server to another. The movements of volumes do not influence the directory structure.

In AFS are used snapshots that are stored on replica servers and are usually made periodically. When the other servers are overloaded, the replica server provides files to the clients instead of these servers.

AFS uses pessimistic replication method. Pessimistic replication means that the replicas in AFS are read-only and present a snapshot of the system.

Figure 7 shows the Andrew File System architecture.

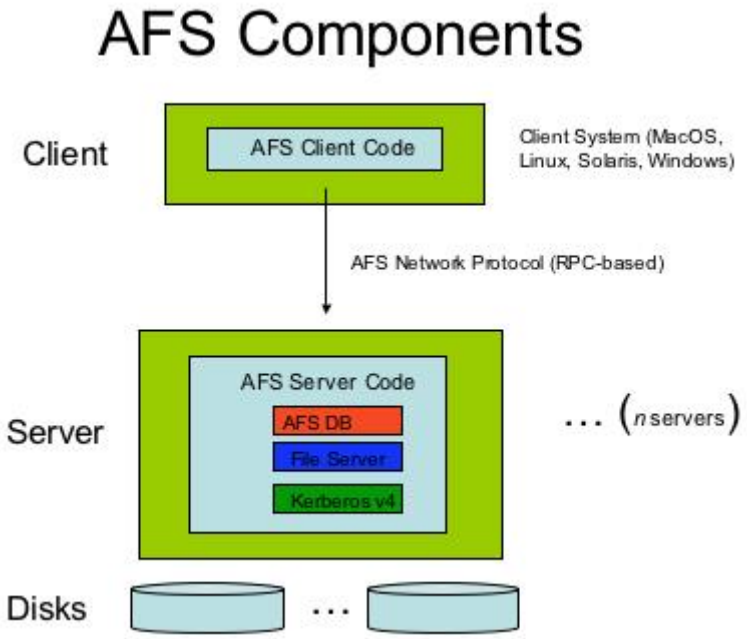

Figure 7: AFS Components

2.2.4 Coda

Coda [10] was developed at Carnegie Mellon University in 1990. It is based on the AFS idea and is implemented as a client and several servers. This system was mainly designed to achieve high availability. The client uses a local cache. Cached files can be used by clients even after disconnection from Coda server. This feature is called disconnected operation. While the client is disconnected from the server, all changes made to files are stored in a local cache. After reconnecting to the server, all these changes are propagated to the server. If any collision occurs, the user has to solve it manually.

The difference between Coda and AFS is in replication. Both of these systems use replication for achieving reliability. Coda uses optimistic replication; optimistic replication means that all replicas are writable. Client in Coda system must ensure file updating in all given replicas.

2.3 Distributed File Systems

Distributed file systems (DFS) allow users to store and share data. They also allow users to work with these data as simply as if the data were stored on the user's own computer.

Compared to a traditional DFS, data in can be stored on several nodes to achieve replication. Replication can be used for achieving better system performance and/or for achieving reliability of the system.

The data in a DFS are more protected from a node failure. If one or more nodes fail, other nodes are able to provide all functionality. Files can also be moved among nodes. If the capacity of the nodes is not enough for storing files, new nodes can be added to the existing DFS to increase DFS capacity.

Figure 8 shows a typical architecture of a distributed file system. It is important to remember that some elements may not be present in some DFS because for design choices they are implemented differently.

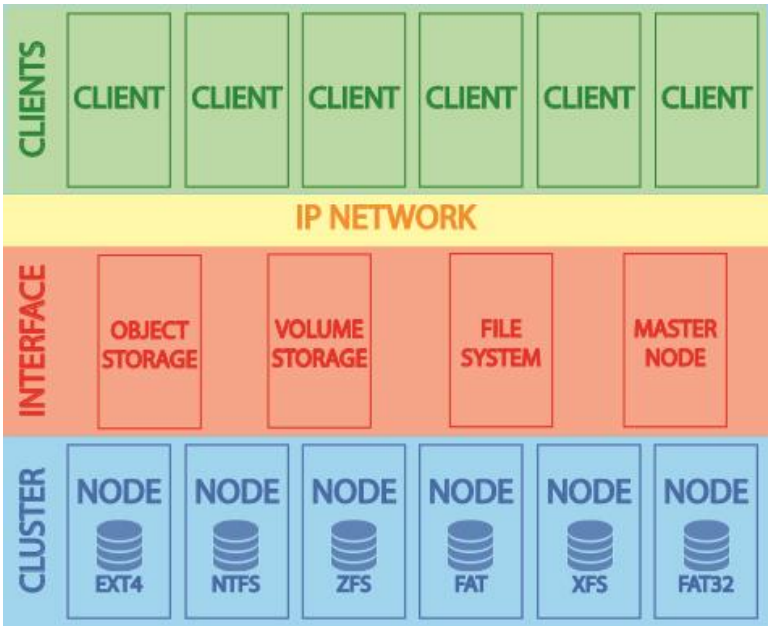

Figure 8: Distributed File Systems Levels

Below there is a list of the most important distributed file system.

2.3.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) [11] is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject.

HDFS is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications, coordinated by YARN. HDFS will just work under a variety of physical and systemic circumstances. By distributing storage and computation across many servers, the combined storage resource can grow linearly with demand while remaining economical at every amount of storage.

Figure 9 shows the Hadoop Distributed File System architecture.

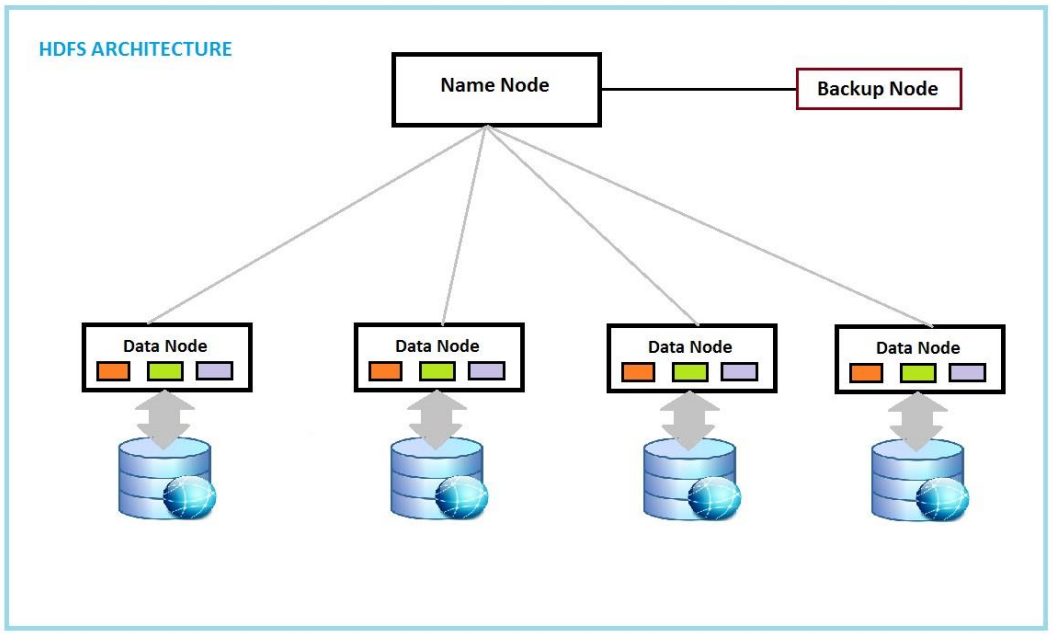

Figure 9: Hadoop Distributed File System Architecture

2.3.2 Ceph

Ceph [12] is a unified, distributed storage system designed for excellent performance, reliability, and scalability.

Ceph's software libraries provide client applications with direct access to the RADOS object-based storage system. It also provides some advanced features, such as RADOS Block Device (RBD), RADOS Gateway (RGW), and the Ceph File System (CephFS).

Ceph's object storage system isn't limited to native binding or RESTful APIs. You can mount Ceph as a thinly provisioned block device. When you write data to Ceph using a block device, Ceph automatically stripes

and replicates the data across the cluster. Ceph's RADOS Block Device (RBD) also integrates with Kernel Virtual Machines (KVMs), bringing Ceph's virtually unlimited storage to KVMs running on your Ceph clients.

Ceph's object storage system offers a significant feature compared to many object storage systems available today: Ceph provides a traditional file system interface with POSIX semantics. Object storage systems are a significant innovation, but they complement rather than replace traditional file systems. As storage requirements grow for legacy applications, organizations can configure their legacy applications to use the Ceph file system too! This means you can run one storage cluster for the object, block, and file-based data storage.

Figure 10 shows the Ceph architecture.

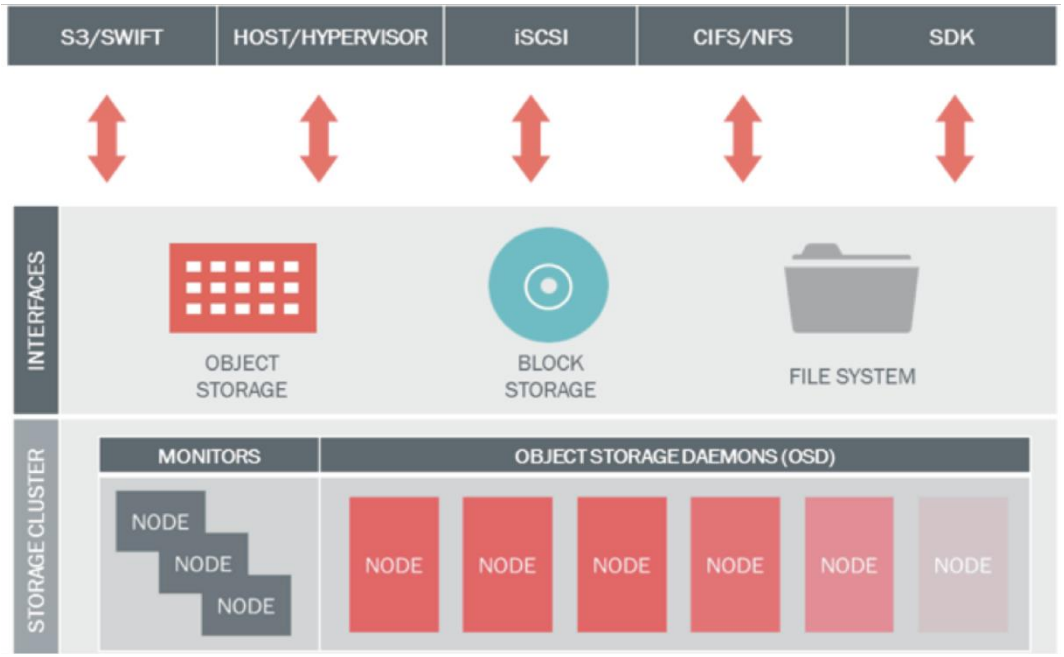

Figure 10: Ceph Architecture

2.3.3 GlusterFS

Gluster [13] is a distributed scale-out filesystem that allows rapid provisioning of additional storage based on your storage consumption needs. It incorporates automatic failover as a primary feature. All of this is accomplished without a centralized metadata server.

Gluster is an easy way to provision your own storage backend NAS using almost any hardware you choose.

It is possible to configure failover automatically so that if a server goes down, you don’t lose access to the data. When you fix the server that failed and brings it back online, you don’t have to do anything to get the data back except wait. In the meantime, the most current copy of your data keeps getting served from the node that was still running.

It is possible to build a clustered filesystem in a matter of minutes.

It takes advantage of what we refer to as “commodity hardware”, which means, It is possible to run on just about any hardware you can think of, from that stack of comm's and gigabit switches in the corner no one can figure out what to do with, to that dream array you were speccing out online.

It takes advantage of commodity software too. No need to mess with kernels or fine tune the OS to a tee. Glusterfs run on top of most Unix filesystems, with XFS and ext4 being the most popular choices.

Figure 11 shows the GlusterFS architecture.

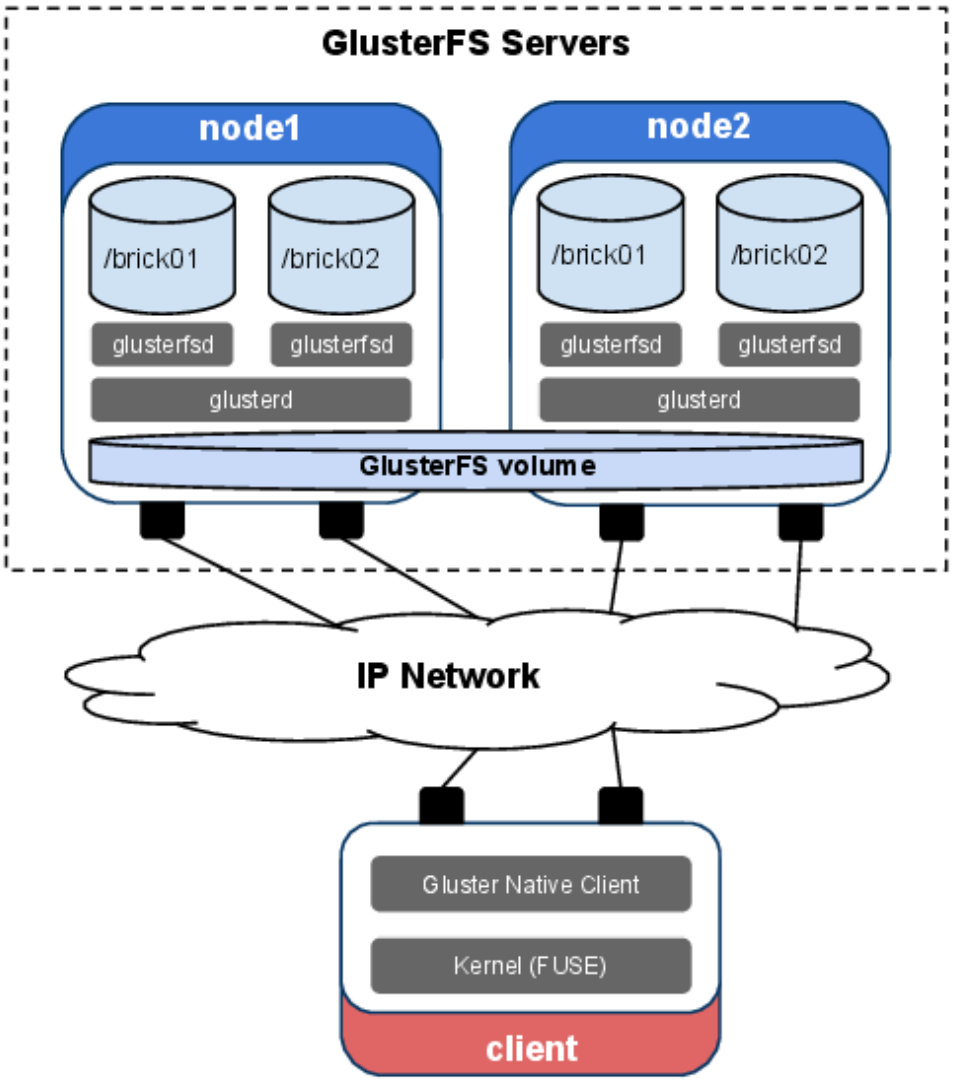

Figure 11: GlusterFS architecture

2.3.4 HekaFS

HekaFS [14] is intended to be a version of an existing distributed/parallel filesystem that's suitable for deployment by a provider as a permanent, shared service. It could also be used as infrastructure for hosting virtual-machine images, and in fact, the underlying technology (GlusterFS) is often used in this role currently. Users can already deploy this class of file system privately in the cloud, within their own virtual machines, but they pay both a performance and an administrative cost for doing so.

Specifically, HekaFS provides:

- Stronger authentication and authorization;
- Encryption, both on the wire and at rest;
- Multi-tenancy (isolating tenants' namespaces and IDs from one another);
- A management framework (CLI and web app) to configure the other features.

All of these features can be implemented in a modular way so that users can deploy only those they deem necessary or appropriate for their situation.

2.3.5 OrangeFS

OrangeFS [15] is a scale-out network file system designed for use on high-end computing (HEC) systems that provides very high-performance access to multi-server-based disk storage, in parallel. The OrangeFS server and client are user-level code, making them very easy to install and manage. OrangeFS has optimized support for parallel and distributed applications, and it is leveraged in production installations and used as a research platform for distributed and parallel storage.

OrangeFS is now part of the Linux kernel as of version 4.6. As this version of the kernel becomes widely available, it will simplify the use of parallel storage by Linux applications through OrangeFS.

The OrangeFS project has developed diverse methods of parallel access including Linux kernel integration, native Windows client, HCFS-compliant JNI interface to the Hadoop ecosystem of applications, WebDAV for native client access and direct POSIX-compatible libraries for pre-loading or linking.

2.3.6 GridFS

GridFS [16] is a specification for storing and retrieving files that exceed the BSON-document size limit of 16 MB.

Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB with the exception of the last chunk. The last chunk is only as large as necessary. Similarly, files that are no larger than the chunk size only have a final chunk, using only as much space as needed plus some additional metadata.

GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. The section GridFS Collections describes each collection in detail.

When you query GridFS for a file, the driver will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. You can also access information from arbitrary sections of files, such as to “skip” to the middle of a video or audio file.

GridFS is useful not only for storing files that exceed 16 MB but also for storing any files for which you want to access without having to load the entire file into memory. This is the reason for the division of the files into 16MB parts; in this way it is possible to load any portion of the file by skipping the previous blocks.

2.3.7 XtreemFS

XtreemFS [17] replicates your file data across multiple storage servers, which can be distributed worldwide. The replication algorithm is designed to cope with all the problems and failure scenarios that can occur in truly distributed systems: message loss, network partitions, server crashes etc. For each replicated file, XtreemFS keeps a list of replicas which gives you complete control where to place replicas. It is possible to use different placement schemes for each file. When a replicated file is opened, XtreemFS automatically elects a primary replica for that file which processes all file operations. If the primary fails, one of the backup replicas will automatically take over after a short failover period. The applications won't see a difference between regular and replicated file because the XtreemFS file replication is completely transparent.

In addition to regular file replication, XtreemFS provides read-only replication. This replication mode works on immutable files and

supports a large number of replicas. The read-only replication helps users quickly build a caching infrastructure on top of XtreemFS in order to reduce latency and bandwidth consumption between datacenters. In addition to full replicas that contain a complete copy, XtreemFS also supports partial replicas. These replicas are filled on demand when a client accesses data.

With asynchronous metadata server (MRC) backups, it is possible to easily create a consistent snapshot of metadata without interrupting operations. The MRC will create the backup in the background and continue to process client requests.

XtreemFS implements striping of files over several storage servers for increased I/O bandwidth and capacity. This is similar to RAID0. Parallel I/O ensures that read/write operations are executed in parallel on the storage servers.

XtreemFS supports versioned files and asynchronous metadata snapshots. Together these two techniques allow you to create snapshots of a volume instantaneously.

The XtreemFS client can cache metadata to speed up interactive use, e.g. from your shell. This feature is particularly useful if you mount volumes over the WAN or slow links like DSL.

2.3.8 Amazon Simple Storage Service

Amazon Simple Storage Service [18] is storage for the Internet. It is designed to make web-scale computing easier for developers.

Amazon S3 has a simple web RESTful interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses

to run its own global network of websites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

2.4 Systems comparison

In the previous section, we described the state of the art of the technologies in the three categories of file systems under consideration. We have chosen only the technologies that have been distinguished for adoption, for new technology introduced or for ease of use.

The study of these technologies has led us to understand that a cloud environment works better through a distributed file system because it can provide support for replication, fault tolerance, striping, etc. and can work even if there are network issues (which is not true in cases of traditional distributed file systems).

The thing to be understood is which of the previous distributed file systems will need to be thoroughly explored. The table in Figure 11 shows the comparison of the technologies described above.

For first comparison three parameters were taken into account:

- File System Typology: it is a first parameter that allows us to discard local and traditional distributed file systems;
- License: it has been chosen to take into account only open source technologies;
- Access API: a good file system has to offer the user many ways of accessing the data so that they can meet multiple needs.

From this comparison, some differences emerged that led to four technologies to be further explored: Hadoop Distributed File System, Ceph, GlusterFS, and XtreemFS.

Hadoop Distributed File System has been chosen for its ease of use. In addition, HDFS has been one of the first distributed file systems used in the enterprise environment that still maintains a comparison with modern systems.

Ceph, on the other hand, is a modern distributed file system that has been chosen not only for the many access possibilities it offers but also for many new introductions that will be discussed in the next chapter.

Finally, GlusterFS and XtreemFS have been chosen for their ease of use and for the many available access options. GlusterFS also has many possibilities for building volumes that will be discussed in the next chapter.

Name	Typology	License	Access API
File Allocation Table (FAT)	LFS	There are a series of patents for key parts of the FAT file system.	POSIX
New Technology File System (NTFS)	LFS	GNU General Public License	POSIX
B-tree FS (BTRFS)	LFS	BSD License	POSIX
Extended Filesystem (EXT)	LFS	Creative Commons Attribution-ShareAlike License	POSIX
Oracle ZFS	LFS	Oracle and third-party licensing	POSIX
Network File System (NFS)	TDFS	Creative Commons Attribution-ShareAlike License	POSIX
Samba (SMB)	TDFS	GNU General Public License	POSIX
Andrew File System (AFS)	TDFS	Creative Commons Attribution-ShareAlike License	POSIX
Coda	TDFS	Creative Commons Attribution-ShareAlike License	POSIX
Hadoop Distributed File System	DFS	Apache License 2.0	Java and C client, HTTP
Ceph	DFS	GNU Lesser General Public License	librados (C, C++, Python, Ruby), S3, Swift, FUSE

GlusterFS	DFS	GNU General Public License v3.0	libglusterfs, FUSE, NFS, SMB, Swift, libgfapi
HekaFS	DFS	GNU General Public License v3.0	POSIX API, etc.
OrangeFS	DFS	GNU General Public License	WebDAV and direct POSIX compatible libraries
GridFS	DFS	MIT License (MIT)	API
XtreemFS	DFS	BSD License	libxtreemfs (Java, C++), FUSE
Amazon Simple Storage Service	DFS and more	Royalty-free patent license	REST API, clients, etc.

Figure 12: State of the Art Comparison

Chapter 3: HDFS, Ceph, GlusterFS, and XtreemFS

The world of Distributed File Systems over the last decade has expanded considerably and more and more technologies are used in enterprise companies. We have already illustrated the differences between this type of file system against Local and Traditional Distributed File Systems and we showed the state of the art of the three categories. This allowed us to choose three technologies for a closer look.

Hadoop Distributed File System is a distributed file system developed by Apache to support the Hadoop data processing system and it is used for Big Data storage.

Ceph is a distributed file system that is widely spreading thanks to its ease of use and the features it offers (such as fault tolerance, striping, placement groups, etc.).

GlusterFS is a distributed file system among the most widely used and adopted. It offers a lot of features such as the various types of volumes that provide different types of storage to the users.

Ceph GlusterFS are widely used also because in the philosophy of hyperconvergent systems they have all the required features.

XtreemFS is a federated replicated object-based file system, it is designed to operate on a set of clusters that are connected by a wide area network. It runs on commodity hardware and doesn't require any specific.

To explore the features offered by HDFS, Ceph, GlusterFS, and XtreemFS, it was necessary to find comparative terms that could highlight the differences between them. The first phase of a study of other work was needed to understand the best methodology for approaching the comparison. From this, nine fundamental features of a distributed file system emerged which are briefly described below.

The first feature is the architecture of the Distributed File System. For a better understanding of the functioning of a technology, there is the need to start from architecture. Thanks to this, it was possible to go on with the study because the model on which the distributed file system was based was clear.

After architecture, we considered the deployment methods. These are an important point of comparison since there are systems that offer automated deployment capabilities with script files, others that provide dedicated installation tools, etc. This feature becomes important when you need to manage a large Distributed File System.

Consistency is another comparison term because a Distributed File System must be able to guarantee data consistency. In this part, you go deep into what a specific technology does to prevent data corruption.

The fourth point is one of the most important features of a Distributed File System: Replication. Usually, a DFS is chosen to cope with data loss and replication is critical to preventing it.

Despite what has been said, a fault in a cluster may still occur, so the analysis of fault tolerance behaviors remains a fundamental point of this in-depth study.

The sixth point of comparison is related to load balancing. You have to take this into account in two precise moments: when a node is down and when a new node is added as it may be necessary to rebalance data in the cluster.

Another important point is snapshot support. A user may be interested in wanting to restore a consistent state of his file system because of a fault, and snapshots are the best tool to make this possible.

The eighth point is accessibility. A Distributed File System must easily offer its users access via APIs or dedicated applications. Here you will want to show the various possibilities of use offered.

The last point we want to talk about is tied to a technology that is taking shape in recent years: container platform. Containerization refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances. Such instances, called containers, may look like real computers from the point of view of programs running in them. A computer program running on an ordinary computer's operating system can see all resources (connected devices, files, and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer. However, programs running inside a container can only see the container's contents and devices assigned to the container. A container problem is that for persistent storage they must be able to point to a portion of the host's storage. In this part, we want to analyze how the various Distributed Files System provide persistent storage support to containers.

3.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) [11] is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

An HDFS instance may consist of hundreds of server machines that store part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. The detection of faults and the quick and automatic recovery is very important.

Applications that run on HDFS need streaming access to their data sets. They are not general-purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access.

A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move closer to where the data is located.

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

3.1.1 Architecture

HDFS has a master/slave architecture as it is shown in Figure 13. An HDFS cluster consists of a single NameNode (a master server that manages the file system namespace and regulates access to files by clients). In addition, there are a number of DataNodes which manage storage attached to the nodes that they run on.

HDFS exposes a file system namespace and allows user data to be stored in files that are split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories and determines the mapping of blocks. The DataNodes are responsible for serving read and write requests from the file system's clients and perform block creation, deletion, and replication upon instruction from the NameNode.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and

repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

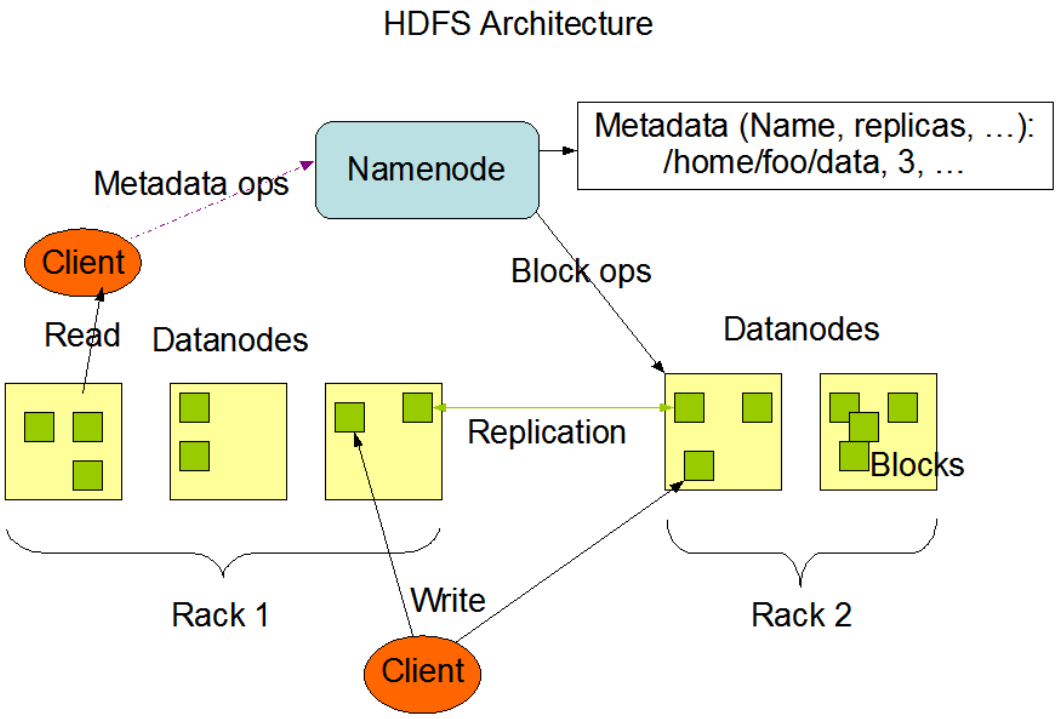

Figure 13: HDFS Architecture

HDFS supports a traditional hierarchical file organization: it is possible to create directories and store files inside these directories. The file system namespace hierarchy is similar to other file systems: the files can be created and removed or moved from one directory to another or renamed.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode.

It is possible to specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

3.1.2 Deployment Methods

The installation of HDFS is very simple and in order to be able to install it correctly, it is enough to follow the official guide. It should be noted that there are no automatic tools since, once downloaded the package from the official site, only a few parameters need to be configured to install it.

The first thing to do is enable ssh access without a password but through the key exchange mechanism. Next, it is necessary to install java on all machines that will host HDFS (it is recommended to install the Oracle version of java and not the open version).

The next step is downloading the desired version of HDFS from the official site and, once the download is complete, extract the contents of the downloaded package into the appropriate folder.

After extracting the folder, you need to modify the *core-site.xml* and *hdfs-site.xml* configuration files by adding the master URL and the desired replication factor on all cluster machines and adding the slave nodes to the *slaves* file of the master machine (or with IP or with logical names).

Finally, just run the dfs-start.sh script on master node to finish the installation.

3.1.3 Consistency

Consistency within a distributed file system must comply with some simple rules. All Distributed File Systems are generally Eventually Consistent, that means they can take time for changes to objects (during an operation of creation, deletion or updates) to become visible to all callers. There is no guarantee a change is immediately visible to the client which just made the change. For example, an object file1.txt may be overwritten with a new set of data, but when a get operation of file1.txt call is made shortly after the update, the original data returned.

HDFS assumes that filesystems are consistent; that creation, updates, and deletions are immediately visible, and that the results of listing a directory are current with respect to the files within that directory. Consistency is implemented with WORM and lease.

3.1.4 Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks (Figure 14); all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance and the block size and replication factor are configurable per file. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks and it periodically receives a heartbeat and a block report from each of the DataNodes in the cluster. Receipt of a heartbeat implies that the DataNode is functioning properly and a block report contains a list of all blocks on a DataNode.

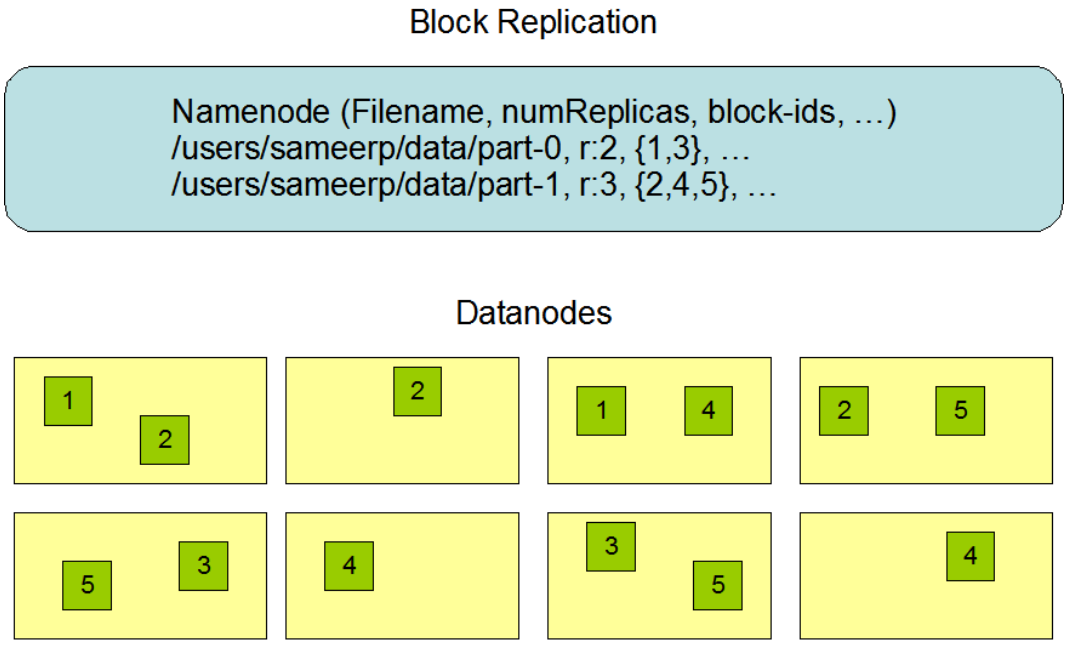

Figure 14: Block Replication

The placement of replicas is critical to HDFS reliability and performance. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. HDFS instances run on a cluster of computers that commonly spread across many racks. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. This prevents losing data when an entire rack fails and allows the use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance the load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

To minimize global bandwidth consumption and read latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request. If HDFS cluster spans multiple data centers, then a replica that is resident in the local data center is preferred over any remote replica.

3.1.5 Fault Tolerance

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures, and network partitions.

In case of disk failure or network partitions (that can cause a subset of DataNodes to lose connectivity with the NameNode), we must take into account that each DataNode sends a heartbeat message to the NameNode periodically. The NameNode detects this condition by the absence of a heartbeat message and marks DataNodes without recent heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary.

The necessity for re-replication may arise due to many reasons:

- a DataNode may become unavailable;
- a replica may become corrupted;
- a hard disk on a DataNode may fail;
- the replication factor of a file may be increased.

3.1.6 Load Balancing

HDFS defines the utilization of a server as the ratio of the space used to the total capacity of the server and fixes a threshold value in the range of (0,1). It decides that a cluster is balanced if for each datanode the usage of the server is different from the usage of the cluster by no more than the threshold value. In HDFS, if a node is unbalanced, replicas are moved from it to another one respecting the placement policy.

3.1.7 Snapshots Support

HDFS Snapshots are read-only point-in-time copies of the file system. Snapshots can be taken on a subtree of the file system or the entire file system. Some common use cases of snapshots are data backup, protection against user errors and disaster recovery.

Additional memory is used only when modifications are made relative to a snapshot. Blocks in DataNodes are not copied, the snapshot files record the block list and the file size. There is no data copying.

Snapshots do not adversely affect regular HDFS operations: modifications are recorded in reverse chronological order so that the current data can be accessed directly. The snapshot data is computed by subtracting the modifications from the current data.

Snapshots can be taken in any directory once the directory has been set to snapshottable. Administrators may set any directory to be snapshottable. If there are snapshots in a snapshottable directory, the directory can be neither deleted nor renamed before all the snapshots are deleted.

3.1.8 Accessibility

HDFS provides a code library that allows users to read, write and delete files and create and delete directories. Users access a file using its path in the namespace and contact the NameNode to know where to retrieve or put files' blocks, and then request the transfer by communicating directly with the DataNodes. This is done in a transparent way using some API in C, Java or REST.

HDFS allows user data to be organized in the form of files and directories. The syntax of this command set is similar to bash commands. Tables below show some example of these commands.

ACTION	COMMAND
CREATE A DIRECTORY NAMED /FOODIR	`bin/hadoop dfs -mkdir /foodir`
REMOVE A DIRECTORY NAMED /FOODIR	`bin/hadoop dfs -rmr /foodir`
VIEW THE CONTENTS OF A FILE NAMED /FOODIR/MYFILE.TXT	`bin/hadoop dfs -cat /foodir/myfile.txt`

HDFS, also, uses a web server to expose the HDFS namespace through a configurable TCP port. This allows a user to navigate the HDFS namespace and view the contents of its files using a web browser.

3.1.9 Persistence Storage Support in Containers Environment

When we talk about persistent storage support in containers, we need to consider the storage mechanisms that tools such as docker make available (typically, you can mount a portion of the host file system inside the container).

For HDFS, therefore, the problem moves to make your storage available in a portion of the host's file system. The only way to do this is by using additional tools that, through the use of APIs, keep a folder with HDFS synchronized.

3.2 Ceph

Ceph [12] uniquely delivers object, block, and file storage in one unified system. Ceph is highly reliable, easy to manage, and free. Ceph delivers extraordinary scalability–thousands of clients accessing petabytes to exabytes of data. A Ceph Node leverages commodity hardware and intelligent daemons and a Ceph Storage Cluster accommodates large numbers of nodes, which communicate with each other to replicate and redistribute data dynamically.

3.2.1 Architecture

Ceph provides a Storage Cluster based on Reliable Autonomic Distributed Object Store (RADOS). A Ceph Storage Cluster consists of two types of daemons: Monitor and Object Storage Daemon (OSD).

A Ceph Monitor maintains a master copy of the cluster map, for this reason, multiple instances of Ceph monitors ensure high availability if a monitor daemon fails. Clients retrieve a copy of the cluster map from the Ceph Monitor.

A Ceph OSD Daemon checks its own state and the state of other OSDs and reports back to monitors. OSDs are in charge of storage.

Most of traditional file systems have a single point of failure and some limits on performances and scalability because clients talk to a centralized component, which acts as a single point of entry. Ceph eliminates the centralization to enable clients to interact with Ceph OSD Daemons directly and also uses a cluster of monitors to ensure high availability. To eliminate centralization, Ceph uses an algorithm called CRUSH that provides a better data management mechanism compared to older approaches and enables massive scale by cleanly distributing the work to all the clients and OSD daemons in the cluster. The CRUSH

algorithm distributes data objects among storage devices according to a per-device weight value, approximating a uniform probability distribution. The distribution is controlled by a hierarchical cluster map representing the available storage resources and composed of the logical elements from which it is built.

The Ceph Storage Cluster receives data from Ceph Clients–whether it comes through a Ceph Block Device, Ceph Object Storage, the Ceph Filesystem or a custom implementation you create using librados–and it stores the data as objects. Each object corresponds to a file in a filesystem, which is stored on an Object Storage Device. Ceph OSD Daemons handle the read/write operations on the storage disks. Figure 15 shows this process.

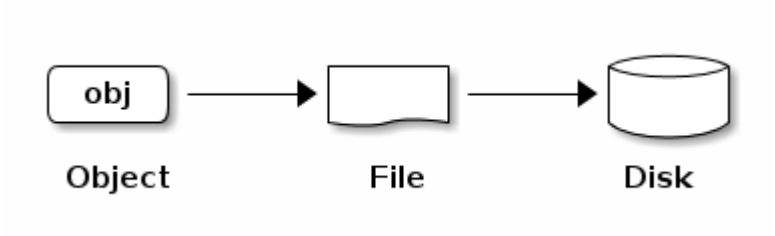

Figure 15: Ceph Storage Process

Ceph OSD Daemons store all data as objects in a flat namespace. An object has an identifier, binary data, and metadata consisting of a set of name/value pairs. The semantics are completely up to Ceph Clients. In Figure 16 there is an example, CephFS uses metadata to store file attributes such as the file owner, created date, last modified date, and so forth.

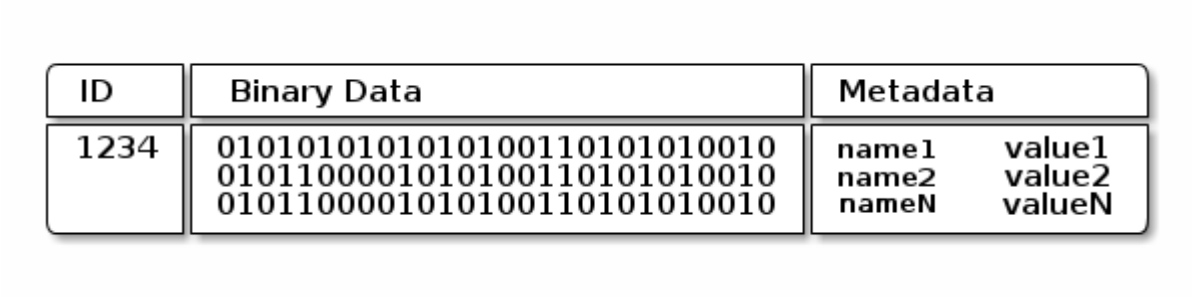

Figure 16: Ceph Metadata

3.2.2 Deployment Methods

Ceph is a distributed file system that provides an automated deployment tool called ceph-deploy.

The ceph-deploy tool is a way to deploy Ceph relying only upon SSH access to the servers, sudo privileges, and some Python packages. It runs on a workstation and does not require additional servers or tools.

The ceph-deploy tool was designed to get Ceph up and running quickly with sensible initial configuration settings. Users who want fine-control over security settings, partitions or directory locations should use a tool such as Juju, Puppet, Chef or Crowbar.

With ceph-deploy, it is possible to develop scripts to install Ceph packages on remote hosts, create a cluster, add monitors, gather (or forget) keys, add OSDs and metadata servers, configure admin hosts, and tear down the clusters.

3.2.3 Consistency

As part of maintaining data consistency and cleanliness, Ceph OSDs can also scrub objects. Ceph OSDs can compare object metadata with its replicas stored in other OSDs to catch eventually filesystem errors. OSDs can also perform deeper scrubbing by comparing data in objects bit-for-bit to find bad sectors on a disk.

In Ceph, reads must reflect an up-to-date version of data and writes must reflect a particular order of occurrences. It allows concurrent read and writes accesses using a simple locking mechanism that gives users the ability to read, read and cache, write or write and buffer data. Ceph also provides some tools to relax consistency, for example, it is possible to allow users to read a file even if it is currently rewritten.

3.2.4 Replication

When a client writes data, Ceph transforms them into objects that are put on different Placement Groups (PGs) and then stored on OSDs. Thanks to the CRUSH algorithm it is possible to choose the PGs and OSDs according to free disk space and other policies. Ceph implements three synchronous replication strategies:

1. Primary-Copy Replication: in primary-copy replication, the first OSD in the PG forwards the writes to the other OSDs and once the latter has sent an acknowledgment, it applies its writes, then reads are allowed;
2. Chain Replication: in chain replication, writes are applied sequentially and reads are allowed once the last replication on the last OSD have been made;
3. Splay Replication: in splay replication, half of the number of replicas are written sequentially and then in parallel. Reads are permitted once all OSDs have applied the write.

3.2.5 Fault Tolerance

A Ceph Cluster can operate with a single monitor, but this introduces a single point of failure.

To achieve reliability and fault tolerance, Ceph supports a cluster of monitors. In a cluster of monitors, latency and other faults can cause one or more monitors to fall behind the current state of the cluster. For this reason, Ceph must have agreement among various monitor instances regarding the state of the cluster. Ceph always uses a majority of monitors (e.g., 1, 2:3, 3:5, 4:6, etc.) and the Paxos algorithm to establish a consensus among the monitors about the current state of the cluster. Paxos is a family of protocols for solving consensus in a network of unreliable processors. Consensus is the process of agreeing on one result among a group of participants. This problem becomes difficult when the participants or their communication medium may experience failures.

To detect failures, OSDs periodically exchange heartbeats. Ceph provides a small cluster of monitors which holds a copy of the cluster map. Each OSD can send a failure report to any monitor in the cluster and the failed OSD will be marked as down. Each OSD can also query an up-to-date version of the cluster map and an OSD can join the system again after repairing.

3.2.6 Load Balancing

The Ceph storage system supports the notion of ‘Pools’, which are logical partitions for storing objects. Ceph Clients retrieve a Cluster Map from a Ceph Monitor and write objects to pools. Each pool has a number of placement groups (PGs) and CRUSH maps PGs to OSDs dynamically.

Mapping objects to placement groups (Figure 17) create a layer of indirection between the Ceph OSD Daemon and the Ceph Client because Ceph can rebalance where it stores objects dynamically. If the client knew which OSD Daemon had which object, that would create a tight coupling between the client and the OSD Daemon. CRUSH algorithm maps each object to a placement group and then maps each placement group to one or more OSD Daemons. This layer of indirection allows Ceph to rebalance dynamically when new OSD Daemons and the underlying OSD devices come online.

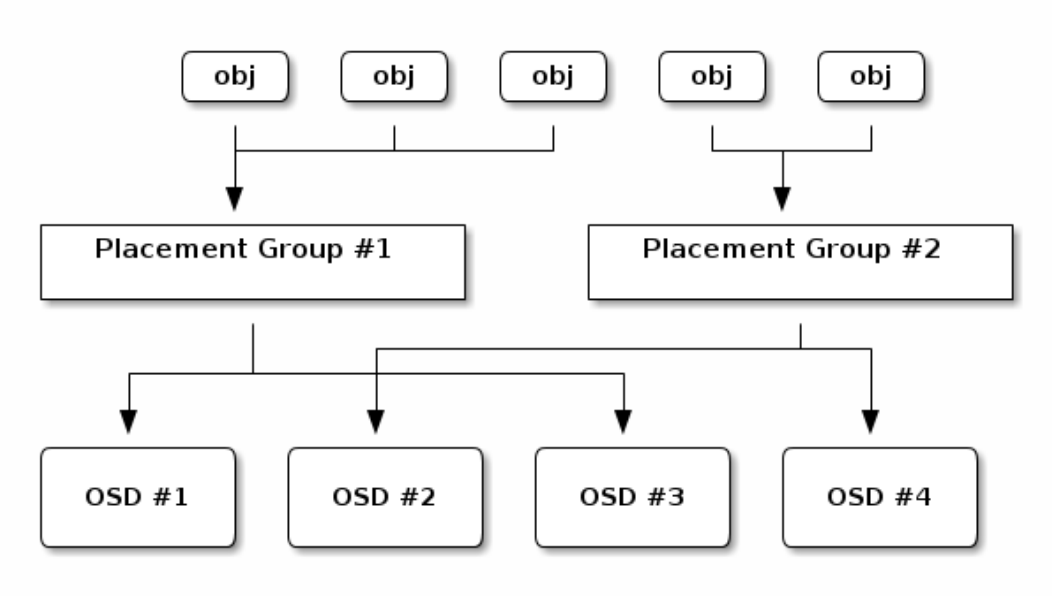

Figure 17: Objects Mapping in Placement Groups

When an OSD Daemon is added to a Ceph Storage Cluster, it changes object placement. The diagram in Figure 18 depicts the rebalancing process where some, but not all of the PGs migrate from existing OSDs to the new OSD. Many of the placement groups remain in their original configuration, and each OSD gets some added capacity, so there are no load spikes on the new OSD after rebalancing is complete.

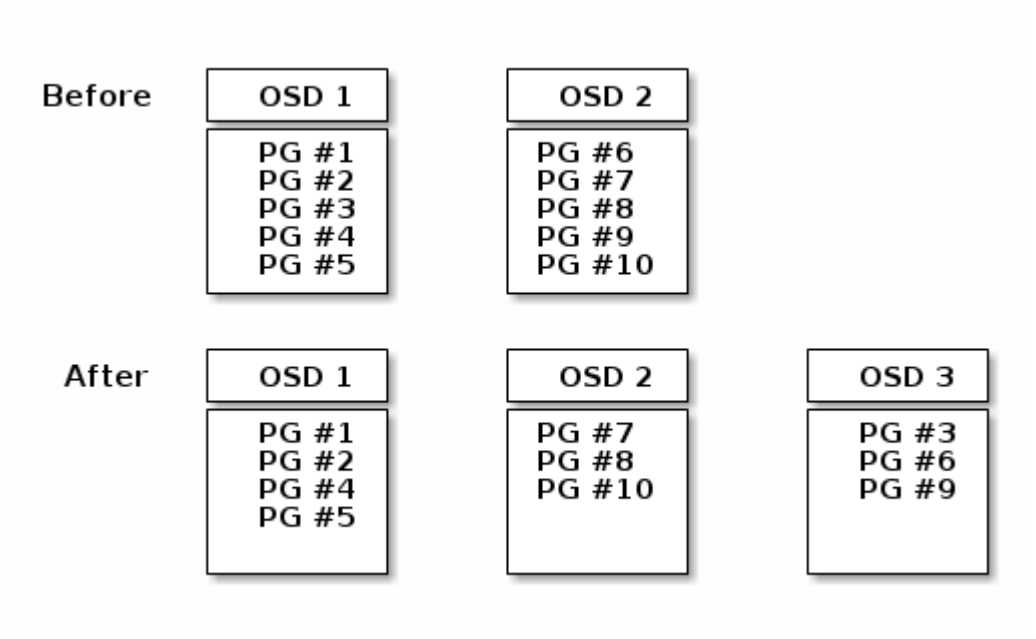

Figure 18: Rebalancing Process

3.2.7 Snapshots Support

Ceph supports the creation of snapshots and allows you to keep the status of images. A Ceph clone is a read-only copy of the active image. However, before you create a snapshot, it is important to stop I/O operations to prevent an inconsistent state of the snapshot. Figure 19 shows an example of Ceph snapshot.

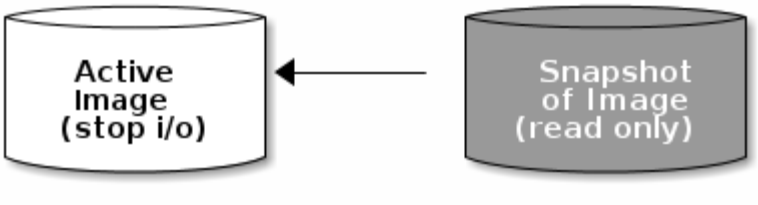

Figure 19: Ceph Snapshot

Ceph, also, supports the creation of copy-on-write (COW) snapshots clones, which allows you to create clones quickly without having to use the primary instance. In a COW policy, if a resource is duplicated but not modified, it is not necessary to create a new resource; the resource can be shared between the copy and the original. Modifications must still create a copy, hence the technique: the copy operation is deferred to the first write. By sharing resources in this way, it is possible to significantly reduce the resource consumption of unmodified copies, while adding a small overhead to resource-modifying operations. Each cloned image, as shown in Figure 20, stores a reference to the main snapshot, which allows the cloned image to open the main snapshot and read it. A COW clone of a snapshot is just like any other Ceph image. You can read, write, clone and resize cloned images.

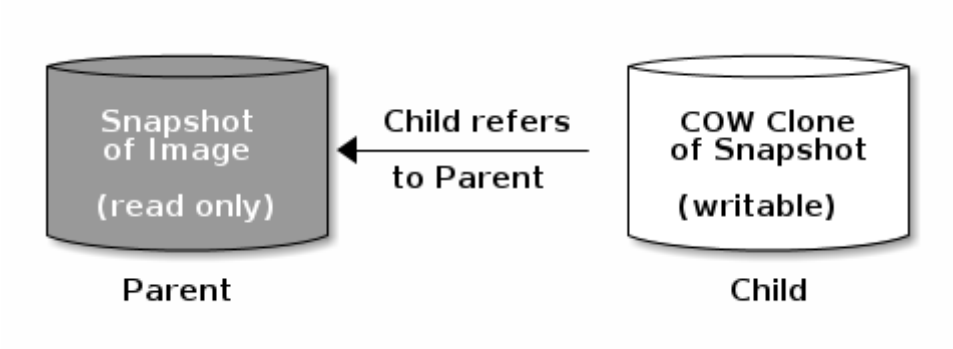

Figure 20: Ceph Clone Snapshot

3.2.8 Accessibility

Ceph Clients include a number of service interfaces including Block Device, Object Storage, and Filesystem.

The Ceph Block Device provides resizable, thin-provisioned block devices with snapshotting and cloning. Ceph stripes a block device across the cluster for high performance and supports both kernel objects (KO) and a QEMU hypervisor that uses librbd directly–avoiding the kernel object overhead for virtualized systems.

The Ceph Object Storage provides RESTful APIs with interfaces that are compatible with most common systems.

The Ceph Filesystem provides a POSIX compliant filesystem usable with a mount or as a filesystem in userspace (FUSE).

The Ceph Object Storage daemon (radosgw) is a FastCGI service that provides a RESTful HTTP API to store objects and metadata. It layers on top of the Ceph Storage Cluster with its own data formats and maintains its own user database, authentication, and access control. The RADOS Gateway uses a unified namespace, which means you can use either the OpenStack Swift-compatible API or the Amazon S3-compatible API.

A Ceph Block Device stripes a block device image over multiple objects in the Ceph Storage Cluster, where each object gets mapped to a placement group and distributed, and the placement groups are spread across separate ceph-osd daemons throughout the cluster. In virtual machine scenarios, people typically deploy a Ceph Block Device with the rbd network storage driver in QEMU/KVM, where the host machine uses librbd to provide a block device service to the guest. Many cloud computing stacks use libvirt to integrate with hypervisors.

The Ceph Filesystem (Ceph FS) provides a POSIX-compliant filesystem as a service that is layered on top of the object-based Ceph

Storage Cluster. Ceph FS files get mapped to objects that Ceph stores in the Ceph Storage Cluster. Ceph Clients mount a CephFS filesystem as a kernel object or as a Filesystem in User Space (FUSE).

3.2.9 Persistence Storage Support in Containers Environment

Ceph is, of course, a scale-out software-defined storage system that provides blocks, files and objects storage. Once a container starts, the point where need persistent storage is mounted to an RBD (Rados Block Device) image.

A RADOS Block Device (RBD) is software that facilitates the storage of block-based data in the open source Ceph distributed storage system. The RBD software breaks up block-based application data into small chunks. The data chunks are subsequently stored as objects within the RADOS. RBD orchestrates the storage of the objects in virtual block devices throughout the Ceph storage cluster. These block devices are the virtual equivalent of physical disk drives.

RBD client then maps the RBD image to the appropriate container node and mounts it using desired filesystem (for example ext4). The wonderful thing is everything happens automatically and the Ceph storage administrators only need to manage container project quotas for storage.

In Ceph we need to create an RBD pool for the containers and also create a Ceph authx keyring to access the Ceph cluster from containers applications. In this way, developers or users can easily consume Ceph storage within containers applications.

3.3 GlusterFS

GlusterFS [13] is a scalable network filesystem. Using common off-the-shelf hardware, it is possible to create large, distributed storage. GlusterFS is free and open source software. GlusterFS aggregates various storage servers over Ethernet into one large parallel network file system.

GlusterFS has a client and server component. Servers are typically deployed as storage bricks, with each server running a glusterfsd daemon to export a local file system as a volume. The glusterfs client process, which connects to servers with a custom protocol over TCP/IP, InfiniBand or Sockets Direct Protocol, creates composite virtual volumes from multiple remote servers using stackable translators.

A key element of GlusterFS is volumes. A volume is a logical collection of bricks where each brick is an export directory on a server in the trusted storage pool. Most of the gluster management operations are performed on the volume. To create a new volume in the storage environment, specify the bricks that comprise the volume. After you have created a new volume, you must start it before attempting to mount it.

3.3.1 Architecture

GlusterFS has a client-server design in which there is no metadata server. Instead, GlusterFS stores data and metadata on several devices attached to different servers. The set of devices is called a volume which can be configured to stripe data into blocks and/or replicate them. Thus, blocks will be distributed and replicated across several devices inside the volume.

Most of the functionality of GlusterFS is implemented as translators, including file-based mirroring and replication, file-based striping, file-based load balancing, volume failover, scheduling and disk caching, storage quotas, and volume snapshots with user serviceability.

The GlusterFS server exports an existing directory as-is, leaving it up to client-side translators to structure the store. The clients are stateless, do not communicate with each other, and are expected to have translator configurations consistent with each other. GlusterFS relies on an elastic hashing algorithm, rather than using either a centralized or distributed metadata model.

Volumes can be added, deleted, or migrated dynamically, helping to avoid configuration coherency problems, and allowing GlusterFS to scale up to several petabytes on commodity hardware by avoiding bottlenecks that normally affect more tightly coupled distributed file systems.

Figure 21 shows a possible architecture of GlusterFS using a distributed volume.

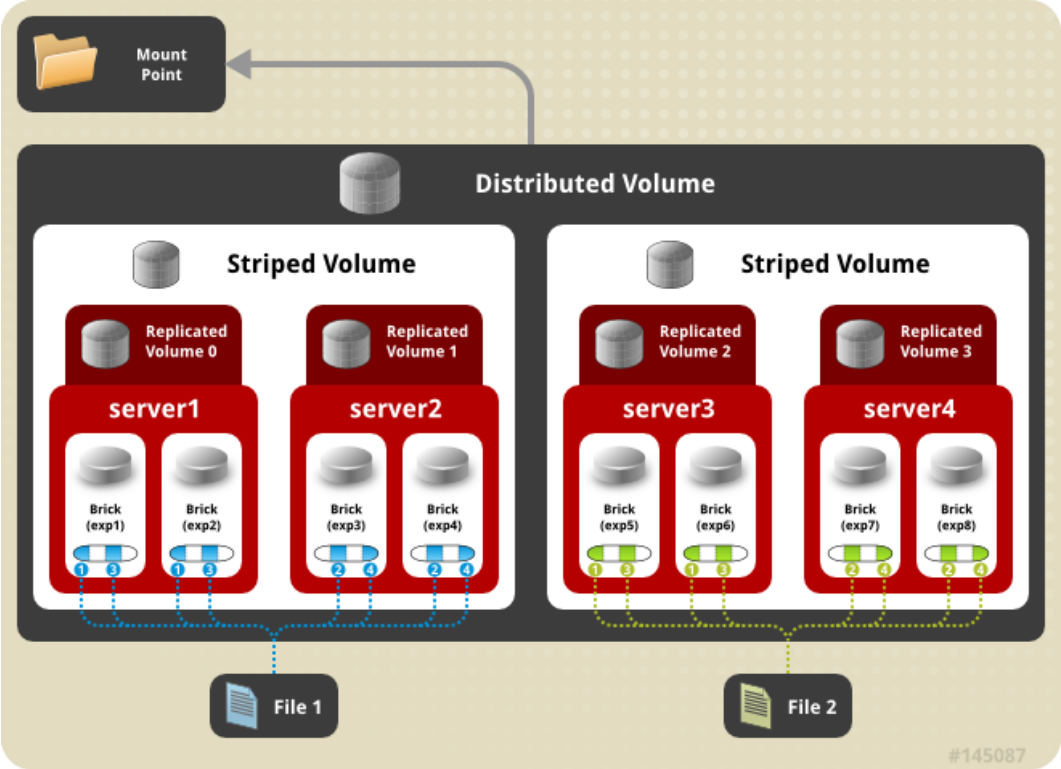

Figure 21: GlusterFS Architecture

3.3.2 Deployment Methods

GlusterFS has different methodologies to deploy. In this work, we preferred to follow the installation recommended by the official documentation, but it is possible to deploy GlusterFS automatically using Puppet.

First, you must have at least two nodes with a network connection and at least two disks. It is recommended that you format the disks before you begin and mount them in the desired directory.

Next, you need to install the necessary packages directly from the official repository and start the server.

After installation, you must configure the cluster through the probe operation that provides to add the various servers to the list of connected peers (this operation is done by a single host and the list is updated on the entire cluster).

Finally, you can create the desired volume using the bricks you created earlier. You must start a volume before use it. Once on, you can mount the volume client side.

3.3.3 Consistency

Consistency in GlusterFS is achieved with the use of a translator. A translator converts requests from users into requests for storage can modify requests on the way through convert one request type to another modify paths, flags, even data.

Figure 20 shows the default/general hierarchy of translators.

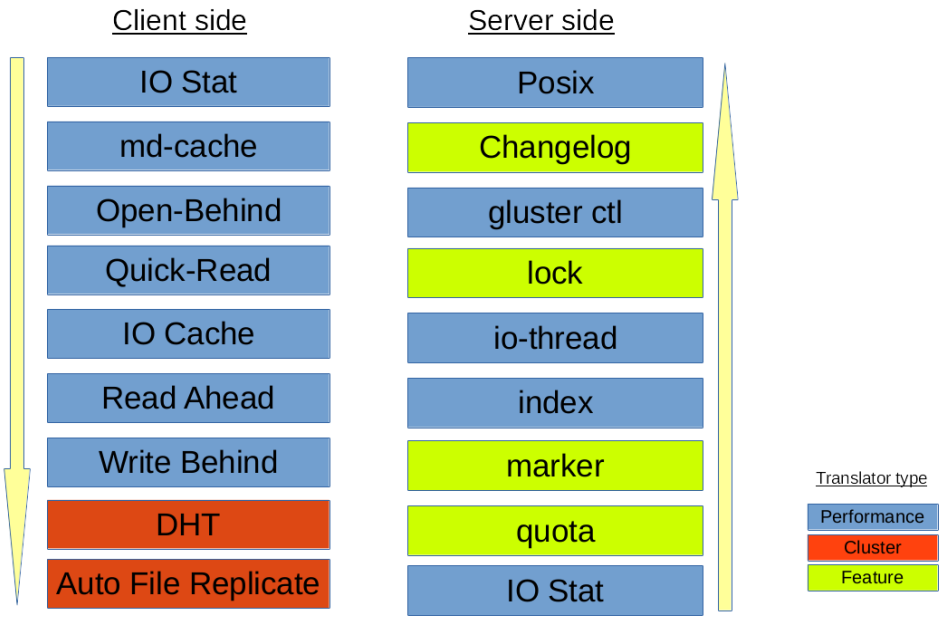

Figure 22: The default/general hierarchy of translators

All the translators hooked together to perform a function is called a graph. The left-set of translators comprises of Client-stack. The right-set of translators comprises of Server-stack. One of the most important and the first translator the data/request has to go through is fuse translator which falls under the category of Mount Translators.

The Distributed Hash Table (DHT) Translator is the real core of how GlusterFS aggregates capacity and performance across multiple servers. Its responsibility is to place each file on exactly one of its subvolumes. It's a routing function, not splitting or copying. The basic method used in DHT is consistent hashing. Each subvolume (brick) is assigned a range within a 32-bit hash space, covering the entire range with no holes or overlaps. Then each file is also assigned a value in that same space, by hashing its name. Exactly one brick will have an assigned range including the file's hash value, and so the file "should"

be on that brick. However, there are many cases where that won't be the case, such as when the set of bricks (and therefore the range assignment of ranges) has changed since the file was created, or when a brick is nearly full. Much of the complexity in DHT involves these special cases, which we'll discuss in a moment.

The Automatic File Replication (AFR) translator in GlusterFS makes use of the extended attributes to keep track of the file operations. It is responsible for replicating the data across the bricks.

Its responsibilities include the following:

- Maintain replication consistency: data on both the bricks should be same, even in the cases where there are operations happening on same file/directory in parallel from multiple applications/mount points as long as all the bricks in replica set are up;
- Provide a way of recovering data in case of failures as long as there is at least one brick which has the correct data;
- Serve fresh data for read/stat/readdir etc.

3.3.4 Replication

When we talk about replication we need to explain the different types of volume present in GlusterFS. Volume is the collection of bricks and most of the gluster file system operations happen on the volume. Gluster file system supports different types of volumes based on the requirements. Some volumes are good for scaling storage size, some for improving performance and some for both. It is important to remember that it is not possible to combine the volume types.

The first type of volume is the Distributed Glusterfs Volume. Here, files are distributed across various bricks in the volume. So file1 may be stored only in brick1 or brick2 but not on both. Hence there is no data redundancy. This also means that a brick failure will lead to complete loss of data because in this type of volume it is not present replication.

Figure 23 shows this type of volume.

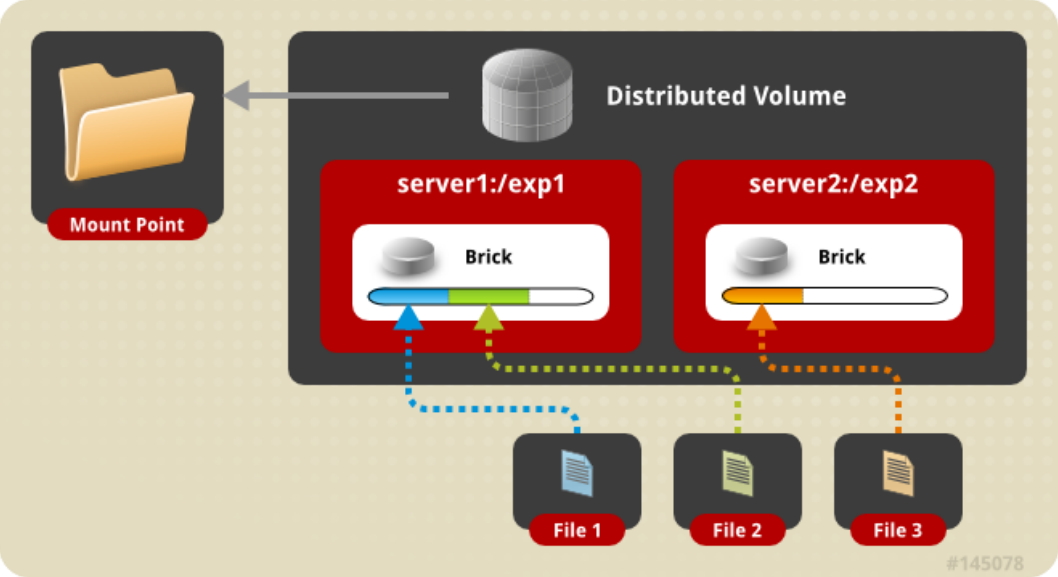

Figure 23: Distributed Glusterfs Volume

In Replicated Glusterfs Volume we overcome the data loss problem faced in the distributed volume. Here exact copies of the data are maintained on all bricks. The number of replicas in the volume can be decided by the client while creating the volume. One major advantage of such a volume is that even if one brick fails the data can still be accessed from its replicated bricks. Such a volume is used for better reliability and data redundancy.

Figure 24 shows this type of volume.

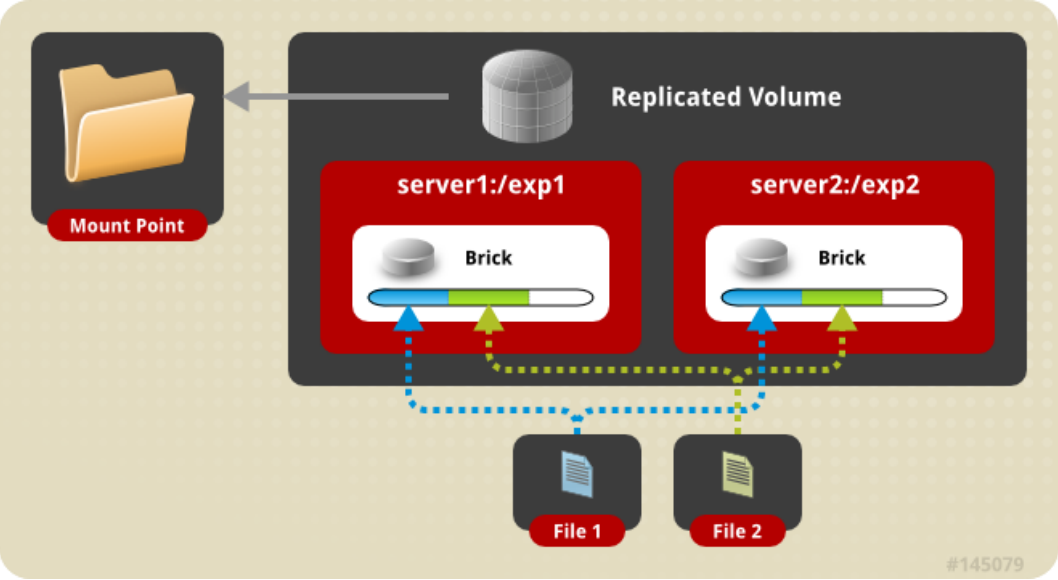

Figure 24: Replicated Glusterfs Volume

In Distributed Replicated Glusterfs Volume files are distributed across replicated sets of bricks. The number of bricks must be a multiple of the replica count. Also, the order in which we specify the bricks matters since adjacent bricks become replicas of each other. This type of volume is used when the high availability of data due to redundancy and scaling storage is required.

Figure 25 shows this type of volume.

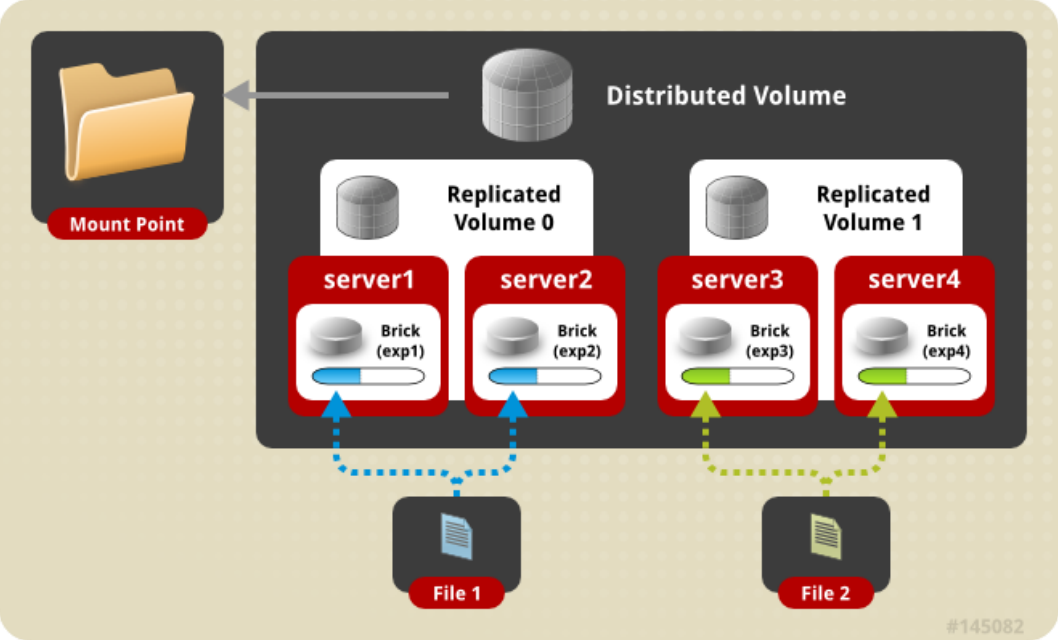

Figure 25: Distributed Replicated Glusterfs Volume

In Striped Glusterfs Volume the data is stored in the bricks after dividing it into different stripes. So, the large file will be divided into smaller chunks and each chunk is stored in a brick. Now the load is distributed and the file can be fetched faster but no data redundancy provided.

Figure 26 shows this type of volume.

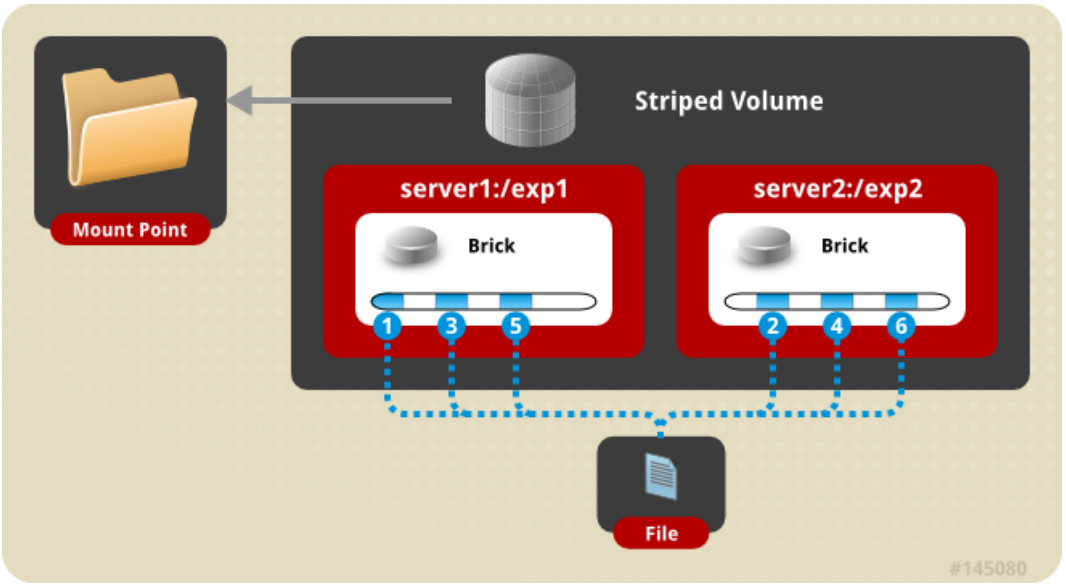

Figure 26: Striped Glusterfs Volume

Distributed Striped Glusterfs Volume is similar to Striped Glusterfs volume except that the stripes can now be distributed across more number of bricks. However, the number of bricks must be a multiple of the number of stripes. So, if we want to increase volume size we must add bricks in the multiple of stripe count.

Figure 27 shows this type of volume.

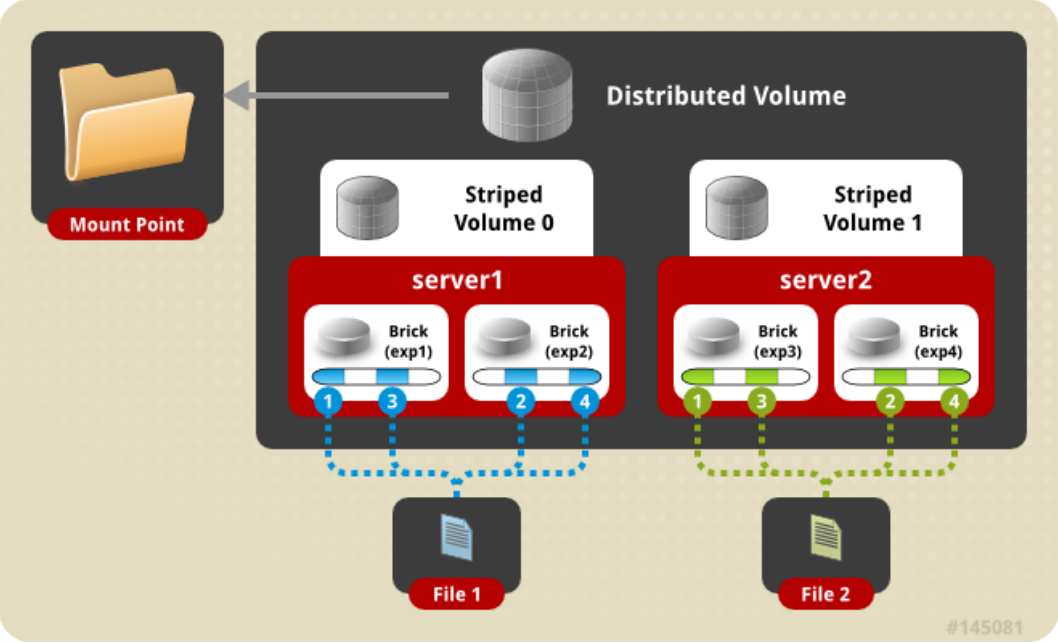

Figure 27: Distributed Striped Glusterfs Volume

3.3.5 Fault Tolerance

Typically, fault tolerance is achieved with the RAID, but, often, storage was shared between servers to allow multiple hosts to access the same files and this would leave the design with several single points of failure.

Traditionally many web host implementations overcame this limitation by using some eventually consistent method to keep a copy of the entire file tree on the local storage for every server.

GlusterFS replace the fault tolerance of RAID with replication allows you to spread that vulnerability between 2 or more complete servers. With multipath routing using two or more network cards, it is possible to eliminate most single points of failure between clients and data. If a server or network connection goes down, the client will continue to operate with the remaining servers.

The downside of this method is that in order to assure consistency and that the client is not getting stale data, it needs to request metadata from each replica. This is done during the lookup portion of establishing a file descriptor (FD). This can be a lot of overhead if you're opening thousands of small files, or even if you're trying to open thousands of files that don't exist.

3.3.6 Load Balancing

GlusterFS uses a hash function to convert files' pathnames into values and assigns several logical storages to a different set of value. This is uniformly done and allows GlusterFS to be sure that each logical storage almost holds the same number of files. Since files have not the same size and storage devices can be added or removed, GlusterFS affects each logical storage to a physical one according to the used disk space. Thus, when a disk is added or removed, virtual storage can be moved to a different physical one in order to balance the system.

In addition, by having your files replicated between multiple servers, in a large-file read-heavy environment, such as a streaming music or video provider, you have the ability to spread those large reads among multiple replicas. The replica translator works on a first-to-respond basis so if requesting a specific file becomes popular enough that it starts to cause load on a server, the less loaded server will respond first ensuring the optimal performance to the end-user.

3.3.7 Snapshots Support

GlusterFS snapshots are thinly provisioned Logical Volume Manager (LVM) based snapshots, and hence they have certain pre-requisites. Considering a volume with two bricks, that are mounted on independent, thinly-provisioned LVMs. When the volume is mounted, the client process maintains a connection to both the bricks. A GlusterFS snapshot, is also internally a GlusterFS volume with the exception that, it is a read-only volume and it is treated differently than a regular volume in certain aspects.

When we take a snapshot of the GlusterFS volume it is checked if the volume is in started state, and if so are all the brick processes up and running. At this point in time, we barrier certain file operations, in order

to make the snapshots crash-consistent. What it means is even though it is an online snapshot, certain write file operations will be barriered for the duration of the snapshot. The file operations that are on the fly when the barrier is initiated will be allowed to complete, but the acknowledgment to the client will be pending till the snapshot creation is complete.

After successfully barriering file operations on all brick processes, we proceed to take individual copy-on-write LVM snapshots of each brick. A copy-on-write snapshot LVM snapshot ensures a fast, space-efficient backup of the data currently on the brick. These LVM snapshots reside in the same LVM thin pool as the GlusterFS brick LVMs.

Once this snapshot is taken, we carve bricks out of these LVM snapshots and create a snapshot volume out of those bricks. Once the snapshot creation is complete, the system unbarrier the GlusterFS volume.

3.3.8 Accessibility

GlusterFS is a userspace filesystem. Being a userspace filesystem, to interact with kernel Virtual File System (VFS), GlusterFS makes use of FUSE (File System in Userspace).

FUSE is a kernel module that supports interaction between kernel VFS and non-privileged user applications and it has an API that can be accessed from userspace. Using this API, any type of filesystem can be written using almost any language you prefer as there are many bindings between FUSE and other languages.

Figure 28 shows the Structural diagram of FUSE.

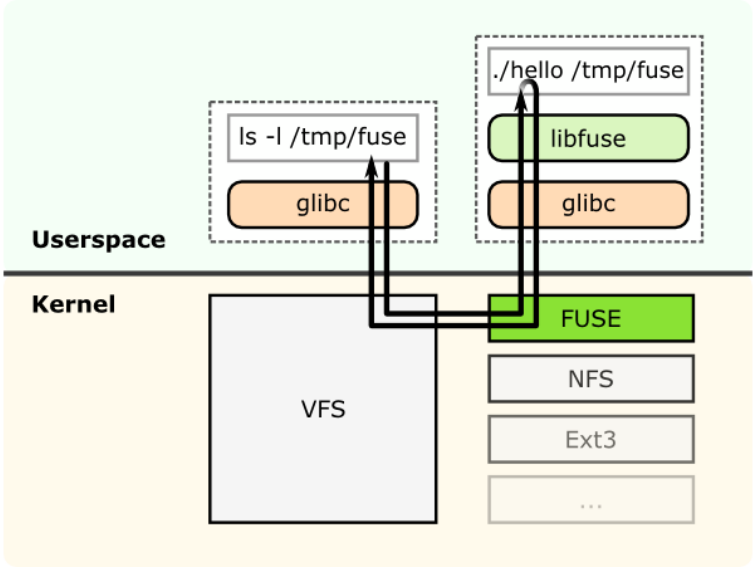

Figure 28: Structural Diagram of FUSE

3.3.9 Persistence Storage Support in Containers Environment

GlusterFS is one of the most widely distributed file systems. Probably, this is due to many features like ease of use, the ability to use FUSE, various types of volumes, etc.

For persistent storage in containers, GlusterFS offers the ability to use its volumes as container storage. Integrating a GlusterFS volume into a container is very easy.

As already mentioned, you can mount a portion of the container file system on a portion of the file system of the host; GlusterFS allows you to mount your volumes on a portion of client-side file system, so you can mount the volume in a portion of the host file system and keep it synchronized with the corresponding volume. Then, just mount the container file system portion into the host file system portion where the GlusterFS volume is mounted. Once the container uses the GlusterFS

volume, it benefits from all the features it offers based on the volume configuration.

3.4 XtreemFS

XtreemFS [19] is a general-purpose storage system. It covers most storage needs in a single deployment. It is open-source, requires no special hardware or kernel modules, and can be mounted on Linux, Windows and OS X.

It is an object-based, distributed file system for wide area networks. XtreemFS' outstanding feature is full (all components) and real (all failure scenarios, including network partitions) fault tolerance, while maintaining POSIX file system semantics. Fault-tolerance is achieved by using Paxos-based lease negotiation algorithms and is used to replicate files and metadata. TLS and X.509 certificates support make XtreemFS usable over public networks.

3.4.1 Architecture

XtreemFS is a federated replicated object-based file system, it is designed to operate on a set of clusters that are connected by a wide area network. It runs on commodity hardware and doesn't require any specific. Although its components are distributed, XtreemFS provides the user with a consistent view of the entire file system. It supports replication of both file metadata and objects. Furthermore, metadata can be partitioned and objects can be striped over multiple servers for a high-throughput parallel access.

Server-side, XtreemFS provide several components:

- Object Storage Device (OSD): OSDs are responsible for storing file content. The content of a single file is represented by one or more objects. Files are replicated by holding replicas of the corresponding objects on different OSDs;
- Metadata and Replica Catalog (MRC): MRCs constitute the metadata service. For availability and performance reasons, there may be multiple MRCs running on different hosts. The MRC offers an interface for metadata access. It provides functionality such as creating a new file, retrieving information about a file or renaming a file;
- Replica Management Service (RMS): The RMS is the responsible for deciding when new replicas should be created or removed. According to the application needs, the topology of the network and the legal limitation on file location, the service will initiate the creation or deletion of replicas.

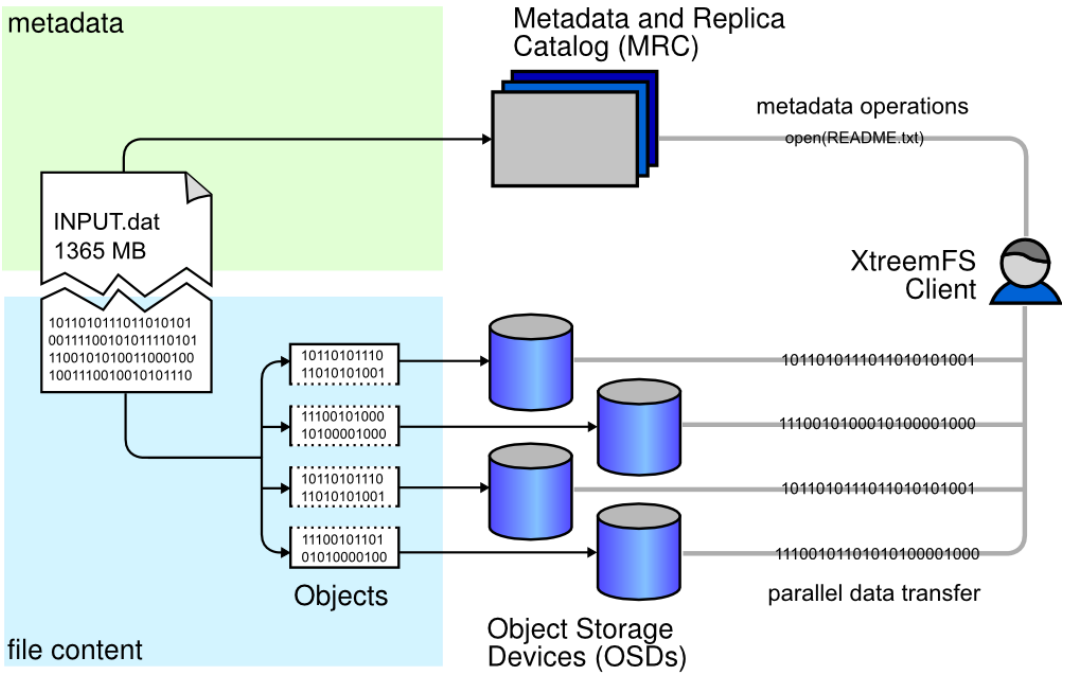

Figure 29: XtreemFS architecture

Figure 29 shows the XtreemFS architecture.

3.4.2 Deployment Methods

XtreemFS has different methodologies to deploy. In this work, we preferred to follow the installation recommended by the official documentation.

First, you must add the XtreemFS repository to your system and install the packages *xtreemfs-server* and *xtreemfs-client*.

Next, you need just to start the Directory Service, the Metadata Server, the OSD and the FUSE kernel module.

After installation, you must mount the desired volumes in the desired locations.

3.4.3 Consistency

XtreemFS provides the user with a consistent view of the entire file system. The consistency of replicas is handled by the OSDs in a distributed manner. To handle consistency issues, it is used a leases mechanism to make sure that only a single OSD is able to modify a specified part of the data at a given time (single writer, multiple readers). If many clients want to modify the same data concurrently, they will be redirected to the OSD that has the lease to avoid unnecessary coordination of operations

3.4.4 Replication

XtreemFS replicates the file data across multiple storage servers, which can be distributed worldwide. The replication algorithm is designed to cope with all the problems and failure scenarios that can occur in truly distributed systems: message loss, network partitions, server crashes, etc. For each replicated file, XtreemFS keeps a list of replicas which gives you complete control where to place replicas. You can even use different placement schemes for each file.

When a replicated file is opened, XtreemFS automatically elects a primary replica for that file which processes all file operations. If the primary fails, one of the backup replicas will automatically take over after a short failover period. Your applications won't see a difference between regular and replicated file because the XtreemFS file replication is completely transparent.

In addition to regular file replication, XtreemFS provides read-only replication. This replication mode works on immutable files and supports a large number of replicas. The read-only replication helps you quickly build a caching infrastructure on top of XtreemFS in order to reduce latency and bandwidth consumption between datacenters. In addition to full replicas that contain a complete copy, XtreemFS also supports partial replicas. These replicas are filled on demand when a client accesses data.

3.4.5 Fault Tolerance

XtreemFS provides fault tolerance and scalability by supporting replication and striping of files. File replica management can be automated by means of a replica management service. A replicated and partitioned metadata service ensures that metadata is highly available and metadata operations can be performed quickly.

XtreemFS provides fault tolerance while maintaining POSIX file system semantics. Fault tolerance is achieved by using Paxos-based lease negotiation algorithms and is used to replicate files and metadata.

3.4.6 Load Balancing

XtreemFS provides an automatic load balancing. When a node with an OSD is down, files are handled by other nodes and the Replication Management Service handles where to place the replicas.

Taking as an example a streaming video, when a node fails, the video stops for a short time and XtreemFS will provide a new streaming link to continue the streaming.

3.4.7 Snapshots Support

XtreemFS can take file system snapshots. A snapshot captures an instantaneous image of all files and directories in a volume, which can later be accessed in a read-only manner.

Support for snapshots comes at the cost of additional storage and I/O overhead, as snapshots require copy-on-write versioning of files across the OSDs. Snapshots are therefore disabled by default, but they can be enabled.

Snapshots are exposed as read-only volumes. To access a snapshot, it is necessary to mount it. A mounted volume snapshot can be browsed normally, and all files can be read as in the original volume. However, any attempt to write data on a snapshot will result in an error.

A snapshot only captures a file in its current state if it is closed. Files that are open when taking a snapshot are captured in the last state in which they were before they were opened. Since files are implicitly closed on an OSD through a timeout rather than an explicit close call, it may happen that files are not included in a snapshot despite having been closed at application level before the snapshot was taken. To make sure a change to a specific file is included in a subsequent snapshot, it is necessary to wait for the close timeout on the OSD before taking the snapshot, which by default is set to 60 seconds.

3.4.8 Accessibility

The XtreemFS access layer represents the interface between user processes and the file system infrastructure. It manages the access to files and directories in XtreemFS for user processes as well as the access to grid specific file system features for users and administrators.

It is the client-side part of the distributed file system and as such has to interact with all components of XtreemFS: MRC, OSD, and RMS. The access layer provides a POSIX interface to the users by using the FUSE framework for file systems in userspace. It translates calls to the POSIX API into corresponding interactions with OSDs and MRCs.

A XtreemFS volume will be simply mounted like a normal file system and users will not need to modify or recompile their application for accessing it. In addition to the POSIX interface, the access layer provides tools for creating, deleting XtreemFS volumes, create and move file replicas, check file integrity, query and change the file striping policies, and other grid specific features

3.4.9 Persistence Storage Support in Containers Environment

XtreemFS provide FUSE access to use the DFS as a normal file system. For persistent storage in containers, it offers the ability to use its volumes as container storage. Integrating a XtreemFS volume into a container is similar to GlusterFS volume.

As already mentioned, you can mount a portion of the container file system on a portion of the file system of the host; XtreemFS allows you to mount your volumes on a portion of client-side file system, so you can mount the volume in a portion of the host file system and keep it synchronized with the corresponding volume. Then, just mount the

container file system portion into the host file system portion where the XtreemFS volume is mounted.

Once the container uses the XtreemFS volume, it benefits from all the features it offers.

3.5 HDFS, Ceph, GlusterFS and XtreemFS Comparison

In the previous sections, we talked about HDFS, Ceph, GlusterFS, and XtreemFS. We've seen how they are structured, how to install them, how to ensure consistency, replication and fault tolerance, etc.

The table in Figure 30 shows the major differences that emerged in this chapter. It is possible to see the differences between HDFS, Ceph, GlusterFS, and XtreemFS considering the points that are illustrated in previous sections.

This comparison has highlighted some important differences between HDFS, Ceph, GlusterFS, and XtreemFS. The three Distributed File Systems have a different architecture, but all have an approach for automatic installation. Consistency is guaranteed in a variety of ways by each of them. Replication and Load Balancing are automated in HDFS, Ceph, and XtreemFS, but not in GlusterFS. Support for Fault Tolerance depends on architectural choices. Snapshots are supported by all four technologies. HDFS has less Accessibility than Ceph, GlusterFS, and XtreemFS. The persistent storage support in container environments is not natively supported by HDFS.

These are the differences that emerged from a study of the documentation of the four technologies. This is the starting point for building the benchmark for distributed file systems that we will discuss in the next chapter.

Feature	HDFS	Ceph	GlusterFS	XtreemFS
Architecture	Centralized Master/Slave	Distributed using Monitors and OSDs	Decentralized using Bricks distribution	Distributed using OSDs, MRCs and RMSs.
Development Methods	Automatic using file script and XML configurations	Automatic using native tool (ceph-deploy) or other tools (juju, puppet, Chef, etc.)	Automatic using official repository or tools (puppet, vagrant, etc.)	Automatic using official repository
Consistency	WORM, lease	Lock	AFR	Leases mechanism
Replication	Automatic Block Replication in slave nodes	Automatic PGs contents replication in OSDs	Depending on Volume type	Automatic with RMSs
Fault Tolerance	Support for NameNode, DataNode and Network failure	Support for Monitor and OSD failure	RAID-like	Paxos based
Load Balancing	Automatic Rebalancing	Automatic Rebalancing	Manual Rebalancing	Automatic Rebalancing
Snapshots Support	Supported	Supported	Supported	Supported
Accessibility	C API, Java API and REST	FUSE, mount, REST, RADOS, Block/Object Device, and Filesystem	FUSE, mount	FUSE, mount
Persistence Storage Support in Containers Environment	Not Supported	Supported	Supported	Supported

Figure 30: HDFS, Ceph, GlusterFS, and XtreemFS Comparison

Chapter 4: Distributed File Systems Benchmark

A distributed file system, as the name suggests, is distributed over multiple nodes in a network. In a typical architecture, we have a distributed storage provider that deals with providing the service with a certain QoS and using a custom application that he or she handles access to storage services in a way more or less transparent as if it were a local file system.

We have two parts, customers and providers, and a network in the middle that allows them to communicate. To be able to evaluate the performance of a distributed file system, we must monitor client-side and server-side behaviors and keep in mind that they might be strongly influenced by the network.

To test the technology, it is good to set up a test plan, which is a document describing the scope, the approach, the resources, and the scheduling of the testing activities you intend to perform. It's good practice to identify the elements to be tested, the features to be verified, the individual testing activities and all the risks that require specific checks.

Performing a benchmark of distributed file systems do not just focus on monitoring cluster's keys (such as CPU loads, busy RAM, storage usage, and bandwidth), but we need to go to evaluate case-by-case as a particular distributed file system can be helpful or not.

To do a good benchmark there is the need to test functional and non-functional features. The first is a process that guarantees quality assurance of a particular product or service and is based on the specifications of the product you want to examine. Generally, certain functions are tested giving them a certain input, and subsequently evaluating their success through output analysis. This type of test sees the product as a "black box", it does not take into account the internal structure, but it is considered the communication with the system

through specific APIs. A functional test, therefore, is a type of behavioral test (focused on an observable product behavior) that looks at the system as a collection of functions. An example of this kind of test applicable to distributed file systems is the ability to use client-side an application that offers compatibility with the POSIX system.

Testing a system is usually considered appropriate to test the system against non-functional requirements such as safety, speed, accuracy, and reliability. At this level, external interfaces are also tested against other applications, standard components, hardware devices, and the operating environment.

During a benchmark, therefore, you cannot focus solely on functional tests as you will not be able to analyze many parts of the system that are of enormous importance.

4.1 Evaluation Criteria

In a distributed file system, provider-side there is the part that takes care of storing the data in memory, providing recovery in case of failures, replication support, etc. All of these features (which may or may not be present in a distributed file system) mean that a particular technology is suitable for a specific use rather than another.

Below there is a list of evaluation criteria [20] that are used for distributed file systems benchmarking.

4.1.1 Architecture

Software architecture refers to the high-level structures of a software system, how to create such structures, and the documentation of these structures. Each structure comprises software elements, relations among them, and properties of both elements and relations. Software architecture is about making fundamental structural choices which are costly to change once implemented. Software architecture choices include specific structural options from possibilities in the design of software.

A file system, local or distributed, is primarily concerned with the implementation of the POSIX API system, including file access, access control, metadata operations, etc. A functional test goes to test the architectural model of the file system and how it implements the APIs.

The POSIX (Operating System Interface Portable) system guarantees the portability of a given interface for data access on the various systems. The aim is to improve the portability of the various applications in a scenario where the underlying devices are heterogeneous.

A distributed file system that provides compatibility with POSIX semantics ensures that data can be replicated on multiple devices (even with different operating systems) and, above all, that applications can access them without having to undergo changes depending on the operating system used.

Deployment is the delivery and execution of a particular application. This operation consists of a set of under relatively important operations since they allow you to approach deployment in several ways. The most relevant operations are how to install an application and how to provide scalability support.

Systems offer us various deployment modes that are based on different approaches:

- Manual Approach: the user determines every single object/component and transfers it to the appropriate nodes with an appropriate sequence of commands;
- Scripting Approach: some script files (scripts in some shell, bash, perl, etc.) have to be executed that enclose the command sequence to get to the configuration that has dependencies between objects;
- Model-based or declarative language-based approach: automatic configuration support through declaring languages or configuration patterns to be obtained (eg XML deployment files and Java5 annotations);
- Tool-based approach that automates installation: the user can use some tools that automatically deploy the application.

Distributed file systems have different deployment modes, and most of all, there are many tools developed many times by third parties to automate the installation phases. One feature common to these file systems, however, is the ability to take advantage of "scale-out" so that you can build large-scale and high-performance clusters.

Deployment testing, therefore, involves system evaluation in various scenarios such as automatic setup configuration, cluster size, hardware configuration (server, storage, network, etc.), automatic load balancing, High availability, etc.

Usability is defined as the effectiveness, efficiency, and satisfaction with which certain users reach certain goals in certain contexts. It defines the degree of satisfaction with which the interaction between man and a system is performed. The term does not refer to an intrinsic feature of the system, but to the interaction process between users and system. The usability issue arises when the designer's model (ideas about product operation that transfers to the design of the product itself) does not coincide with the end user model (the idea that the user conceives of the product and its functioning).

In a distributed file system, usability can be seen as making available certain end-user features to simplify its use.

Extensibility is the ability of a system to grow to meet the needs of an increasing number of users. The concept of extensibility is easily accountable to scalability that refers to the capabilities of a system to be able to grow or decrease the number of nodes depending on needs and availability. Talking about these parameters makes sense since large-scale cloud computing has spread.

According to the definition of NIST, cloud computing should provide on-demand services on the network, exploiting the concept of a pool of resources, trying to be as flexible and fast as possible. A distributed file system is the base of storage in a cloud computing system. Extensibility tests relate to the ability of the file system to expand to the lowest possible impact on performance.

Stability is the ability of a system that dynamically changes its architecture to continue to provide the same functionality transparently to its internal modifications. The user using a system that tends to expand inside does not have to be aware of what is happening internally

to the system and to accomplish this, there is no need for the system to present alterations or changes in the QoS.

A distributed file system after the installation phase becomes available on the network from a multitude of customers; The idea is that this network availability is infinite and does not suffer any interruption. Stability tests are typically carried out over relatively long periods (from 30 to 180 days 24 hours a day) and monitor the ability of the system to continue to perform normally despite operations being carried out internally.

A distributed file system must also be evaluated in its integration capabilities with other systems. Here is the possibility of using DFS by another complex system. For example, Ceph is used by OpenStack.

Another point of interest is the supported platforms. This point evaluates how much a distributed file system is really multiplatform. A particular DFS can run on different operating systems (Windows, Linux or Mac) or not and this is a big difference if you want to build your own storage system on multiple platforms.

Computer security is very important in computer science. Typically, in a distributed system, some client applications access the file system. These applications can be used with or without access. For greater security, the various DFSs support different methodologies to ensure security.

4.1.2 Consistency

The consistency guarantees to the user that the data are significant and usable within their applications. When analyzing a distributed file system, consistency takes on some importance when you need to maintain consistency between the data in the file system and the data coming from outside. Consistency in a file system means that following various operations you can continue to ensure data integrity by avoiding any loss or error on the files.

In a distributed file system, it is also important to ensure consistency in the replication of files on the various nodes; this means that a file that has been replicated on nodes A, B and C must be consistent for users who want to use it from any of the 3 nodes.

One aspect that needs to be taken into account when talking about file systems, in general, is the possibility that they have multiple processes to write the same file. If two users write the same file at the same time, this would affect the consistency of the same file. It is, therefore, necessary to also examine the various lock mechanisms that may highlight certain incorrect behaviors of a file system.

Consistency is also related to the concept of the snapshot. A snapshot represents an object's state at a given instant of time. The term, in computer science, is used to represent the state of a system at a specific time. Its use is related to backup and restore aspects. The idea is that you can have multiple snapshots of the same system at different instants of time so you can bring the system back to a previous state.

When creating snapshots, the snapshot mechanism must ensure that they are consistent. This means that each snapshot starts at a certain point-in-time and the image of the disk or volume that the snapshot will reflect will be the exact image of the volume at that point-in-time.

A snapshot can be:

- Crash-consistent snapshots: while apps are running, there may be open transactions, buffered memories in the disk, open files, and other uncompleted tasks. When a snapshot is executed during execution, all that is in memory is discarded. The snapshot is "crash-consistent", which means it's the same as if someone pulled out a computer's power cable and then reinserted it. The most modern applications (such as databases) know how to recover from that state. Modern file systems such as Ext3, Ext4, and NTFS have a journaling mechanism that will allow recovering open files without leaving them corrupt. The databases have transaction logs, which will allow them to return to the last consistent state even if they were at the center of a transaction;
- Application-consistent snapshots: this is the case when you want to make sure that a snapshot is consistent, which means that when a restore is made, applications resume from a consistent state and do not have problems. Most applications have APIs to let you know when backups are running. They can then make sure the transactions are complete, flushed buffers, closed files, etc. The application enters what is called "backup mode" or simply a "freeze". While the application is freeze, it cannot respond to requests, at least, of writing (which means that depending on the case you can continue to serve read requests or not). It is important to keep the application frozen for as short a time as possible, so as not to hinder the application operations (as the requests may start to fail); the assumption is that the application must remain alive during backup and a short freeze will not fail. Because snapshots are consistent with their starting point, once the snapshot starts, you can release the "freeze" operation regardless of how long it takes to complete the snapshot. The snapshot mechanism will ensure snapshot consistency.

4.1.3 Fault Tolerance, Replication, Striping and Caching

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause a total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems. The ability of maintaining functionality when portions of a system break down is referred to as graceful degradation.

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. The term is most commonly used to describe computer systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software.

Within the scope of an individual system, fault tolerance can be achieved by anticipating exceptional conditions and building the system to cope with them, and, in general, aiming for self-stabilization so that the system converges towards an error-free state. However, if the consequences of a system failure are catastrophic, or the cost of making it sufficiently reliable is very high, a better solution may be to use some form of duplication. In any case, if the consequence of a system failure is so catastrophic, the system must be able to use reversion to fall back to a safe mode. This is similar to roll-back recovery but can be a human action if humans are present in the loop.

In computer data storage, data striping is the technique of segmenting logically sequential data, such as a file, so that consecutive segments are stored on different physical storage devices.

Striping is useful when a processing device requests data more quickly than a single storage device can provide it. By spreading segments across multiple devices which can be accessed concurrently, total data throughput is increased. It is also a useful method for balancing I/O load across an array of disks. Striping is used across disk drives in a redundant array of independent disks (RAID) storage, network interface controllers, different computers in clustered file systems and grid-oriented storage, and RAM in some systems.

One method of striping is done by interleaving sequential segments on storage devices in a round-robin fashion from the beginning of the data sequence. This works well for streaming data, but subsequent random accesses will require knowledge of which device contains the data. If the data is stored such that the physical address of each data segment is assigned a 1-to-1 mapping to a particular device, the device to access each segment requested can be calculated from the address without knowing the offset of the data within the full sequence.

Other methods might be employed in which sequential segments are not stored on sequential devices. Such non-sequential interleaving can have benefits in some error correction schemes.

Advantages of striping include performance and throughput. Sequential time interleaving of data accesses allows the lesser data access throughput of each storage devices to be cumulatively multiplied by the number of storage devices employed. Increased throughput allows the data processing device to continue its work without interruption, and thereby finish its procedures more quickly. This is manifested in improved performance of the data processing.

Because different segments of data are kept on different storage devices, the failure of one device causes the corruption of the full data

sequence. In effect, the failure rate of the array of storage devices is equal to the sum of the failure rate of each storage device. This disadvantage of striping can be overcome by the storage of redundant information, such as parity, for the purpose of error correction. In such a system, the disadvantage is overcome at the cost of requiring extra storage.

Caching is the process of storing data in a cache, that is a temporary storage area. For example, the files you automatically request are stored on your hard disk in a cache subdirectory under the directory of the distributed file system. When you return to a file you've recently requested, the file system can get those file from the cache rather than the original server, saving you time and saving the network the burden of additional traffic.

4.1.4 Performance and Stressing

A computer system, before it can be put into production, is subjected to a performance check in order to evaluate how it responds to certain stress. The performance of a system is evaluated by different people with different purposes and points of view. However, the common goal is to verify that the system meets certain efficiency requirements and can be used for a particular purpose.

The performance of a distributed file system is to be evaluated on multiple critical dimensions without ever losing sight of the application scenario in which the system must perform. With this analysis, you can determine whether a file system is suitable for a particular application scenario or not. Performance typically takes account of indicators that monitor small and large file shifts and update their metadata.

A computer system used in the enterprise world, as has already been pointed out, must provide certain services for a long-lifetime. During

the life cycle of a system, there are times when requests are contained and moments where, however, requests grow and they can influence the performance of the system. It's fairly easy, therefore, to understand that a good system must be able to maintain its QoS standards even when the load grows. In a distributed file system there is always an upper limit of the load capacity that it can withstand.

When the system reaches the maximum load threshold, some issues may arise as performance degradation, the occurrence of abnormal events or, in extreme cases, the failure to provide the services.

4.2 Benchmarking Tools

The issues to be tested in a Distributed File System have been listed in the previous section. Many of the features do not need a practical phase to be analyzed, but some of them need practical tests. In this section, attention is focused on features that require a practical test phase and how some tools may be helpful or not in structuring these tests.

For the analysis of the architectural features and all those on which no explicit practical tests can be made, the differences will be illustrated by using the official documentation of the various technologies.

For better support during the practical test phase, we have analyzed some tools to evaluate their use in this work. These tools will be briefly illustrated below to understand their features.

4.2.1 Ganglia Monitoring System

Ganglia [21] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

Ganglia is a BSD-licensed open-source project that grew out of the University of California, Berkeley Millennium Project which was

initially funded in large part by the National Partnership for Advanced Computational Infrastructure (NPACI) and National Science Foundation RI Award EIA-9802069. NPACI is funded by the National Science Foundation and strives to advance science by creating a ubiquitous, continuous, and pervasive national computational infrastructure: the Grid. Current support comes from Planet Lab: an open platform for developing, deploying, and accessing planetary-scale services.

It is based on a hierarchical design targeted at federations of clusters. It relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on over 500 clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

The ganglia system comprises two unique daemons, a PHP-based web front-end, and a few other small utility programs.

Ganglia Monitoring Daemon (gmond): gmond is a multi-threaded daemon which runs on each cluster node you want to monitor. Installation does not require having a common NFS filesystem or a database back-end, installing special accounts or maintaining configuration files. Gmond has four main responsibilities: monitor changes in the host state, announce relevant changes, listen to the state of all other ganglia nodes via a unicast or multicast channel and answer requests for an XML description of the cluster state. Each gmond transmits information in two different ways: unicasting or Multicasting

host state in external data representation (XDR) format using UDP messages and sending XML over a TCP connection.

Ganglia Meta Daemon (gmetad): federation in Ganglia is achieved using a tree of point-to-point connections amongst representative cluster nodes to aggregate the state of multiple clusters. At each node in the tree, a Ganglia Meta Daemon (gmetad) periodically polls a collection of child data sources, parses the collected XML, saves all numeric, volatile metrics to round-robin databases and exports the aggregated XML over a TCP socket to clients. Data sources may be either gmond daemons, representing specific clusters, or other gmetad daemons, representing sets of clusters. Data sources use source IP addresses for access control and can be specified using multiple IP addresses for failover. The latter capability is natural for aggregating data from clusters since each gmond daemon contains the entire state of its cluster.

Ganglia PHP Web Front-end: the Ganglia web front-end provides a view of the gathered information via real-time dynamic web pages. Most importantly, it displays Ganglia data in a meaningful way for system administrators and computer users. Although the web front-end to ganglia started as a simple HTML view of the XML tree, it has evolved into a system that keeps a colorful history of all collected data. The Ganglia web front-end caters to system administrators and users. For example, one can view the CPU utilization over the past hour, day, week, month, or year. The web front-end shows similar graphs for memory usage, disk usage, network statistics, number of running processes, and all other Ganglia metrics. The web front-end depends on the existence of the gmetad which provides it with data from several Ganglia sources. Specifically, the web front-end will open the local port 8651 (by default) and expects to receive a Ganglia XML tree. The web pages themselves are highly dynamic; any change to the Ganglia data appears immediately on the site. This behavior leads to a very responsive site but requires that the full XML tree is parsed on every

page access. Therefore, the Ganglia web front-end should run on a fairly powerful, dedicated machine if it presents a large amount of data. The Ganglia web front-end is written in PHP and uses graphs generated by gmetad to display history information. It has been tested on many flavors of Unix (primarily Linux) with the Apache web server and the PHP5 module.

Figure 31 shows the Ganglia Web Interface.

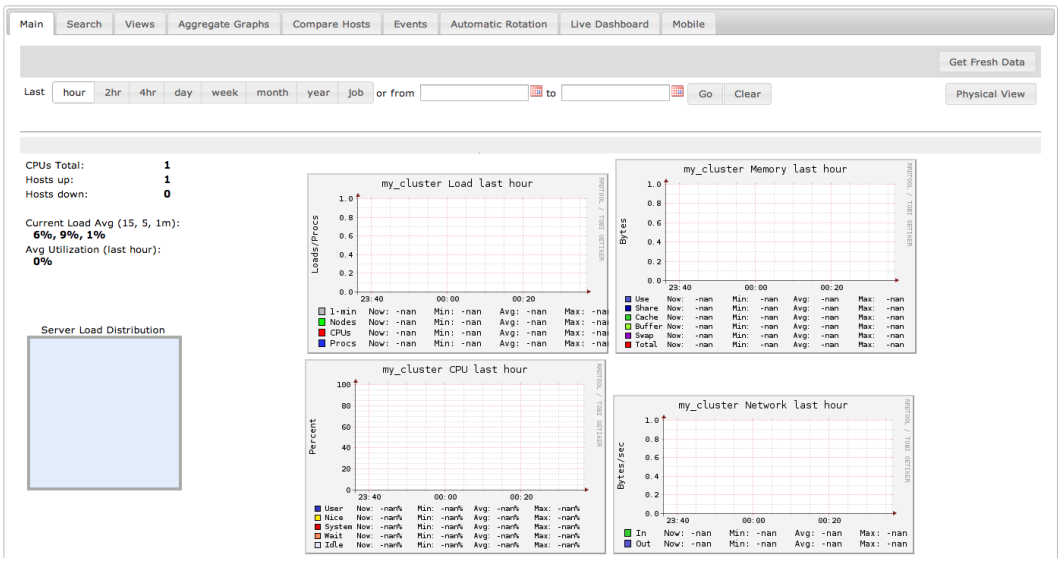

Figure 31: Ganglia Web Interface

4.2.2 Bonnie

Bonnie [22] performs a series of tests on a file of known size. If the size is not specified, Bonnie uses 100 Mb; but that probably isn't enough for a big modern server - you your file to be a lot bigger than the available RAM.

Bonnie works with 64-bit pointers if available. For each test, Bonnie reports the bytes processed per elapsed second, per CPU second, and the %CPU usage (user and system). In each case, an attempt is made to keep optimizers from noticing it's all bogus. The idea is to make sure that these are real transfers between user space and the physical disk.

The tests are:

- Sequential Output:
    - Per-Character: the file is written using the putc() stdio macro. The loop that does the writing should be small enough to fit into any reasonable I-cache. The CPU overhead here is that required to do the stdio code plus the OS file space allocation;
    - Block: the file is created using write(2). The CPU overhead should be just the OS file space allocation;
    - 1.3 Rewrite: each Chunk (currently, the size is 16384) of the file is read with read(2), dirtied, and rewritten with write(2), requiring an lseek(2). Since no space allocation is done, and the I/O is well-localized, this should test the effectiveness of the filesystem cache and the speed of data transfer.
- Sequential Input:
    - Per-Character: the file is read using the getc() stdio macro. Once again, the inner loop is small. This should exercise only stdio and sequential input;

- Block: the file is read using read(2). This should be a very pure test of sequential input performance.

- Random Seeks: this test runs SeekProcCount (currently 4) processes in parallel, doing a total of 4000 lseek()s to locations in the file computed using by random() in bsd systems, drand48() on sysV systems. In each case, the block is read with read(2). In 10% of cases, it is dirtied and written back with write(2).

4.2.3 IOzone

IOzone [23] is a filesystem benchmark tool. The benchmark generates and measures a variety of file operations. Iozone has been ported to many machines and runs under many operating systems.

Iozone is useful for performing a broad filesystem analysis of a vendor's computer platform. The benchmark tests file I/O performance for the following operations: read, write, re-read, re-write, read backward, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write. Figure 32 shows an example of a report of a read performance test.

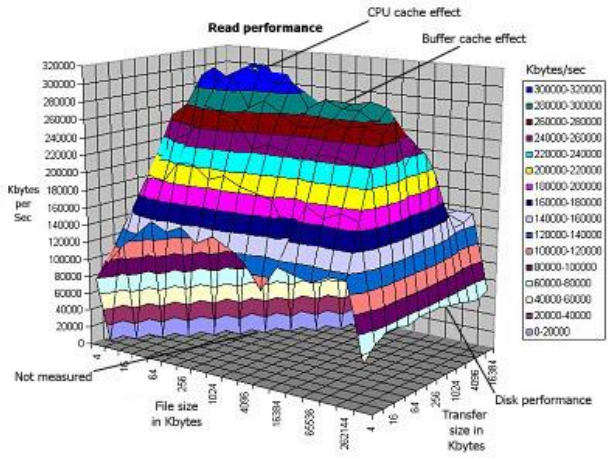

Figure 32: IOzone Read Performance Report Example

While computers are typically purchased with an application in mind it is also likely that over time the application mix will change. Many vendors have enhanced their operating systems to perform well for some frequently used applications. Although this accelerates the I/O for those few applications it is also likely that the system may not perform well for other applications that were not targeted by the operating system. An example of this type of enhancement is Database. Many operating systems vendors have tested and tuned the filesystem so it works well with databases. While the database users are happy, the other users may not be so happy as the entire system may be giving all of the system resources to the database users at the expense of all other users. As time rolls on the system administrator may decide that a few more office automation tasks could be shifted to this machine. The load may now shift from a random reader application (database) to a sequential reader.

4.2.4 FIO

FIO [24] is an I/O tool meant to be used both for benchmark and stress/hardware verification. It has support for 19 different types of I/O engines (sync, mmap, libaio, posixaio, SG v3, splice, null, network, syslet, guasi, solarisaio, and more), I/O priorities (for newer Linux kernels), rate I/O, forked or threaded jobs, and much more. It can work on block devices as well as files. FIO accepts job descriptions in a simple-to-understand text format. Several example job files are included. FIO displays all sorts of I/O performance information, including complete IO latencies and percentiles. FIO is in wide use in many places, for both benchmarking, QA, and verification purposes. It supports Linux, FreeBSD, NetBSD, OpenBSD, OS X, OpenSolaris, AIX, HP-UX, Android, and Windows.

4.2.5 Filebench

Filebench is a file system and storage benchmark that can generate a large variety of workloads. Unlike typical benchmarks, it is extremely flexible and allows to specify the application's I/O behavior using its extensive Workload Model Language (WML). Users can either describe desired workloads from scratch or use (with or without modifications) workload personalities shipped with Filebench (e.g., mail, web, file, and database server workloads). Filebench is equally good for micro and macro benchmarking, quick to setup, and relatively easy to use.

4.2.6 Inotify

Inotify (inode notify) [25] is a Linux kernel subsystem that acts to extend filesystems to notice changes to the filesystem and report those changes to applications. It replaces an earlier facility, dnotify, which had similar goals.

Inotify was created by John McCutchan, and it was merged into the Linux kernel mainline in kernel version 2.6.13, released on August 29, 2005; later kernel versions included further improvements. The required library interfaces were added into the GNU C Library (glibc) in its version 2.4, released in March 2006, while the support for inotify was completed in glibc version 2.5, released in September 2006.

One major use is in desktop search utilities like Beagle, where its functionality permits reindexing of changed files without scanning the filesystem for changes every few minutes, which would be very inefficient.

Inotify can also be used to automatically update directory views, reload configuration files, log changes, backup, synchronize, and upload.

The inotify API provides a mechanism for monitoring filesystem events. Inotify can be used to monitor individual files or to monitor directories. When a directory is monitored, inotify will return events for the directory itself, and for files inside the directory.

With careful programming, an application can use inotify to efficiently monitor and cache the state of a set of filesystem objects. However, robust applications should allow for the fact that bugs in the monitoring logic or races of the kind described below may leave the cache inconsistent with the filesystem state. It is probably wise to do some consistency checking, and rebuild the cache when inconsistencies are detected.

4.3 Benchmark Tools Comparison

In the previous sections, we discussed the features that need to be evaluated in a distributed file system benchmark and the tools that may be helpful for testing. This section discusses the test tools described and which ones can help us in the benchmark.

The table in Figure 33 shows a comparison of the described benchmark tools.

Name	Typology
Ganglia	Monitoring tool
Bonnie	I/O test tool
IOzone	Filesystem benchmark tool
FIO	I/O test tool
Filebench	File system and storage benchmark tool
Inotify	Low level file system monitoring tool

Figure 33: Benchmark tools

Chapter 5: Benchmark Plan and Experimental Results

In this work, we started with a description of the cloud world and we focused on the cloud storage analysis, in particular on hyperconvergence systems that can support a cloud environment. We've seen that there are many use cases that require storage support and various ways to implement it.

You can use the local file system, but this strongly limits what you can do with the cloud. A traditional distributed file system, on the other hand, tries to overcome the lack of local file systems through the use of a network architecture but has numerous limitations that the distributed file systems exceed.

For this reason, in this work, we focused on the analysis of distributed file systems. We started with a whole set of technologies and we chose to focus on four of them: HDFS, Ceph, GlusterFS, and XtreemFS.

After the analysis of their characteristics on some fundamental points, we want to see their behavior during execution and then have a complete final comparison. In this chapter, we want to describe the final benchmark plan and then present the results obtained by HDFS, Ceph, GlusterFS, and XtreemFS.

Before describing the tests on the DFSs, we will briefly describe the practical steps for choosing the tools used.

5.1 Tests on Benchmark Tools

The issues to be tested in a Distributed File System have been listed in the previous section. Many of the features do not need a practical phase to be analyzed, but some of them need practical tests and some tools may be helpful or not in structuring these tests. For better support during the practical test phase, we have analyzed some tools to evaluate their use in this work.

Ganglia is a monitoring system developed to be scalable and distributed. It allows you to monitor a large cluster through special services.

Bonnie performs a series of tests on a file of known size. It allows to evaluate the read and write performance of a given file system through the use of files with random content.

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

FIO is an I/O tool meant to be used both for benchmark and stress/hardware verification. It has support for 19 different types of I/O engines. It can work on block devices as well as files.

Filebench is a file system and storage benchmark that can generate a large variety of workloads. Unlike typical benchmarks, it is extremely flexible and allows to specify the application's I/O behavior using its extensive Workload Model Language (WML).

Inotify (inode notify) is a Linux kernel subsystem that acts to extend filesystems to notice changes to the filesystem and report those changes to applications. It replaces an earlier facility, dnotify, which had similar goals.

The tools have been tested to evaluate their usability in our tests. The table in Figure 34 shows the different use cases in which a tool can be used.

Name	Typology	Use Cases
Ganglia	Monitoring tool	Used for monitoring load, RAM, network, etc. on a cluster of many hosts.
Bonnie	I/O test tool	Allows testing of I/O operations in a file system, not recommended for DFS.
IOzone	Filesystem benchmark tool	It allows testing of I/O operations in a file system and is particularly used on NFS.
FIO	I/O test tool	It is used to Stress a file system and for Hardware verification.
Filebench	File system and storage benchmark tool	It is extremely flexible and allows to specify I/O behavior using WML.
Inotify	Low level file system monitoring tool	Allows you to monitor everything that happens on an inode by listening to the kernel events.

Figure 34: Benchmark Tools Comparison

For the benchmark, we need tools for distributed file systems, which allowed us to discard tools for different types of file systems. The intention of this work is to verify the behavior of the three technologies described above in various use cases; to do this, a host monitoring tool and a tool for monitoring changes within a file system are required.

For host machines monitoring, we chose Ganglia as it allows us to have a database on a server that stores all data and because the web interface it provides is very useful.

To monitor file system changes, however, we preferred using a low-level tool that allows us to monitor all events that occur on a precise inode, for this reason, we preferred Inotify.

5.2 Benchmark Plan

In the previous chapter are described evaluation criteria used to compare distributed file systems and some tools that can be useful for monitoring, performance measurement, and stress. It remains to be decided how the various operations must be carried out.

It is easy to understand that not all these criteria need a practical test phase, but some of these are to be considered purely theoretical. For example, deployment modes or client applications for a distributed file system are merely descriptive, while performance analysis or time propagation changes are evaluation criteria that need a practical test phase.

The comparative analysis is divided into two parts:

- Theoretical phase: a first part of the theoretical differences;
- Practical phase: a second part where the account is taken of measurable differences with appropriate instruments.

5.2.1 Theoretical Phase

The first part of the benchmark is a comparison of the features offered by the various technologies. For sources, as described above, is used the official documentation that can be found on the site of each of them.

In this section, the following points will be evaluated:

- POSIX semantics compatibility: it will be evaluated if a given distributed file system provides support for use through POSIX standard APIs;
- Usability: the various applications (if any) to support the client-side file system are displayed;
- Extensibility: describes the way which the distributed file system adds a new node and how it changes the architecture;
- Consistency of data: it will be evaluated as the distributed file system guarantees (or does not support) consistency (for example if checksum or other algorithms are used);
- Snapshots and Backup Support: Describes the snapshot support and backup format of the distributed file system and the support it provides to the consistency of the system by focusing on aspects of crash-consistency and application-consistency;
- Fault Tolerance: it will be evaluated how to support uninterrupted use, replication modes and fault tolerance support;
- Striping: it will be evaluated striping modes;
- Caching: it will be evaluated the supported caching methodologies;
- Integration with other systems: it will be evaluated the support that a DFS offers to other complex systems;
- Supported platforms: it will be evaluated the supported platforms by a DFS;
- Security: it will be evaluated the security models used.

5.2.2 Practical Phase

The second part of the benchmark deals a practical phase that measures the most important indicators of distributed file system. This section describes how some of the tools outlined above can offer test support.

The first step is to install the monitoring system of the Ganglia on server-side machines and on machines used to simulate the end users. Ganglia will monitor each CPU load node, RAM occupied, used storage, and bandwidth.

Before the description of the tests to be performed, it is necessary to take some preliminary steps.

Finding ourselves in a distributed situation may be a situation where the clocks of the various cluster machines are not properly synchronized, so for most accurate results, a first synchronization of the clocks via NTP is needed.

Secondly, according to the structure of the benchmark tool, it is necessary to enable ssh password less access in the machine from which you want to start the script.

The last thing to do is download the source tools of the benchmark tool developed in this work on all machines and add the bin folder path to it.

Once the preliminary operations are complete, it is possible to start the testing operations.

We intend to further subdivide this part into the following sub-sections to facilitate its description and implementation:

- Deployment Time: the estimated time required for installing the distributed file system is measured from the launch of the installation to the start of all daemons;

- Writes time: represent the time that needs to the system to write some files described below. It is measured on all nodes where replicas are placed;
- Writes propagation times: represent the time that needs to the system for the writes propagation. It is the time required for DFS to have two consistent replicas and three consistent replicas;
- System load with multiple clients: represents the system load in a situation where the number of clients is variable and tends to bring the system to a saturation state;
- Fault tolerance system behavior: this involves causing a system failure (such as dropping a node) and seeing how the system responds to this;
- Node re-balancing time: represent the time taken by a distributed file system to return its state to a consistent situation following the addition or removal of a node;
- Cluster Monitoring: represent the monitoring results during all the writes of files described below.

Below it is possible to see in detail the operations we intend to perform at each point and how it is possible to implement the individual operations. It is important to remember that during all of the following operations, the Ganglia tool will be running so you can know the situation of cluster health indicators at any time.

In tests where it is necessary to use files, we will use the files described in the table in Figure 35. For the test *a-1-10MB* we use one file of 10MB, for the test *b-1-1GB* one file of 1GB, for the test *c-1-10GB* one file of 10GB, for the test *d-100-1GB* hundred files of 1GB (approximately 10MB per file), for the test *e-100-10GB* hundred files of 10GB (approximately 100MB per file) and for the test *f-clients-10-10-10GB* we used ten files for each of the ten clients that are simulated (every file's size is generated randomly between 50MB and 150MB).

Test Name	Number of Files	Total Dimension
a-1-10MB	1	10MB
b-1-1GB	1	1GB
c-1-10GB	1	10GB
d-100-1GB	100	1GB
e-100-10GB	100	10GB
f-clients-10-10-10GB	10 for each of the 10 clients	10GB

Figure 35: File used during tests

In the first point, we are going to measure the deployment time. We know that HDFS, Ceph, GlusterFS, and XtreemFS are installed with an automatic deployment and this means that no additional time for human operations is needed.

In the second point, we are going to measure the propagation time of files in Figure 31 in cluster machines. To do this, we use the logs of Inotify tool.

In the third point, we are going to measure the system load in a situation where 10 clients write simultaneously on the DFS. For this test, we use the files of *f-clients-10-10-10GB* test.

In the fourth point, we are going to measure the behavior of DFS in a situation where a node goes down.

Finally, in the fifth point, we are going to measure the re-balancing time after a fault. For this test, we use the files of *f-clients-10-10-10GB* test.

5.2.3 Benchmark Table

We have described the Benchmarking Plan and the tests we will be doing in the two phases. For convenience below (Figure 36) is a summary table of the tests.

Name	Phase	Description
POSIX semantics compatibility	Theoretical Phase	Support of distributed file system to use POSIX standard APIs.
Usability	Theoretical Phase	Description of the various applications (if any) to support the client-side file system.
Extensibility	Theoretical Phase	Description of the way which the distributed file system adds a new node and how it changes the architecture.
Consistency	Theoretical Phase	Description of how the distributed file system guarantees consistency.
Snapshots and Backup Support	Theoretical Phase	*Description of the snapshot and backup support.*
Fault Tolerance	Theoretical Phase	Description of how to support uninterrupted use, replication modes and fault tolerance support.
Striping	Theoretical Phase	Description of various striping modes.
Caching	Theoretical Phase	Description of supported caching methodologies.
Integration with other systems	Theoretical Phase	Description of integration with other complex systems.

Supported Platforms	Theoretical Phase	Description of supported platforms.
Security	Theoretical Phase	Description of security methods.
Deployment Time	Practical Phase	Measurement of the deployment time.
Writes time	Practical Phase	Represent the time that needs to the system to write some files described below. It is measured on all nodes where replicas are placed.
Writes propagation time	Practical Phase	Measurement of the time that needs for the writes propagation.
System load with multiple clients	Practical Phase	Measurement of the system load in a situation where the number of clients is variable.
Fault tolerance system behavior	Practical Phase	The behavior of the system after a system failure.
Node re-balancing time	Practical Phase	Measurement of the time taken by a distributed file system to return in a consistent situation.
Cluster Monitoring	Practical Phase	Represent the monitoring results during all the writes of files described below.

Figure 36: Benchmark Table

5.3 Experimental Results

The Distributed File Systems present a lot of features that need to be tested. In previous sections, it is described all these features in details and it is presented the benchmark plane of this work.

In this chapter, before describing the experimental results, we present the structure of the cluster used for the benchmark and how we have installed the distributed file systems on them.

Once described the cluster, we are going to present the experimental results of this work. The results are split according to the phases described above, so first will be presented the results of the theoretical phase and then those of the practical phase.

5.3.1 Cluster Structure

When you want to test a distributed file system, you must use a cluster structure that is adapted to what you want to test. In this work, what we want to test is the replication, fault tolerance and performance of HDFS, Ceph, GlusterFS, and XtreemFS.

To do this, four machines were used: three to host distributed file systems server-side and one to simulate clients and for monitoring. The machines have been connected to a 100MB/s network.

The table in Figure 37 shows the description of the machines.

Name	Description
Processor Name	Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz
Processor Speed	1700 MHz
Execution Technology	6/6 cores; 6 threads
Memory Technology	64-bit Capable
RAM	40 GB 1866 MHz
HDD	2TB (WD Red Pro 2TB NAS Hard Disk Drive - 5400 RPM Class SATA 6 Gb/s 64MB Cache 3.5 Inch - WD20EFRX)
Tipologia di memoria computer	DDR3 SDRAM

Figure 37: Machines description

Figure 38 shows the cluster structure.

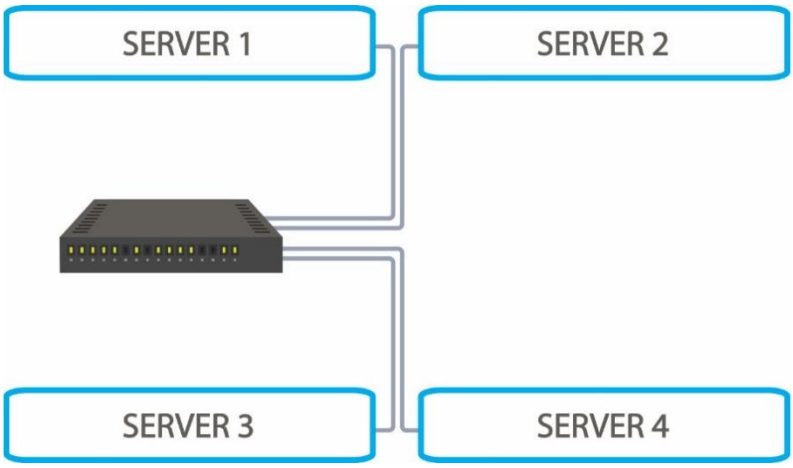

Figure 38: Cluster Structure

After describing the cluster at a physical level, we will briefly describe the architecture that was used for the three different distributed file systems.

Hadoop Distributed File System

For HDFS, we used three nodes to install the DFS and the fourth node to simulate clients and for Ganglia server. Remember that all the servers have a Ganglia monitor installed. Below there is the list of components installed on each server:

- Server 1: HDFS NameNode, HDFS DataNode and Ganglia monitor (gmond);
- Server 2: HDFS DataNode and Ganglia monitor (gmond);
- Server 3: HDFS DataNode and Ganglia monitor (gmond);
- Server 4: Clients and Ganglia Server (gmetad).

Figure 39 shows the HDFS architecture used for the benchmark.

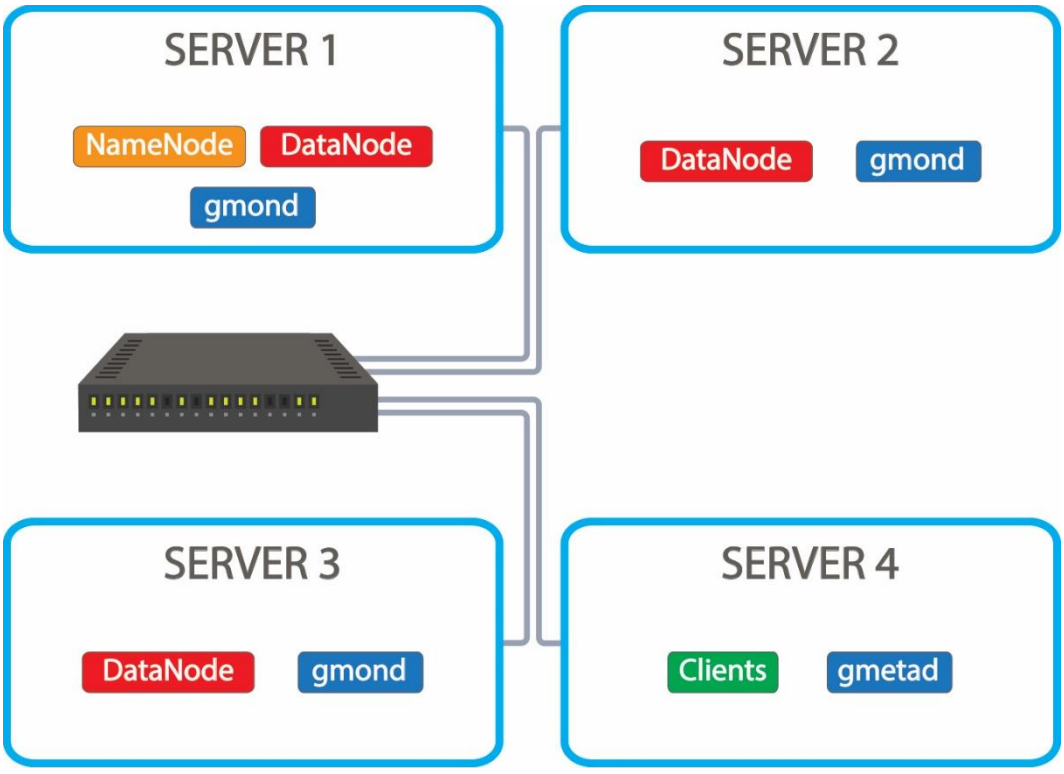

Figure 39: HDFS Benchmark Architecture

Ceph

For Ceph, we used three nodes to install the DFS and the fourth node to simulate clients and for Ganglia server. Below there is the list of components installed on each server:

- Server 1: Ceph Monitor, Ceph OSD and Ganglia monitor (gmond);
- Server 2: Ceph Monitor, Ceph OSD and Ganglia monitor (gmond);
- Server 3: Ceph Monitor, Ceph OSD and Ganglia monitor (gmond);
- Server 4: Clients and Ganglia Server (gmetad).

Figure 40 shows the Ceph architecture used for the benchmark.

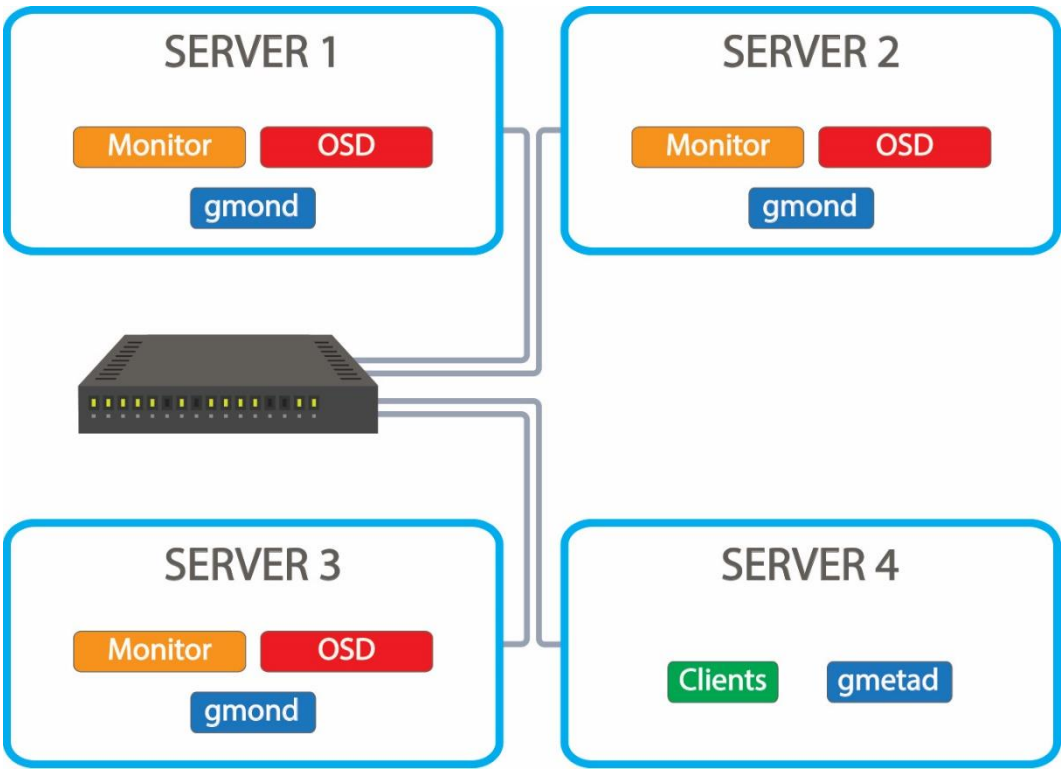

Figure 40: Ceph Benchmark Architecture

GlusterFS

For GlusterFS, we used three nodes to install the DFS and the fourth node to simulate clients and for Ganglia server. Below there is the list of components installed on each server:

- Server 1: GlusterFS server and Ganglia monitor (gmond);
- Server 2: GlusterFS server and Ganglia monitor (gmond);
- Server 3: GlusterFS server and Ganglia monitor (gmond);
- Server 4: Clients and Ganglia Server (gmetad).

Figure 41 shows the GlusterFS architecture used for the benchmark.

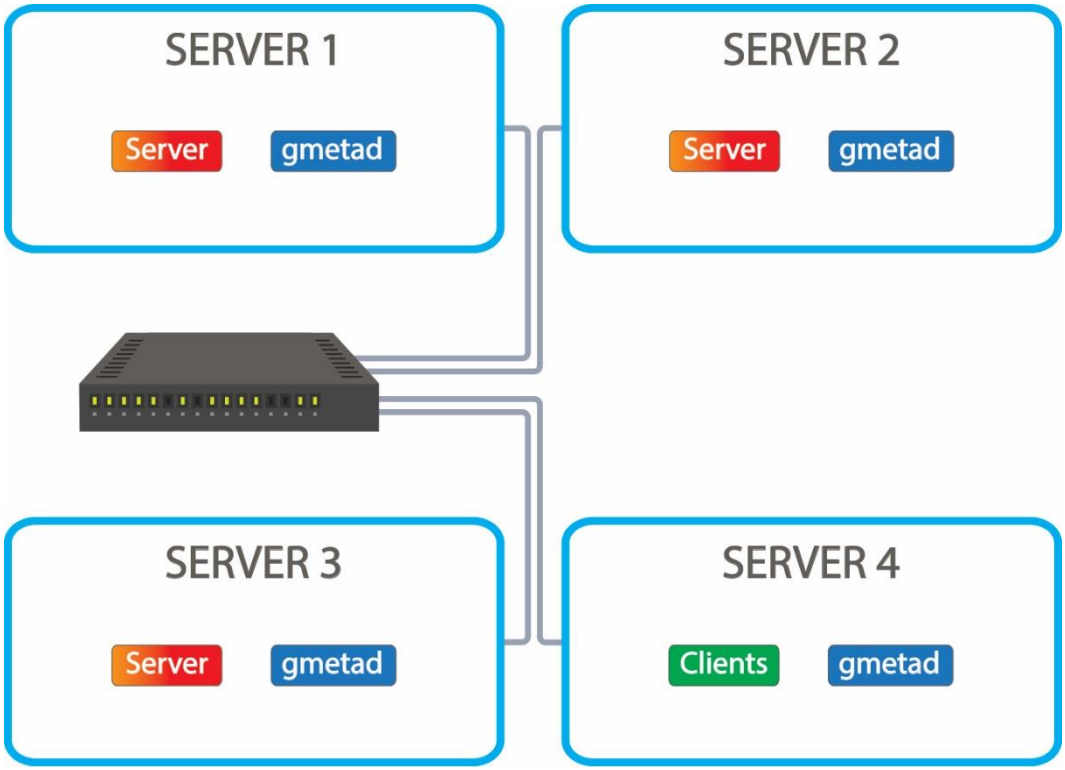

Figure 41: GlusterFS Benchmark Architecture

XtreemFS

For XtreemFS, we used three nodes to install the DFS and the fourth node to simulate clients and for Ganglia server. Below there is the list of components installed on each server:

- Server 1: XtreemFS MRC, XtreemFS OSD and Ganglia monitor (gmond);
- Server 2: XtreemFS OSD and Ganglia monitor (gmond);
- Server 3: XtreemFS RMS, XtreemFS OSD and Ganglia monitor (gmond);
- Server 4: Clients and Ganglia Server (gmetad).

Figure 42 shows the XtreemFS architecture used for the benchmark.

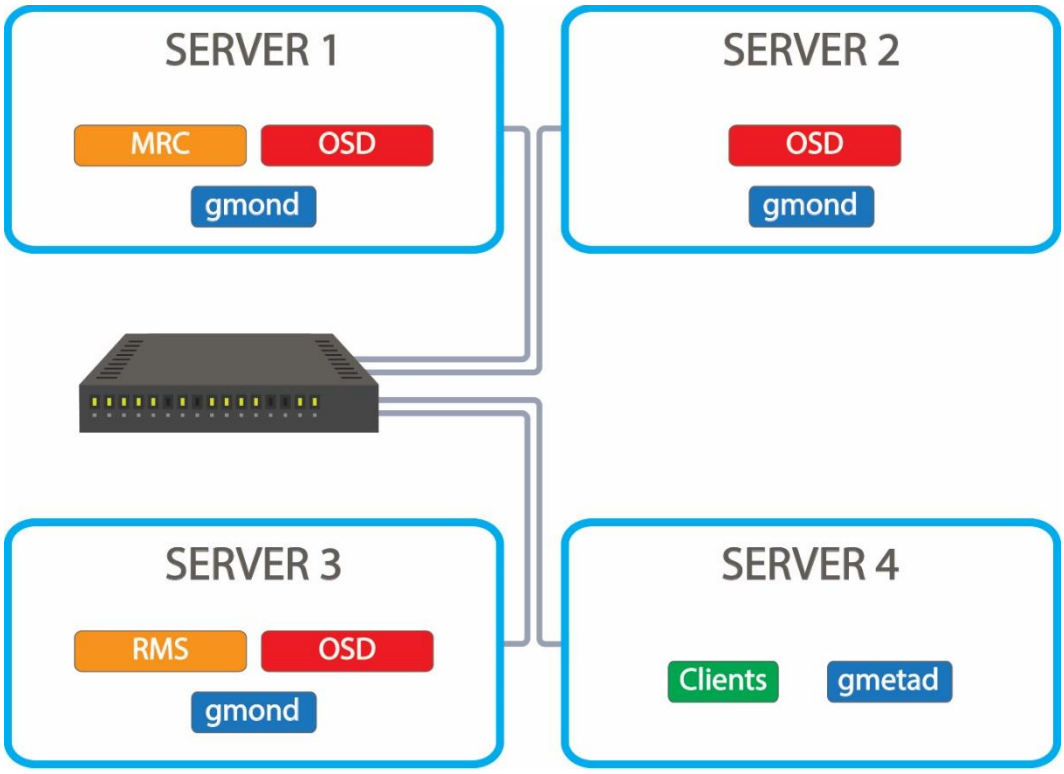

Figure 42: XtreemFS Benchmark Architecture

5.3.2 Theoretical Phase

The first part we analyze concerns the theoretical phase of the benchmark. This phase will show features that do not require a practical test phase. For each of the features described in the test plan, HDFS, Ceph, GlusterFS, and XtreemFS will be compared.

POSIX semantics compatibility

The POSIX system guarantees the portability of a given interface for data access on the various systems. A distributed file system that provides compatibility with POSIX semantics ensures that data can be replicated on multiple devices and that applications can access them without having to change the operating system.

HDFS is not POSIX compatible. Append-only semantics is the best-known exception, but only the tip of the iceberg. For example, it also seems to lack support for extended attributes and doesn't honor POSIX durability semantics. The people who use HDFS, and the way that they use it, do not create a strong need for POSIX compliance.

CephFS aims to adhere to POSIX semantics wherever possible. For example, in contrast to many other common network file systems like NFS, CephFS maintains strong cache coherency across clients. The goal is for processes communicating via the file system to behave the same when they are on different hosts as when they are on the same host. However, there are a few places where CephFS diverges from strict POSIX semantics for various reasons. For example, if a client is writing to a file and fails, its writes are not necessarily atomic. That is, the client may call write on a file opened with O_SYNC with an 8 MB buffer and then crash and the write may be only partially applied (almost all file systems, even local file systems, have this behavior). Another example is in shared simultaneous writer situations because a

write that crosses object boundaries is not necessarily atomic. This means that you could have writer A write “aa|aa” and writer B write “bb|bb” simultaneously (where | is the object boundary), and end up with “aa|bb” rather than the proper “aa|aa” or “bb|bb”. The file-system provides also a POSIX interface named CephFS using an experimental native Linux driver written by Linus Torvalds and integrated into 2.6.34 kernel or using a more stable Fuse-based solution.

GlusterFS is a fully POSIX compliant file system. It implements the POSIX APIs using the POSIX Translator and POSIX Access Control Lists (ACLs). ACLs allows you to assign different permissions for different users or groups even though they do not correspond to the original owner or the owning group. For example User john creates a file but does not want to allow anyone to do anything with this file, except another user, Antony (even though there are other users that belong to the group john). This means, in addition to the file owner, the file group, and others, additional users, and groups can be granted or denied access by using POSIX ACLs.

XtreemFS has the same "Look&Feel" as NFS or your local file system. Applications and users won't see a difference, also for replicated and remote files. You can use XtreemFS as a drop-in replacement for NFS.

Usability

Usability is defined as the effectiveness, efficiency, and satisfaction with which certain users reach certain goals in certain contexts. It defines the degree of satisfaction with which the interaction between man and a system is performed. In a distributed file system, usability can be seen in client applications.

HDFS provides a code library that allows users to read, write and delete files and create and delete directories. Users access a file using its path in the namespace and contact the NameNode to know where to retrieve

or put files' blocks, and then request the transfer by communicating directly with the DataNodes. This is done in a transparent way using some API in C, Java or REST.

Ceph Clients include a number of service interfaces including Block Device, Object Storage, and Filesystem. The Ceph Block Device provides resizable, thin-provisioned block devices with snapshotting and cloning. Ceph stripes a block device across the cluster for high performance and supports both kernel objects (KO) and a QEMU hypervisor that uses librbd directly–avoiding the kernel object overhead for virtualized systems. The Ceph Object Storage provides RESTful APIs with interfaces that are compatible with Amazon S3 and OpenStack Swift. The Ceph Filesystem provides a POSIX compliant filesystem usable with mount or as a filesystem in user space (FUSE).

GlusterFS is a userspace filesystem. Being a userspace filesystem, to interact with kernel VFS, GlusterFS makes use of FUSE (File System in Userspace).

XtreemFS supports the use of FUSE (File System in Userspace).

Extensibility

Most distributed file systems allow you to scale horizontally and not vertically. This means that you can add a node to the architecture to be able to extend the cluster.

HDFS provides scalability in a very simple way. Once we have reached the saturation of our cluster or before it, just add a new node to the cluster (a new DataNode) and the system will automatically rebalance the data between the old nodes and the new node.

In Ceph it is possible to add monitors and OSDs. When you add a monitor, the system automatically updates the cluster map and provides to update the new monitor with the information needed to handle the interactions. When you add an OSD, Ceph automatically re-balance the

data according to the parameters set for each placement group, so what is changing is only the location of the data, but this is transparent to the clients.

In GlusterFS, it is possible to add new bricks to an existing volume. To do this, just prepare the new machine and format the brick before adding it. During the addition of brick, you have to consider the type of volume as it is not always enough to add a single brick. For example, in a replicated volume must be added a number of brick multiple of the replication factor.

In XtreemFS it is possible to add OSDs. When you add an OSD, XtreemFS re-balance the data according to the parameters set for each replication factor.

Consistency

The consistency guarantees to the user that the data are significant and actually usable within their applications. When analyzing a distributed file system, consistency takes on some importance when you need to maintain consistency between the data in the file system and the data coming from outside. Consistency in a file system means that following various operations you can continue to ensure data integrity by avoiding any loss or error on the files.

HDFS is Eventually Consistent, that means they can take time for changes to objects (during an operation of creation, deletion or updates) to become visible to all clients. There is no guarantee a change is immediately visible to the client which just made the change.

In Ceph, reads must reflect an up-to-date version of data and writes must reflect a particular order of occurrences. It allows concurrent read and write accesses using a simple locking mechanism that gives users the ability to read, read and cache, write or write and buffer data. Ceph

also provides some tools to relax consistency, for example, it is possible to allow users to read a file even if it is currently rewritten.

Consistency in GlusterFS is achieved with the use of a translator. A translator converts requests from users into requests for storage can modify requests on the way through convert one request type to another modify paths, flags, even data. The Automatic File Replication (AFR) translator maintains replication consistency, data on both the bricks should be same, even in the cases where there are operations happening on same file/directory in parallel from multiple applications/mount points as long as all the bricks in replica set are up.

XtreemFS provides the user with a consistent view of the entire file system. The consistency of replicas is handled by the OSDs in a distributed manner. To handle consistency issues, it is used a leases mechanism to make sure that only a single OSD is able to modify a specified part of the data at a given time (single writer, multiple readers).

Snapshots and Backup Support

A snapshot represents an object's state at a given instant of time. The term, in computer science, is used to represent the state of a system at a specific time. Its use is related to backup and restore aspects. The idea is that you can have multiple snapshots of the same and you can bring the system back to a previous state.

HDFS Snapshots are read-only point-in-time copies of the file system. Snapshots can be taken on a subtree of the file system or the entire file system. Some common use cases of snapshots are data backup, protection against user errors and disaster recovery.

Ceph supports the creation of snapshots and allows you to keep the status of images. A Ceph clone is a read-only copy of the active image. However, before you create a snapshot, it is important to stop I/O

operations to prevent an inconsistent state of the snapshot. Ceph, also, supports the creation of copy-on-write (COW) snapshots clones, which allows you to create clones quickly without having to use the primary instance. Each cloned image stores a reference to the main snapshot, which allows the cloned image to open the main snapshot and read it.

GlusterFS snapshots are thinly provisioned LVM based snapshots, and hence they have certain pre-requisites. Considering a volume with two bricks, that are mounted on independent, thinly-provisioned LVMs. When the volume is mounted, the client process maintains a connection to both the bricks. A GlusterFS snapshot, is also internally a GlusterFS volume with the exception that, it is a read-only volume and it is treated differently than a regular volume in certain aspects.

XtreemFS is capable of taking file system snapshots. Snapshots are exposed as read-only volumes. To access a snapshot, it is necessary to mount it. A mounted volume snapshot can be browsed normally, and all files can be read as in the original volume. However, any attempt to write data on a snapshot will result in an error.

Fault Tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause a total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems. The ability to maintain functionality when portions of a system break down is referred to as graceful degradation.

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures, and network partitions. In case of disk

failure or network partitions, we must take into account that each DataNode sends a heartbeat message to the NameNode periodically. The NameNode detects this condition by the absence of a heartbeat message and marks DataNodes without recent heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary.

To achieve reliability and fault tolerance, Ceph supports a cluster of monitors. In a cluster of monitors, latency and other faults can cause one or more monitors to fall behind the current state of the cluster. For this reason, Ceph must have agreement among various monitor instances regarding the state of the cluster. Ceph always uses a majority of monitors and the Paxos algorithm to establish a consensus among the monitors about the current state of the cluster. To detect failures, OSDs periodically exchange heartbeats.

GlusterFS replace the fault tolerance of RAID with replication allows you to spread that vulnerability between 2 or more complete servers. With multipath routing using two or more network cards, it is possible to eliminate most single points of failure between clients and data. If a server or network connection goes down, the client will continue to operate with the remaining servers.

XtreemFS provides fault tolerance and scalability by supporting replication and striping of files. File replica management can be automated by means of a replica management service. A replicated and partitioned metadata service ensures that metadata is highly available and metadata operations can be performed quickly

Striping

Striping is useful when a processing device requests data more quickly than a single storage device can provide it. By spreading segments across multiple devices which can be accessed concurrently, total data throughput is increased. It is also a useful method for balancing I/O load across an array of disks. Striping is used across disk drives in a redundant array of independent disks (RAID) storage, network interface controllers, different computers in clustered file systems and grid-oriented storage, and RAM in some systems.

In HDFS, striping concept is connected to Erasure Coding (EC) concept. In storage systems, the most notable usage of EC is Redundant Array of Inexpensive Disks (RAID). RAID implements EC through striping, which divides logically sequential data into smaller units and stores consecutive units on different disks. For each stripe of original data cells, a certain number of parity cells are calculated and stored. In the context of EC, striping has several critical advantages. First, it enables online EC (writing data immediately in EC format), avoiding a conversion phase and immediately saving storage space. Online EC also enhances sequential I/O performance by leveraging multiple disk spindles in parallel; this is especially desirable in clusters with high-end networking. Second, it naturally distributes a small file to multiple DataNodes and eliminates the need to bundle multiple files into a single coding group. This greatly simplifies file operations such as deletion, quota reporting, and migration between federated namespaces. Striped HDFS files are logically composed of block groups, each of which contains a certain number of internal blocks. To reduce NameNode memory consumption from these additional blocks, a new hierarchical block naming protocol was introduced. The client read and write paths were enhanced to work on multiple internal blocks in a block group in parallel. On the output/write path, DFSStripedOutputStream manages a set of data streamers, one for each DataNode storing an internal block in the current block group. The DataNode runs an additional

ErasureCodingWorker task for background recovery of failed erasure coded blocks. Failed EC blocks are detected by the NameNode, which then chooses a DataNode to do the recovery work. The recovery task is passed as a heartbeat response. This process is similar to how replicated blocks are re-replicated on failure.

The RAID type most similar to Ceph's striping is RAID 0, or a 'striped volume.' Ceph's striping offers the throughput of RAID 0 striping, the reliability of n-way RAID mirroring and faster recovery. A Ceph Client converts its data from the representation format it provides to its users (a block device image, RESTful objects, CephFS filesystem directories) into objects for storage in the Ceph Storage Cluster. The simplest Ceph striping format involves a stripe count of 1 object. Ceph Clients write stripe units to a Ceph Storage Cluster object until the object is at its maximum capacity, and then create another object for additional stripes of data. The simplest form of striping may be sufficient for small block device images, S3 or Swift objects and CephFS files. However, this simple form doesn't take maximum advantage of Ceph's ability to distribute data across placement groups and consequently doesn't improve performance very much. If you anticipate large images sizes, large S3 or Swift objects, or large CephFS directories, you may see considerable read/write performance improvements by striping client data over multiple objects within an object set. Significant write performance occurs when the client writes the stripe units to their corresponding objects in parallel. Since objects get mapped to different placement groups and further mapped to different OSDs, each write occurs in parallel at the maximum write speed. By spreading that write over multiple objects (which map to different placement groups and OSDs) Ceph can reduce the number of seeks per drive and combine the throughput of multiple drives to achieve much faster write (or read) speeds.

In GlusterFS when using files that exceed the size of your bricks, or when using a small number of large files that have I/O operations done

to them in random seek locations, the stripe may be a good fit. Files stored on a striped volume should be throw-away as there's no data integrity and a single brick failure will lose all the data. By using stripe plus replication it is possible to offer improved read performance over stripe alone, as well as system redundancy and data integrity security. This should still only be used with over-brick-sized files or large files with random I/O.

XtreemFS supports striping of file content. Different parts of a striped file are handled by different OSDs. Striping can substantially increase read/write throughput, since read and write requests for a single file can be processed by multiple OSDs in parallel. The client ensures that read and write requests for a file are translated to read and write requests at object granularity. Depending on the striping pattern, object requests are then transmitted to the respective OSDs.

Caching

As part of processing, the cache is a high-speed data storage layer that stores a subset of data, typically temporary, to respond to requests more quickly than it would not be possible to access each time in the main path where the data is located. Caching enables you to efficiently reuse used data that has already been recovered or processed. Cache data is generally stored in quick access hardware, such as RAM, and can be processed with the help of a software component.

The main purpose of the cache is to increase data recovery performance by reducing the need for access to the slower, lower storage layer. Cache sacrifices speed capability, so it can usually store only a temporary subset of data, while databases store data in a complete and durable manner.

In HDFS cache is called Centralized cache management in HDFS that is an explicit caching mechanism that allows users to specify paths to

be cached by HDFS. The NameNode will communicate with DataNodes that have the desired blocks on disk, and instruct them to cache the blocks in off-heap caches. Explicit pinning prevents frequently used data from being evicted from memory. Because DataNode caches are managed by the NameNode, applications can query the set of cached block locations when making task placement decisions. Co-locating a task with a cached block replica improves read performance. When the block has been cached by a DataNode, clients can use a new, more-efficient, zero-copy read API. Since checksum verification of cached data is done once by the DataNode, clients can incur essentially zero overhead when using this new API. Centralized caching can improve overall cluster memory utilization. When relying on the OS buffer cache at each DataNode, repeated reads of a block will result in all n replicas of the block being pulled into the buffer cache. With centralized cache management, a user can explicitly pin the only m of the n replicas, saving n-m memory.

In Ceph it is possible to use cache tiering. A cache tier provides Ceph Clients with better I/O performance for a subset of the data stored in a backing storage tier. Cache tiering involves creating a pool of relatively fast/expensive storage devices (e.g., solid state drives) configured to act as a cache tier, and a backing pool of either erasure-coded or relatively slower/cheaper device configured to act as an economical storage tier. The Ceph object handles where to place the objects and the tiering agent determines when to flush objects from the cache to the backing storage tier. So, the cache tier and the backing storage tier are completely transparent to Ceph clients.

In GlusterFS you can use cache in a volume. The brick that has the file is called the “cached subvolume”. Normally, it is predicted by the distributed hash’s algorithm, unless the set of bricks has recently changed. Subsequent LOOKUPs are sent only to the cached subvolumes. Regardless of this phenomenon, the cached sub volume still receives as many LOOKUPs as the path length, due to the path

traversal problem. So, when the test is run a second time, gluster profile still shows 20K LOOKUPs, but only on bricks on the hot tier (the tier translator's cached sub volume), and nearly none on the cold tier. The round trips are still there, and the overall problem persists. To cope with this "lookup amplification", a project has been underway to improve GlusterFS meta-data cache translator (md-cache), so the stat information LOOKUP requests could be cached indefinitely on the client. This solution requires client-side cache entries to be invalidated if another client modified a file or directory. The invalidation mechanism is called an "upcall".

In addition to regular file replication, XtreemFS provides read-only replication. This replication mode works on immutable files and supports a large number of replicas. The read-only replication helps you quickly build a caching infrastructure on top of XtreemFS in order to reduce latency and bandwidth consumption between datacenters. In addition to full replicas that contain a complete copy, XtreemFS also supports partial replicas. These replicas are filled on demand when a client accesses data.

Integration with other systems

A distributed file system can offer some integration capabilities with other systems. Here is the possibility of using DFS by another complex system.

HDFS offers the ability to use its own storage in other applications. Its frequent use is in the big data analysis. Many Apache Spark applications exploit the ability to use HDFS as input and output data storage.

Ceph, in addition to offering an easily accessible distributed file system, is widely used in cloud environments. OpenStack, one of the most popular cloud environments, can use Ceph as internal storage.

GlusterFS, such as HDFS and Ceph, offers many possibilities of use for different applications. Its uses are obviously related to the type of volume used. Frequent use is to support VM storage for QEMU.

XtreemFS provides a client-side file system through FUSE, so it's possible to integrate with all the tools that need a file system, such as a mount point for a container.

Supported Platforms

Another point of interest is the supported platforms. This point evaluates how much a distributed file system is really multiplatform. A particular DFS can run on different operating systems (Windows, Linux or Mac) or not and this is a big difference if you want to build your own storage system on multiple platforms.

HDFS can be used on many platforms. In order to run, Java support is sufficient, so it is supported by Windows, Mac, and Linux.

Ceph, however, has some limitations over HDFS. As a general rule, it is possible to deploy Ceph on newer releases of Linux, but it is recommended to deploy Ceph on releases with long-term support.

GlusterFS, if you are using NFS/CIFS supports the following OSs client-side: all Fedora and Debian based distributions, UNIX (Solaris 10+), AIX, HP-UX, and some version of Microsoft Windows Server.

XtreemFS can be used on many platforms. It is possible to download the distributed file system for Windows, Linux, and Mac or use directly the source code in C++/Java.

Security

Computer security is very important in computer science. Typically, in a distributed system, some client applications access the file system. These applications can be used with or without access. For greater security, the various DFSs support different methodologies to ensure security.

HDFS, by default, runs in a non-secure mode in which no actual authentication is required. By configuring Hadoop runs in secure mode, each user and service needs to be authenticated by Kerberos in order to use Hadoop services. Security features of Hadoop consist of authentication, service level authorization, authentication for Web consoles and data confidentiality.

A key scalability feature of Ceph is to avoid a centralized interface to the Ceph object store, which means that Ceph clients must be able to interact with OSDs directly. To protect data, Ceph provides its cephx authentication system, which authenticates users operating Ceph clients. The cephx protocol operates in a manner with behavior similar to Kerberos.

GlusterFS allows its communication to be secured using the Transport Layer Security standard (which supersedes Secure Sockets Layer), using the OpenSSL library. Setting this up requires a basic working knowledge of some SSL/TLS concepts.

To enforce security, XtreemFS offers mechanisms for user authentication and authorization, as well as the possibility to encrypt network traffic. Authentication describes the process of verifying a user's or client's identity. By default, authentication in XtreemFS is based on local usernames and depends on the trustworthiness of clients and networks. In case a more secure solution is needed, X.509 certificates can be used. Authorization describes the process of checking user permissions to execute an operation. XtreemFS supports the standard UNIX permission model, which allows for assigning

individual access rights to file owners, owning groups and other users. Authentication and authorization are policy-based, which means that different models and mechanisms can be used to authenticate and authorize users. Besides, the policies are pluggable, i.e. they can be freely defined and easily extended. XtreemFS uses unauthenticated and unencrypted TCP connections by default. To encrypt all network traffic, services and clients can establish TLS connections.

5.3.3 Practical Phase

In the second part of the benchmark, we focus on aspects that need a practical testing phase. In this section will be presented the results of the tests.

Deployment Time

Deployment Time is the time required for installing the distributed file system and it is measured from the launch of the installation to the start of all daemons. It must be remembered that this test, in order to be real, was performed through the use of automatic deployment tools.

This test shows that GlusterFS has much smaller installation times than HDFS, Ceph, and XtreemFS. Specifically, the time taken is:

- HDFS: 88 minutes;
- Ceph: 63 minutes;
- GlusterFS: 18 minutes;
- XtreemFS: 27 minutes.

This is probably due to the fact that GlusterFS needs many fewer packages to be able to work.

Figure 43 shows the deployment time of HDFS, Ceph, GlusterFS, and XtreemFS.

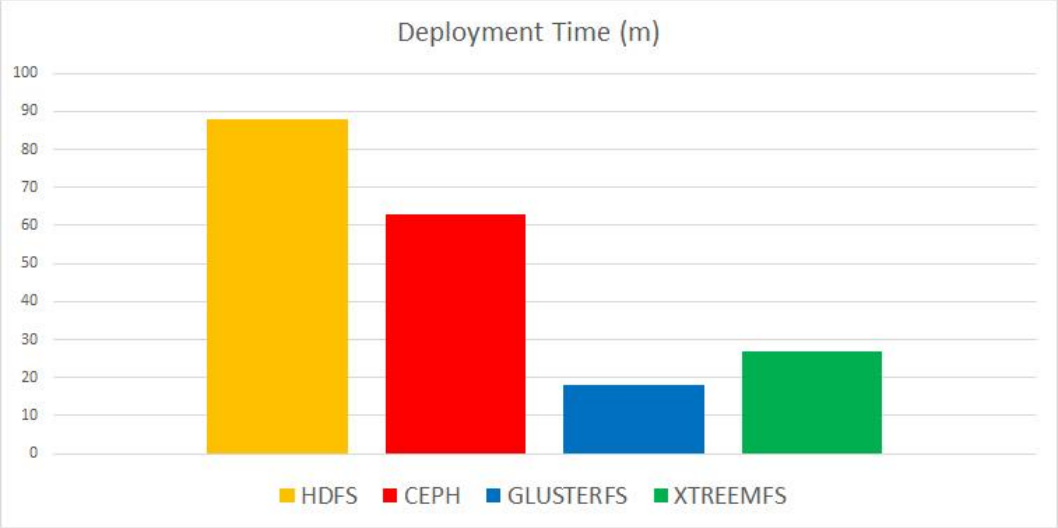

Figure 43: Deployment Time

Writes time

Write time represent the time that need to the system to write files in one node. It is measured on all nodes where replicas are placed.

The graphs below show the time required for DFS to write the files of the tests described above. For example, we suppose that at time t_0 DFS start writing file1 and at time t_1 end writing file1. The graphs show the millisecond between t_0 and t_1 that represents the write time of file1. Obviously, the time represented is the mean time of writing of multiple files over multiple servers.

Figure 44 shows the write time of the test *a-1-10MB*. It is possible to see that Ceph needs more time then HDFS, GlusterFS, and XtreemFS. This probably is due to the presence of Placement Groups that need to be updated. In HDFS, GlusterFS, and XtreemFS, instead, PGs do not exist so the times are smaller. It should be noted that the differences in times are few milliseconds, so they are not so great to cause long write time.

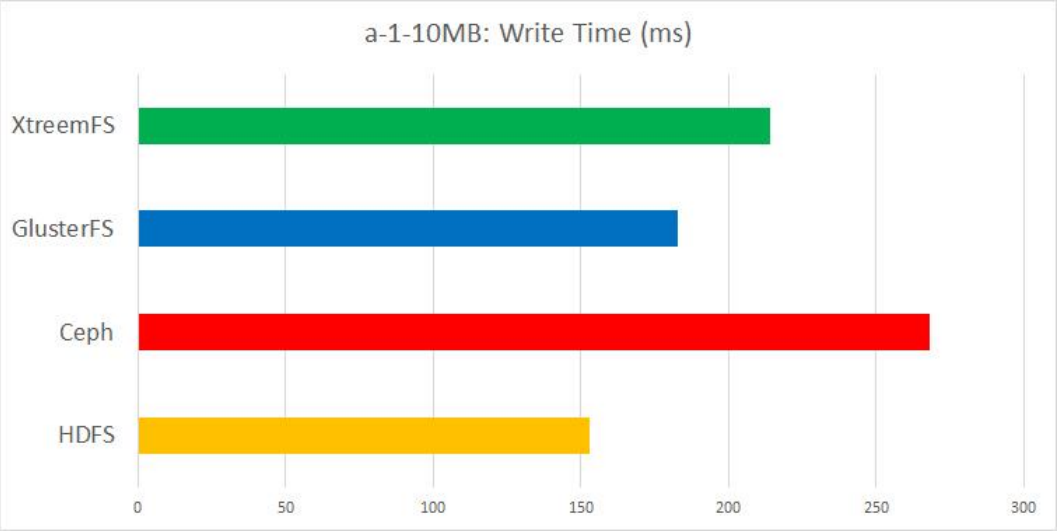

Figure 44: Write Time of test a-1-10MB

Figure 45 shows the write time of the test *b-1-1GB*. It is possible to see that HDFS use less time than Ceph, GlusterFS, and XtreemFS. This probably is due to different architectures of the DFSs. HDFS has a centralized point where are stored metadata and Ceph and GlusterFS have different points where are stored metadata. In HDFS and GlusterFS, also, PGs do not exist so the times are smaller than Ceph. XtreemFS has similar times to GlusterFS, the differences are few milliseconds. It should be noted that the differences in times are few milliseconds.

Figure 45: Write Time of test b-1-1GB

Figure 46 shows the write time of the test *c-1-10GB*. It is possible to see that HDFS use less time than Ceph, GlusterFS, and XtreemFS. This probably is due to different architectures of the DFSs as discussed in the previous test. HDFS architecture allows you to have better times, in addition, clients communicate directly with the storage node. It should be noted that the differences in times are few milliseconds.

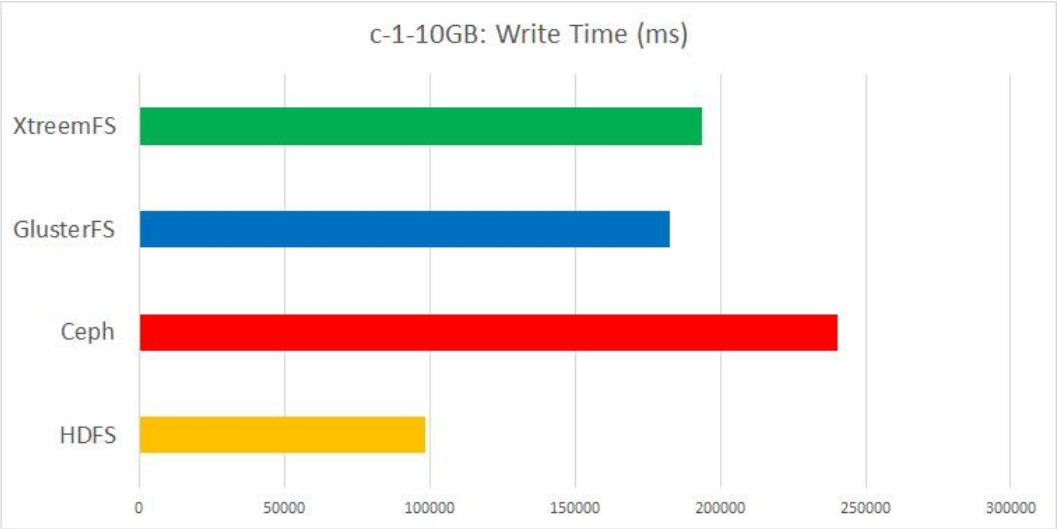

Figure 46: Write Time of test c-1-10GB

Figure 47 shows the write time of the test *d-100-1GB*. It is possible to see, as in previous tests, that Ceph has bigger write times then HDFS, GlusterFS, and XtreemFS. However, the GlusterFS and XtreemFS times are not as low as those of HDFS. This probably is due to the same reason of the previous test.

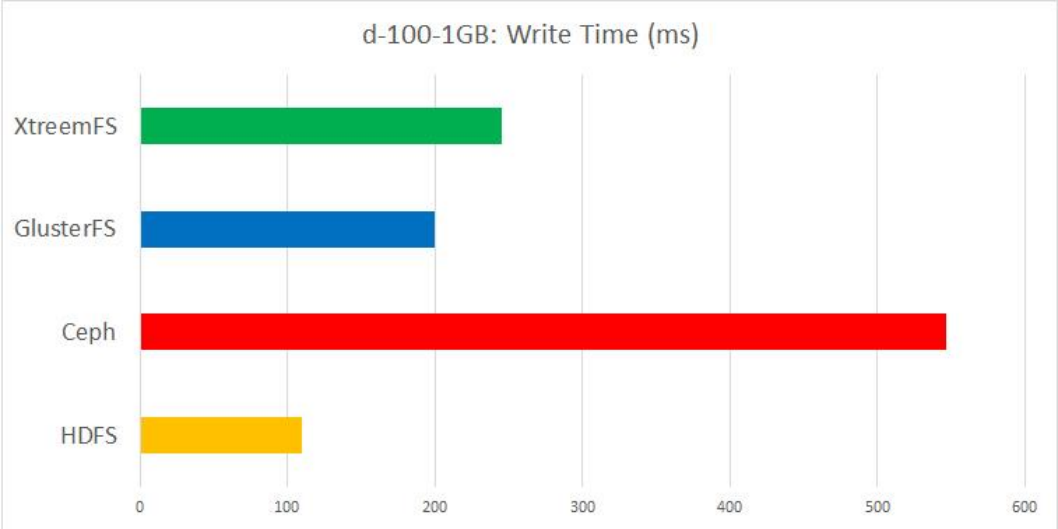

Figure 47: Write Time of test d-100-1GB

Figure 48 shows the write time of the test *e-100-10GB*. It is possible to see, as in previous tests, that HDFS has smaller write times then Ceph, GlusterFS, and XtreemFS. However, the Ceph times are not as low as those of GlusterFS and XtreemFS. This probably is due to the same reason of the previous test.

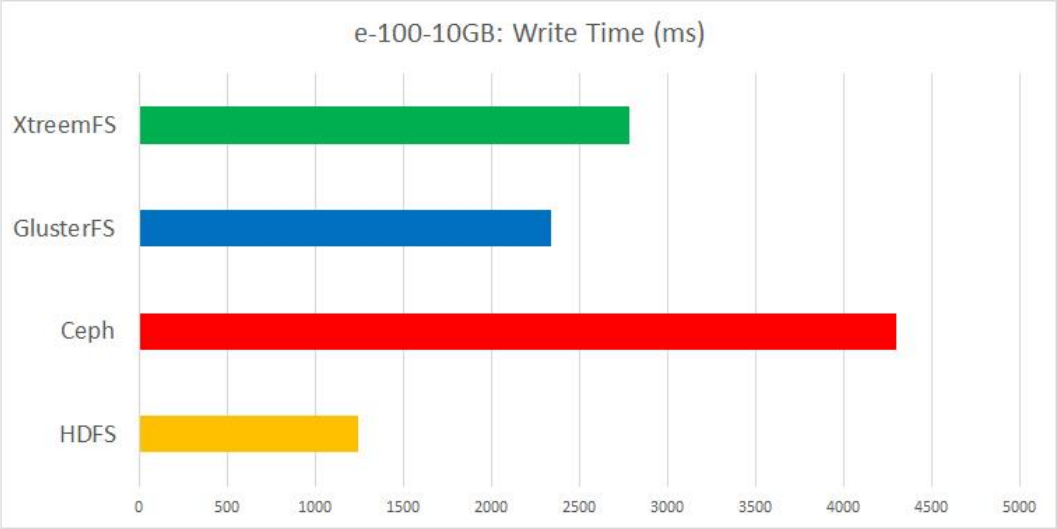

Figure 48: Write Time of test e-100-10GB

Writes propagation time

The using a DFS is closely related to its benefits. One of these is replication, that is the possibility to have copies of the same file on multiple machines and to ensure fault tolerance. One of the most important parameters is the write propagation time that is the time needed to have a consistent state of files on all servers.

The graphs below show the time required for DFS to have two consistent replicas and three consistent replicas. For example, we suppose that at time t_0 file1 is consistent on server1, at time t_1 file1 is consistent on server2 and at time t_2 file1 is consistent on server3. The graphs show the millisecond between t_0 and t_1 for the two consistent

replicas and the millisecond between t_0 and t_2 for the three consistent replicas.

Figure 49 shows the write propagation time of the test *a-1-10MB*. It is possible to see that Ceph has bigger propagation times then HDFS, GlusterFS, and XtreemFS. This probably is due to the presence of Placement Groups. When we write a file in Ceph, the system needs to pass through the level of PGs. In HDFS, GlusterFS, and XtreemFS, instead, this level does not exist so the times are smaller. It should be noted that the differences in times are few milliseconds, so they are not so great to cause big delays.

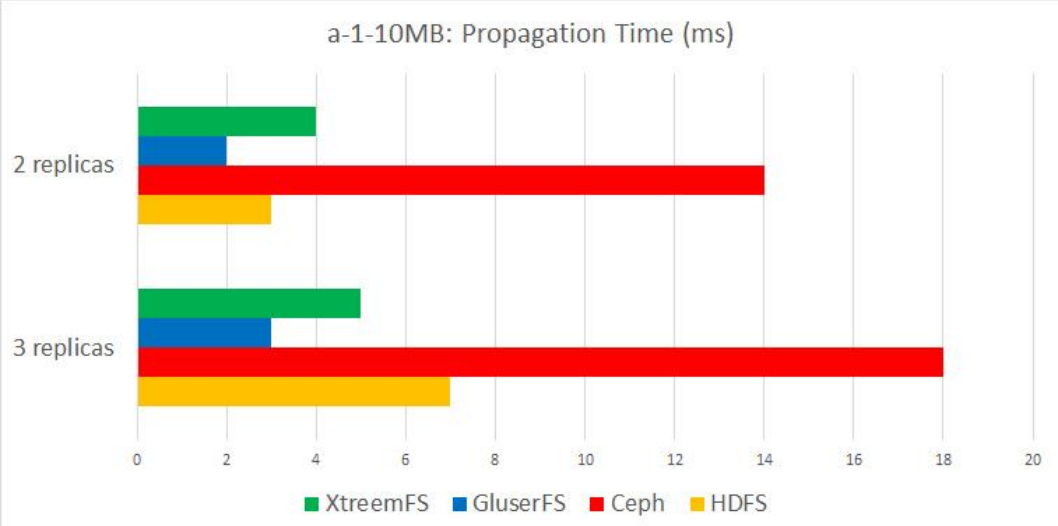

Figure 49: Write propagation Time of test a-1-10MB

Figure 50 shows the write propagation time of the test *b-1-1GB*. It is possible to see that Ceph has bigger propagation times then HDFS, GlusterFS, and XtreemFS. This probably is due to the same reason of the previous test.

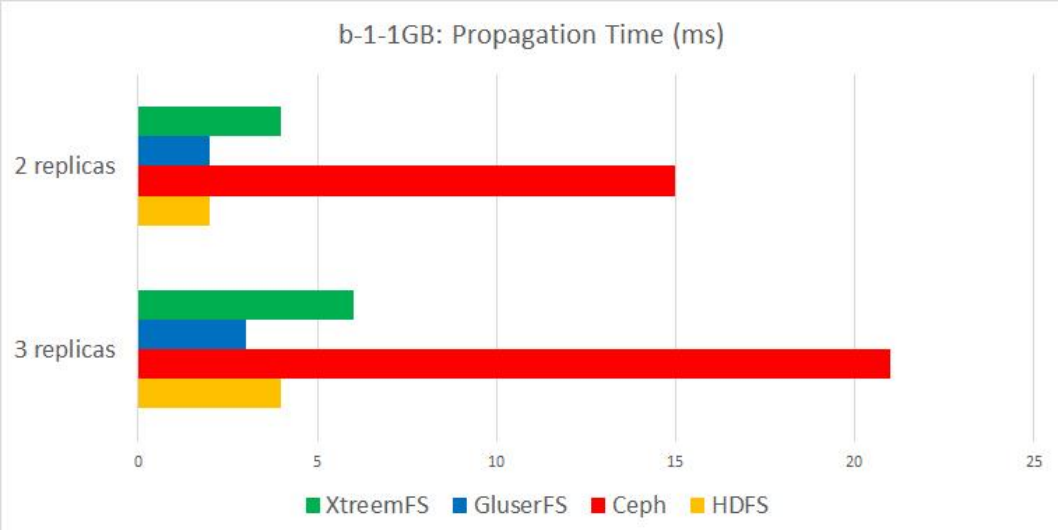

Figure 50: Write propagation Time of test b-1-1GB

Figure 51 shows the write propagation time of the test *c-1-10GB*. It is possible to see that Ceph has bigger propagation times then HDFS, GlusterFS, and XtreemFS. This probably is due to the same reason of the previous test.

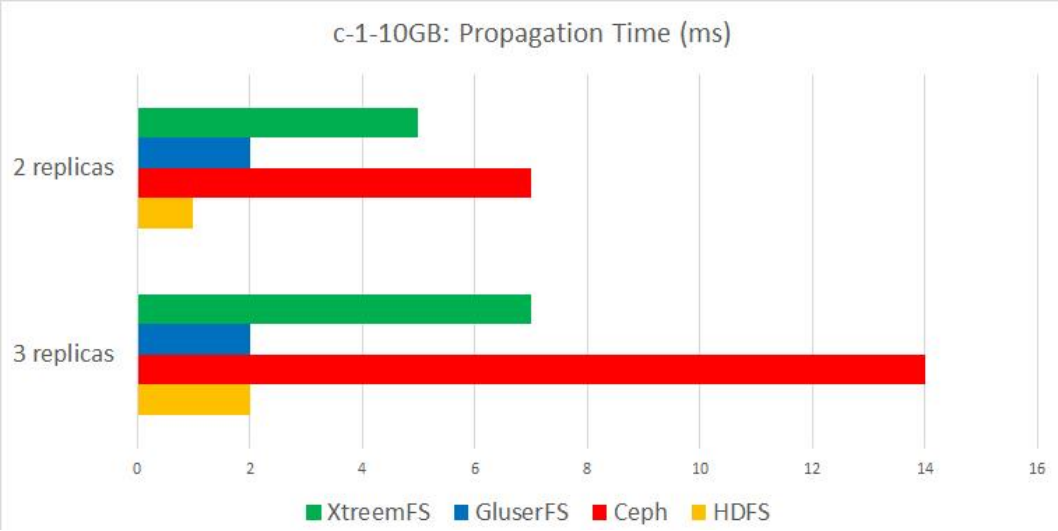

Figure 51: Write propagation Time of test c-1-10GB

Figure 52 shows the write propagation time of the test *d-100-1GB*. It is possible to see, as in previous tests, that Ceph has bigger propagation times then HDFS, GlusterFS, and XtreemFS. This probably is due to the same reason of the previous test.

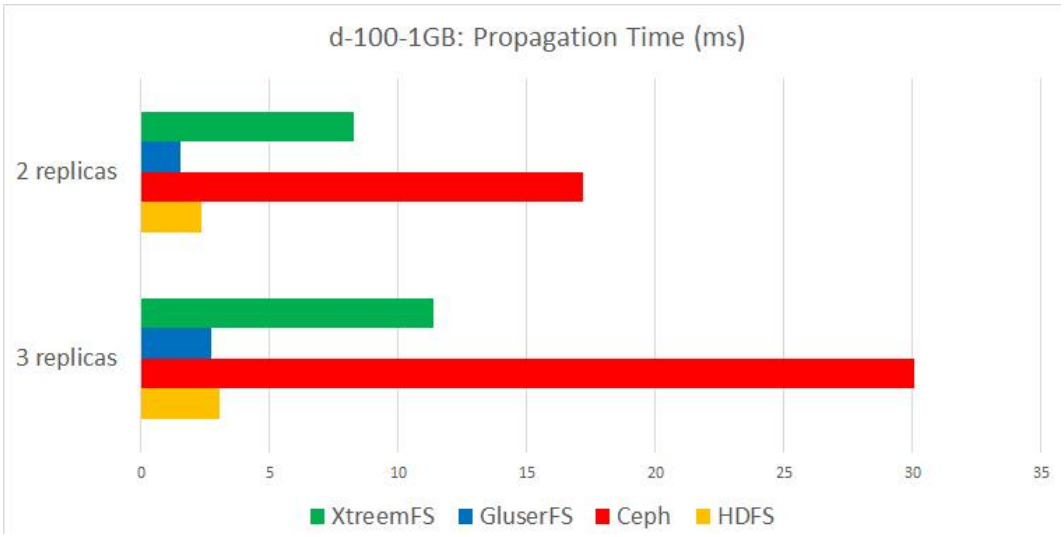

Figure 52: Write propagation Time of test d-100-1GB

Figure 53 shows the write propagation time of the test *e-100-10GB*. It is possible to see, as in previous tests, that Ceph has bigger propagation times then HDFS, GlusterFS, and XtreemFS. This probably is due to the same reason of the previous test.

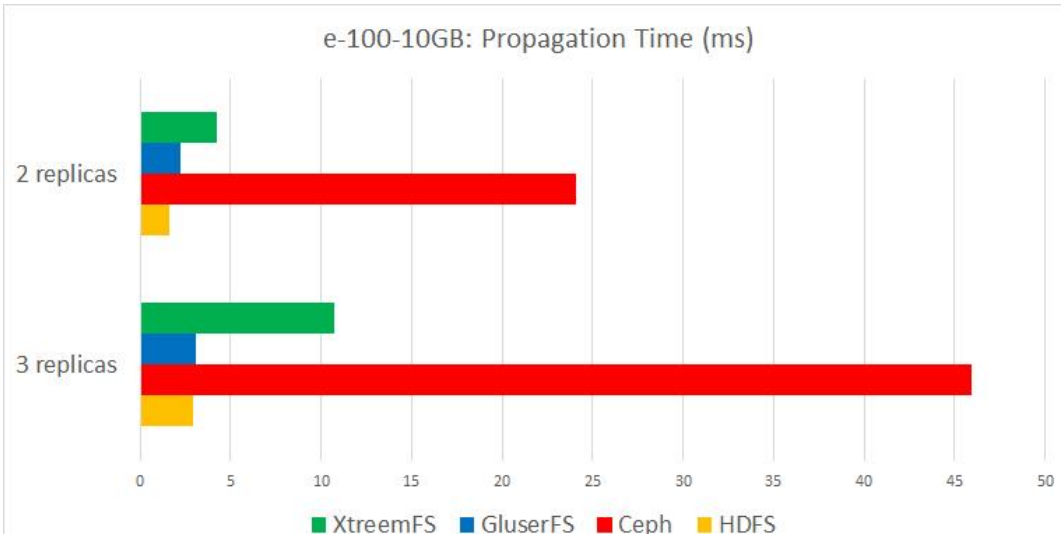

Figure 53: Write propagation Time of test e-100-10GB

This test shows that HDFS, GlusterFS, and XtreemFS achieve a consistent state in less time than Ceph. Ceph, however, has all the benefits of using the placement groups shown earlier. Moreover, as already mentioned, times are expressed in milliseconds, so Ceph's delay does not lead to long periods of non-consistent state.

System load with multiple clients

System load with multiple clients represents the status of the cluster in a situation where the number of clients is variable and tends to bring the system to a saturation state.

In this test, it is possible to see the differences between HDFS, Ceph, GlusterFS, and XtreemFS when 10 clients are writing 10 files of different sizes at the same time.

Figure 54 shows the write time of test *f-10-10-10GB*. It is possible to see the differences between the three DFSs during a phase of stress. Ceph, even in this test, has bigger time than HDFS and GlusterFS. GlusterFS, on the other hand, has higher times than the previous tests, so when there are more clients' connections GlusterFS has bigger write times. HDFS, on the other hand, always has the best write times. As mentioned above, this probably is due to different architectures of HDFS, Ceph, and GlusterFS. HDFS has a centralized point where are stored metadata (the NameNode) and Ceph and GlusterFS have different points where are stored metadata (in Ceph on monitors and in GlusterFS on peer). In HDFS and GlusterFS, also, Placement Groups do not exist so the times are smaller than Ceph. XtreemFS, instead, shows similar results to GlusterFS, but slightly higher.

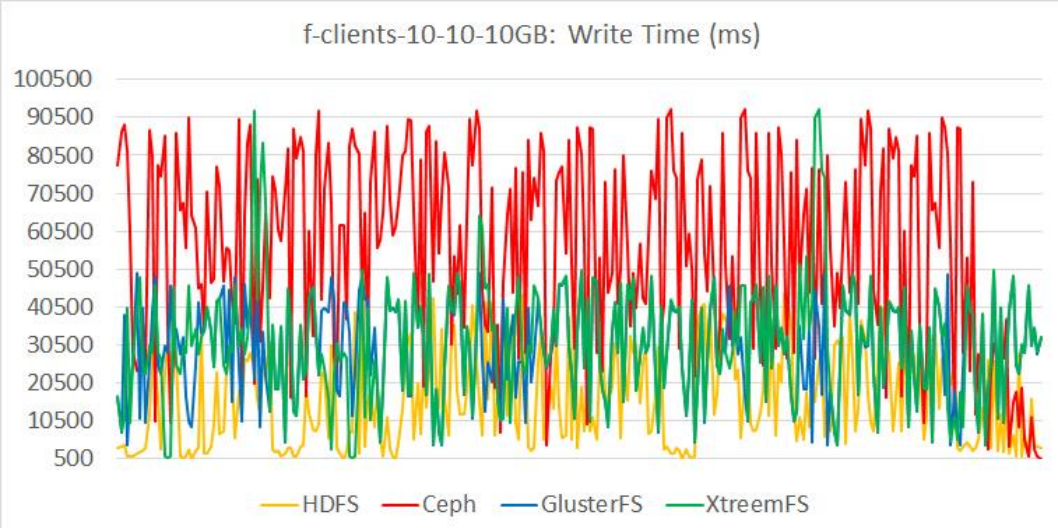

Figure 54: Write Time of test f-10-10-10GB

Figure 55 shows mean writing times of test *f-10-10-10GB*. In this graph, the differences in times are clearer than the previous one.

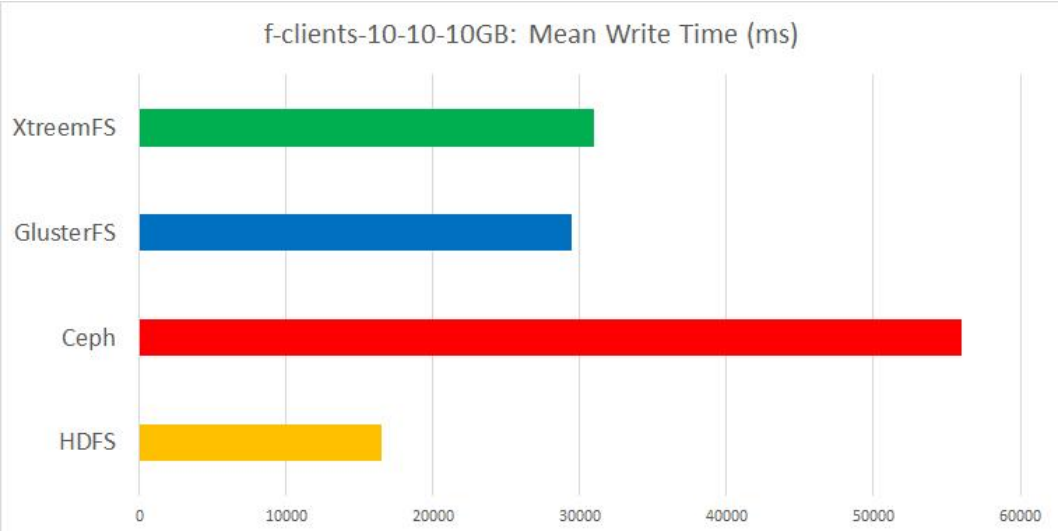

Figure 55: Mean Write Time of test f-10-10-10GB

Fault tolerance system behavior

Fault-tolerant technology is a capability of a system to deliver uninterrupted service, despite one or more of its components failing. Fault tolerance also resolves potential service interruptions related to software or logic errors. The purpose is to prevent failures that could result from a single point of failure.

In this test, we turned off a node to simulate a failure and see the behavior of the DFS. Before the fault, within the DFS there are the files of the *f-clients-10-10-10GB* test that will be used to analyze the rebalancing time in next test.

HDFS provides fault tolerance support as described above. After the node failure, the DFS has marked the node as unreachable and after about 10 minutes it started re-balancing process.

Ceph, on the other hand, was immediately labeled the monitor on the failed node as unreachable and the OSD as disconnected. The re-balancing process started after a few minutes.

GlusterFS immediately marked the peer of the failed node as disconnected but did not launch any re-balancing process because it is not in charge of the DFS, but user.

Finally, XtreemFS provides an automatic load balancing. When a node with an OSD is down, files are handled by other nodes and the Replication Management Service handles where to place the replicas.

Node re-balancing time

This test Measure the time taken by a distributed file system to return to a consistent state after a fault. In *Fault tolerance system behavior* test we turned off a node to simulate a failure and see the behavior of the DFS. In this test, we want to see how HDFS and Ceph provide to re-balance nodes. GlusterFS is not considered in this test because it does not provide an automatic re-balancing process. Remember that in the DFSs there are the files of the *f-clients-10-10-10GB*.

HDFS starts re-balancing process after 10 minutes and 30 seconds, during this time the NameNode tried to contact the unreachable DataNode. To reach a new consistent state, respecting the number of replicas, HDFS took 1 minute and 22 seconds. Then, after about 12 minutes, HDFS reported the DFS in an optimal state of operation.

Ceph, instead, starts re-balancing process after 2 minutes and 10 seconds. To reach a new consistent state, respecting the number of replicas in the placement groups, Ceph took 11 minute and 4 seconds. Then, after about 13 minutes, Ceph reported the DFS in an optimal state of operation.

XtreemFS starts re-balancing process after about 5 minutes. To reach a new consistent state, respecting the number of replicas, XtreemFS took about 10 minutes. Then, after about 15 minutes, XtreemFS reported the DFS in an optimal state of operation.

Figure 56 shows the re-balancing time of different DFSs.

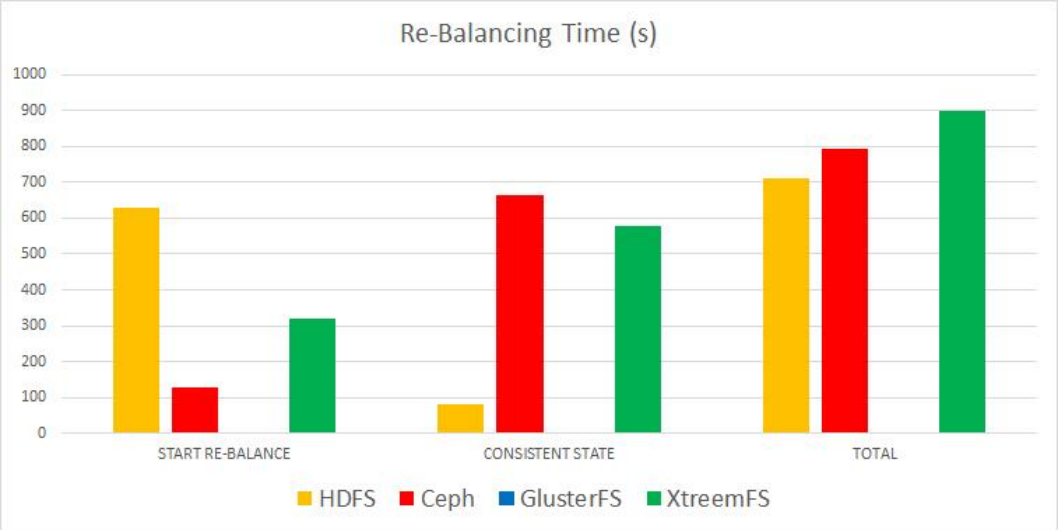

Figure 56: Re-balancing Time

Cluster Monitoring

Previously, we described the tests performed in the benchmark. During the entire execution of the benchmark, Ganglia was running and provided data on the state of the system. This section will show some of the most significant charts.

During the test *a-1-10MB* we upload a file of 10MB in HDFS, Ceph, GlusterFS, and XtreemFS. During this test the load of the system is minimum and the network has a small peak during file transmission. Figure 57 shows the situation of Ceph during this test. The graphs of HDFS and GlusterFS are not reported because are similar.

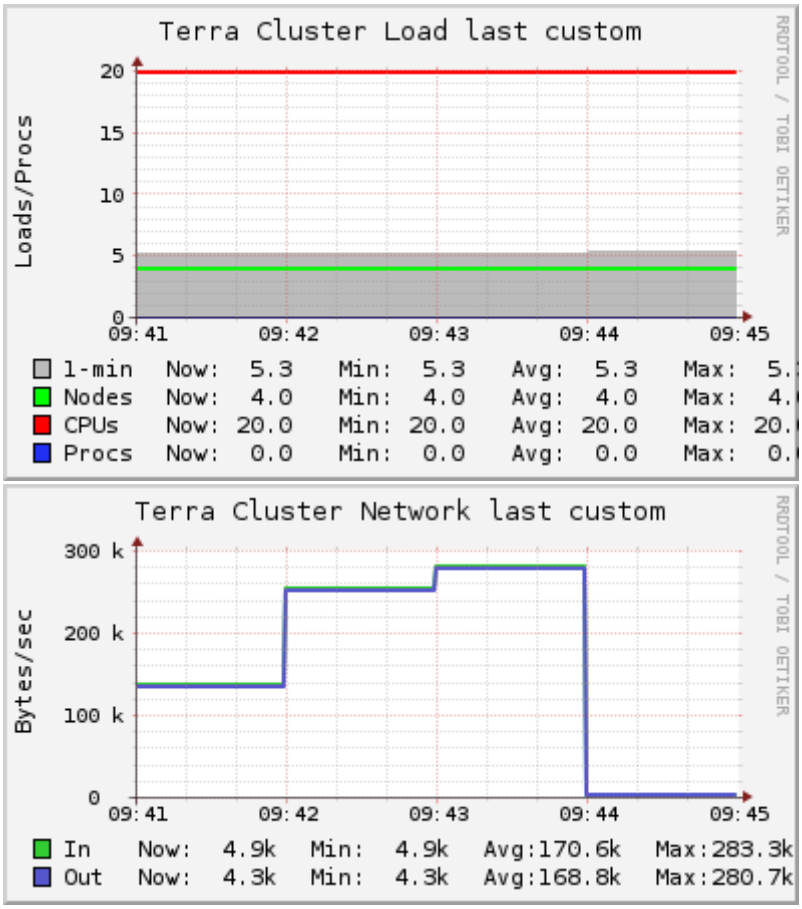

Figure 57: Cluster Load and Network during a-1-10MB test in Ceph

During the test *b-1-1GB* we upload a file of 1GB in HDFS, Ceph, GlusterFS, and XtreemFS. During this test, the load of the system grows in the middle of the transfer and the network has a peak longer than the previous one during file transmission. Figure 58 shows the situation of GlusterFS during this test. The graphs of HDFS and Ceph are not reported because are similar.

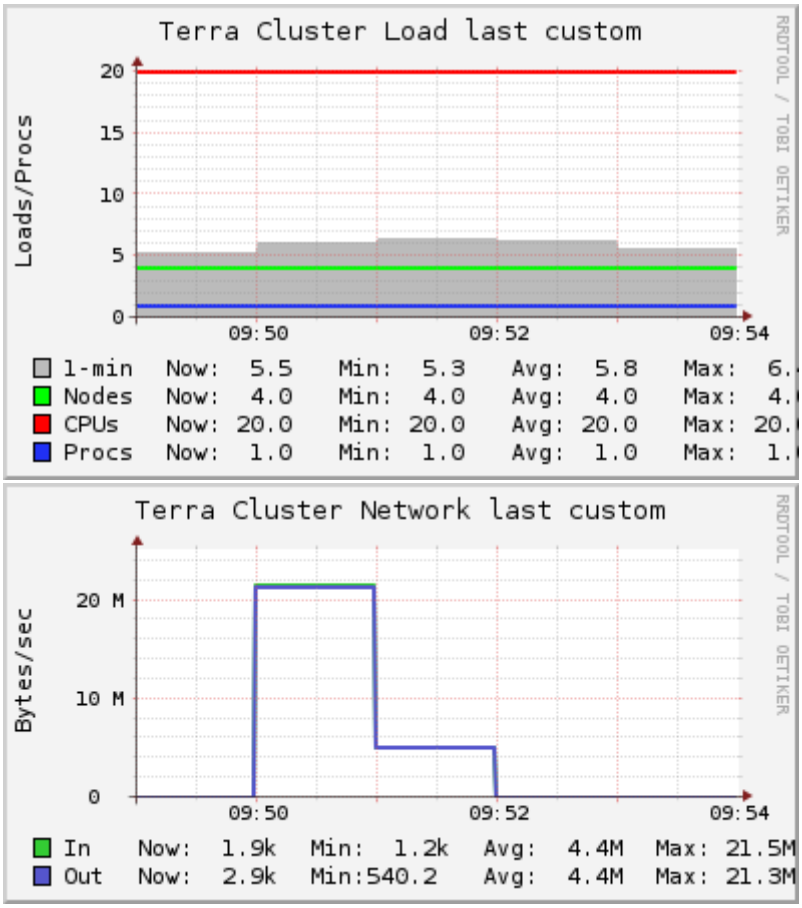

Figure 58: Cluster Load and Network during b-1-1GB test in GlusterFS

During the test *c-1-10GB* we upload a file of 10GB in HDFS, Ceph, GlusterFS, and XtreemFS. During this test, the load of the system grows more than previous tests and the network has a peak longer and bigger than the previous tests during file transmission. Figure 59 shows the situation of HDFS during this test. The graphs of Ceph and GlusterFS are not reported because are similar.

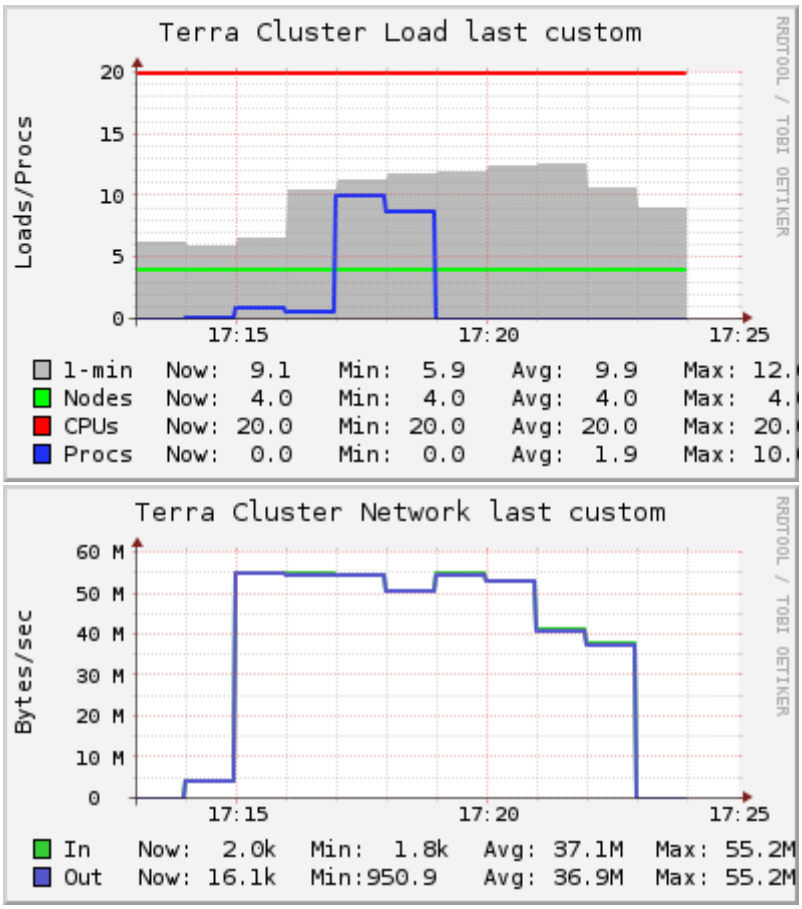

Figure 59: Cluster Load and Network during c-1-10GB test in HDFS

During the test *d-100-1GB* we upload 100 files of 1GB in HDFS, Ceph, GlusterFS, and XtreemFS. During this test the load of the system grows is constant and the network first has a peak and then an attenuation. Figure 60 shows the situation of GlusterFS during this test. The graphs of HDFS and Ceph are not reported because are similar.

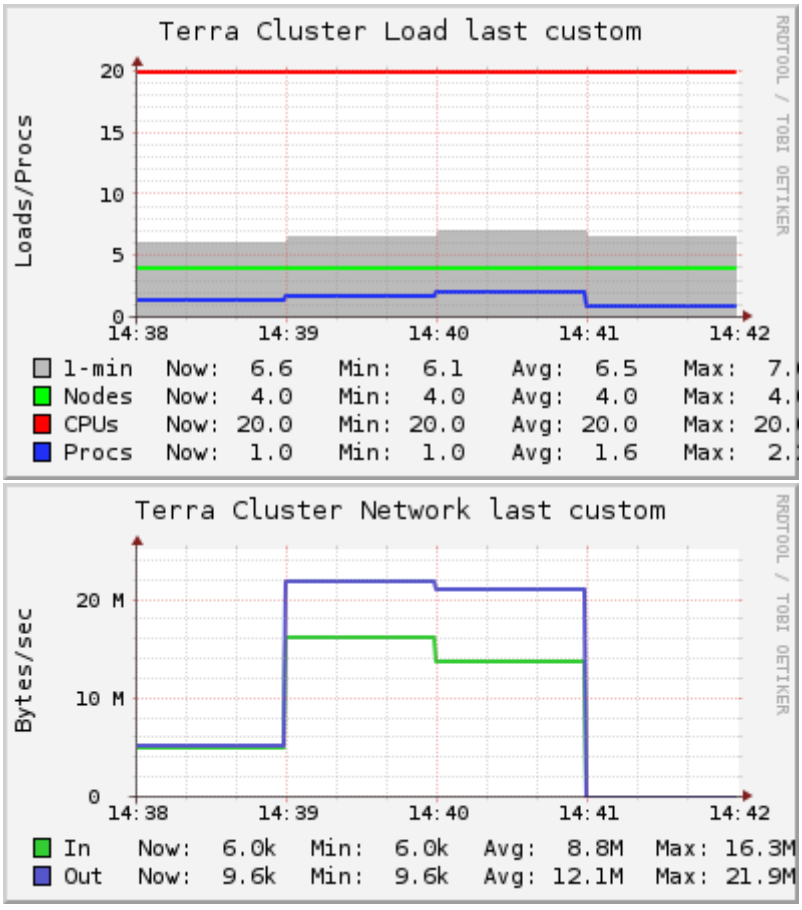

Figure 60: Cluster Load and Network during d-100-1GB test in GlusterFS

During the test e-*100-10GB* we upload 100 files of 10GB in HDFS, Ceph, GlusterFS, and XtreemFS. During this test the load of the system grows is constant, but bigger than the previous test and the network first has a big peak and then an attenuation. Figure 61 shows the situation of Ceph during this test. The graphs of HDFS and GlusterFS are not reported because are similar.

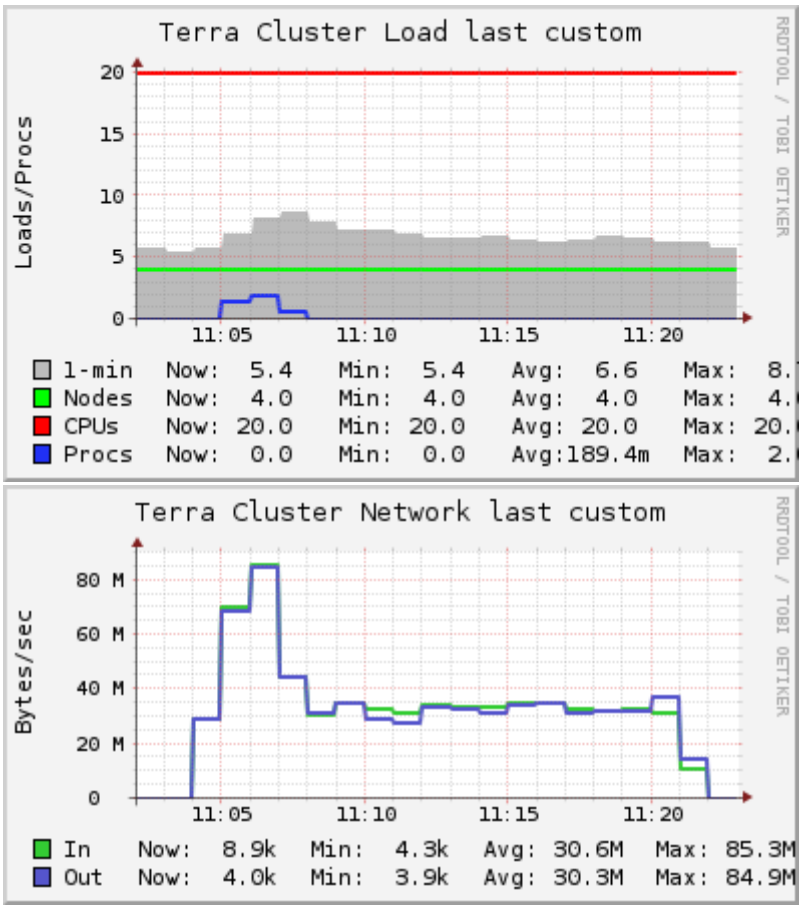

Figure 61: Cluster Load and Network during e-100-10GB test in Ceph

During the test *f-10-10-10GB* 10 clients upload 10 files of 10GB in HDFS, Ceph, GlusterFS, and XtreemFS. During this test, there is a big peak in load of the system and the network has a big peak during the files upload process. Figure 62 shows the situation of HDFS during this test. The graphs of Ceph and GlusterFS are not reported because are similar.

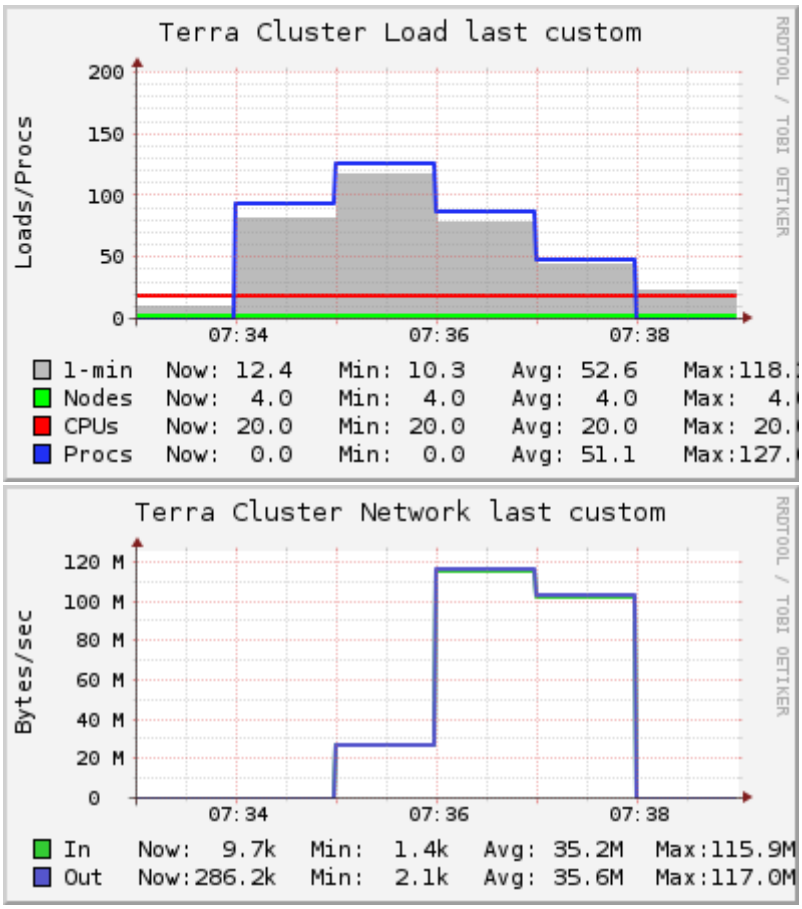

Figure 62: Cluster Load and Network during f-10-10-10GB test in HDFS

Monitoring shows that the system load has small peaks or stays constant when there is a single client, but it has a big peak and increases when multiple clients are using DFS. For the network, however, all DFSs have peaks depending on the size of the data.

5.3.4 Results in a Nutshell

This work is focused on how to provide storage support to cloud environments. We have first evaluated the different types of file systems that can be used. Local File Systems are too limited in a dynamic environment like the cloud. Traditional Distributed File Systems, on the other hand, can be suitable for small cloud environments, but when you want to provide a large cloud service, you have to rely on a Distributed File System.

Distributed file systems are a part of distributed systems. DFS does not directly serve to data processing. They allow users to store and share data. They also allow users to work with these data as simply as if the data were stored on the user's own computer. In addition, through these file systems, it is possible to have features like replication, fault tolerance and striping in very simple way. The real strength, which distinguishes them from Traditional Distributed File Systems, is that they can work (in most of the cases) even if there is no network connection because they do not have a centralized structure on a single storage point, but each node has a part of storage.

After presenting the state of the art of the different file system types, we focused on the four that we considered most important: HDFS, Ceph, GlusterFS, and XtreemFS. After the first analysis of their characteristics, we went deep by analyzing their behavior in certain situations through a custom benchmark. The benchmark was built after a careful analysis of the features to be tested and of the tools that can support the various tests.

The benchmark has been structured in two phases: a theoretical phase and a practical phase. In the first, we evaluated some features that do not require a practical phase (such as POSIX compatibility, fault tolerance support, etc.). In the second phase, instead, we tested all features that require practical testing (such as deployment or writing times). Below there is a summary of all the results of the performed tests.

The POSIX system guarantees the portability of a given interface for data access on the various systems. HDFS is not POSIX compatible. CephFS aims to adhere to POSIX semantics wherever possible. GlusterFS is a fully POSIX compliant file system. It implements the POSIX APIs using the POSIX Translator and POSIX Access Control Lists (ACLs). XtreemFS has the same "Look&Feel" as NFS or your local file system and you can use XtreemFS as a drop-in replacement for NFS.

Usability is defined as the effectiveness, efficiency, and satisfaction with which certain users reach certain goals in certain contexts. In a distributed file system, usability can be seen in client applications. HDFS provides a code library that allows users to read, write and delete files and create and delete directories in a transparent way using some API in C, Java or REST. Ceph Clients include a number of service interfaces including Block Device, Object Storage, and Filesystem. GlusterFS is a userspace filesystem that makes use of FUSE (File System in Userspace). XtreemFS supports the use of FUSE (File System in Userspace).

Most distributed file systems allow you to scale horizontally and not vertically. This means that you can add a node to the architecture to be able to extend the cluster. In HDFS once we have reached the saturation of our cluster or before it, just add a new node to the cluster. In Ceph it is possible to add monitors and OSDs according to the cluster load. In GlusterFS, it is possible to add new bricks to an existing volume. In XtreemFS it is possible to add OSDs. When you add an OSD,

XtreemFS re-balance the data according to the parameters set for each replication factor.

The consistency guarantees to the user that the data are significant and actually usable within their applications. HDFS is Eventually Consistent, that means they can take time for changes to objects (during an operation of creation, deletion or updates) to become visible to all clients. In Ceph, reads must reflect an up-to-date version of data and writes must reflect a particular order of occurrences. It allows concurrent read and write accesses using a simple locking mechanism that gives users the ability to read, read and cache, write or write and buffer data. Consistency in GlusterFS is achieved with the use of a translator (the Automatic File Replication (AFR) translator maintain replication consistency). XtreemFS provides the user with a consistent view of the entire file system. The consistency of replicas is handled by the OSDs in a distributed manner.

Consistency is also related to the concept of the snapshot. A snapshot represents an object's state at a given instant of time. HDFS Snapshots are read-only point-in-time copies of the file system. Ceph supports the creation of snapshots and allows you to keep the status of images. GlusterFS snapshots are thinly provisioned LVM based snapshots, and hence they have certain pre-requisites. XtreemFS is capable of taking file system snapshots. Snapshots are exposed as read-only volumes.

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components. The primary objective of HDFS is to store data reliably even in the presence of failures, so it supports the three common types of possible failures (NameNode failures, DataNode failures, and network partitions). To achieve reliability and fault tolerance, Ceph supports a cluster of monitors. In a cluster of monitors, latency and other faults can cause one or more monitors to fall behind the current state of the cluster. GlusterFS replace the fault tolerance of RAID with replication allows you to spread that vulnerability between 2 or more complete servers.

XtreemFS provides fault tolerance and scalability by supporting replication and striping of files. File replica management can be automated by means of a replica management service.

Striping is useful when a processing device requests data more quickly than a single storage device can provide it. In HDFS, striping concept is connected to Erasure Coding (EC) concept. In storage systems, the most notable usage of EC is Redundant Array of Inexpensive Disks (RAID). The RAID type most similar to Ceph's striping is RAID 0, or a 'striped volume.' Ceph's striping offers the throughput of RAID 0 striping, the reliability of n-way RAID mirroring and faster recovery. In GlusterFS striping is related to the volume type, so if you want to stripe you must use a striped volume. XtreemFS supports striping of file content. Different parts of a striped file are handled by different OSDs.

Caching enables you to efficiently reuse used data that has already been recovered or processed. Cache data is generally stored in quick access hardware, such as RAM, and can be processed with the help of a software component. In HDFS cache is called Centralized cache management in HDFS that is an explicit caching mechanism that allows users to specify paths to be cached by HDFS. In Ceph it is possible to use cache tiering. A cache tier provides Ceph Clients with better I/O performance for a subset of the data stored in a backing storage tier. In GlusterFS you can use cache in a volume and you can specify the cache size. In XtreemFS the read-only replication helps you quickly build a caching infrastructure on top of XtreemFS in order to reduce latency and bandwidth consumption between datacenters.

A distributed file system can offer some integration capabilities with other systems. HDFS offers the ability to use its own storage in other applications such as Apache Spark. Ceph, in addition to offering an easily accessible distributed file system, is widely used in cloud environments such as OpenStack. GlusterFS offers many possibilities of use for different applications. XtreemFS provides the use of FUSE,

so it's possible to integrate with all the tools that need a file system, such as a mount point for a container.

A particular DFS can run on different operating systems (Windows, Linux or Mac) or not and this is a big difference if you want to build your own storage system on multiple platforms. HDFS can be used on many platforms that support Java. It is possible to deploy Ceph on newer releases of Linux, but it is recommended to deploy Ceph on releases with long-term support. GlusterFS, if you are using NFS/CIFS supports all Fedora and Debian based distributions, UNIX (Solaris 10+), AIX, HP-UX, and some version of Microsoft Windows Server. XtreemFS can be used on many platforms. It is possible to download the distributed file system for Windows, Linux, and Mac or use directly the source code in C++/Java.

Client applications in a DFS can be used with or without access. For greater security, the various DFSs support different methodologies to ensure security. Security features of HDFS consist of authentication, service level authorization, authentication for Web consoles and data confidentiality. To protect data, Ceph provides its cephx authentication system, which authenticates users operating Ceph clients. The cephx protocol operates in a manner with behavior similar to Kerberos. GlusterFS allows its communication to be secured using the Transport Layer Security standard (which supersedes Secure Sockets Layer), using the OpenSSL library. XtreemFS offers mechanisms for user authentication and authorization, as well as the possibility to encrypt network traffic.

Deployment Time is the time required for installing the distributed file system and it is measured from the launch of the installation to the start of all daemons. HDFS took 88 minutes, Ceph 63 minutes, GlusterFS 18 minutes, and XtreemFS 27 minutes.

Write time represent the time that needs to the system to write files in one node. It is measured on all nodes where replicas are placed. In this

test, Ceph needs more time then HDFS and GlusterFS. XtreemFS times, instead, are between GlusterFS and Ceph times. This probably is due to different architectures of the DFSs and to the presence of Placement Groups in Ceph that need to be updated. HDFS has a centralized point where are stored metadata and Ceph, GlusterFS, and XtreemFS have different points where are stored metadata. In HDFS, GlusterFS, and XtreemFS, in addition, PGs do not exist so the times are smaller. It should be noted that the differences in times are few milliseconds.

The using a DFS is closely related to its benefits. One of these is replication, that is the possibility to have copies of the same file on multiple machines and to ensure fault tolerance. One of the most important parameters is the write propagation time that is the time needed to have a consistent state of files on all servers. For example, we suppose that at time t_0 file1 is consistent on server1, at time t_1 file1 is consistent on server2 and at time t_2 file1 is consistent on server3. The propagation time is the millisecond between t_0 and t_1 for two consistent replicas and the millisecond between t_0 and t_2 for three consistent replicas. Ceph has bigger propagation times then HDFS, GlusterFS, and XtreemFS. This probably is due to the presence of Placement Groups such as in the previous test. The difference, however, is of few millisecond, so not big.

System load with multiple clients represents the status of the cluster in a situation where the number of clients is variable and tends to bring the system to a saturation state. It is possible to see the differences between the three DFSs during a phase of stress. Ceph, even in this test, has bigger time than HDFS, GlusterFS, and XtreemFS. GlusterFS and XtreemFS, on the other hand, have higher times than the previous tests, so when there are more clients' connections GlusterFS and XtreemFS have bigger write times. HDFS, on the other hand, always has the best write times.

Fault-tolerant technology is a capability of a system to deliver uninterrupted service, despite one or more of its components failing.

HDFS provides fault tolerance support as described above. After the node failure, the DFS has marked the node as unreachable and after about 10 minutes it started re-balancing process. Ceph, on the other hand, was immediately labeled the monitor on the failed node as unreachable and the OSD as disconnected. The re-balancing process started after a few minutes. GlusterFS immediately marked the peer of the failed node as disconnected but did not launch any re-balancing process because the it is not in charge of the DFS. XtreemFS provides fault tolerance support as described above re-balance the file system in about 15 minutes.

This test Measure the time taken by a distributed file system to return to a consistent state after a fault. HDFS starts re-balancing process after 10 minutes and 30 seconds. To reach a new consistent state, respecting the number of replicas, HDFS took 1 minute and 22 seconds. Then, after about 12 minutes, HDFS reported the DFS in an optimal state of operation. Ceph, instead, starts re-balancing process after 2 minutes and 10 seconds. To reach a new consistent state took 11 minute and 4 seconds. Then, after about 13 minutes, Ceph reported the DFS in an optimal state of operation. XtreemFS starts re-balancing process after 5 minutes. To reach a new consistent state took 15 minutes. Then, after about 15 minutes, XtreemFS reported the DFS in an optimal state of operation.

During the entire execution of the benchmark, Ganglia was running and provided data on the state of the system. During this test the load of the system is minimum and the network has a small peak during file transmission. Network peaks occurred during the transmissions of large files, while there were heavy loads while simulating multiple clients.

The table in Figure 63 shows the results of the benchmark.

	HDFS	**Ceph**	**GlusterFS**	**XtreemFS**
POSIX semantics compatibility	Not compatible	Compatible only in some features	Fully compatible	Similar to NFS
Usability	C API, Java API and REST	FUSE, mount, REST, RADOS, Block Device, Object Storage and Filesystem	FUSE, mount	FUSE, mount
Extensibility	Horizontal (with new DataNode)	Horizontal (with new Monitor or OSDs)	Horizontal (with new Brick)	Horizontal (with new OSDs)
Consistency	WORM, lease	Lock	AFR	Lease
Snapshots and Backup Support	Supported	Supported	Supported	Supported
Fault Tolerance	Support for NameNode, DataNode and Network failure	Support for Monitor and OSD failure	RAID-like	RAID-like
Striping	Supported	Supported	Supported	Supported
Caching	Centralized cache management	Cache tiering	Cache of bricks	With read-only replica
Integration with other systems	Supported	Supported	Supported	Supported
Supported Platforms	Windows, Linux, and Mac	Linux	Fedora, Debian, UNIX, AIX, HP-UX, and	Windows, Linux, Mac, and source in C++/Java

			Windows Server	
Security	Supported	Supported	Supported	Supported
Deployment Time	~ 88 minutes	~ 63 minutes	~ 18 minutes	~ 27 minutes
Writes Time	Low	High	Medium	Medium
Writes Propagation Time	Low	Medium	Low	Low - Medium
System load with multiple clients	Low	High	Medium	Medium
Fault tolerance system behavior	Detection of fault and re-balance process	Detection of fault and re-balance process	Detection of fault	Detection of fault and re-balance process
Node re-balancing time	Low	Low	-	Low
Cluster Monitoring	Low Load and some network peaks	Low Load and some network peaks	Low Load and some network peaks	Low Load and some network peaks

Figure 63: Table of Results

Conclusion

Cloud computing can offer benefits to many companies that can provide their applications as a service. It is a business model where the user does not buy the product, but the ability to use that product and to do so remotely with a simple Internet browser.

This thesis focused on cloud storage solution, also known today as hyper convergent. To thoroughly analyze these technologies, we studied the behavior of three different types of file systems: local file systems, network file systems, and distributed file systems.

The aim of this work is to explore the various methodologies that can offer storage to a cloud environment. To do this, some of the most commonly distributed file systems will be tested through a custom benchmark.

We started studying storage solutions in cloud computing, the differences between the various file system types and the common use cases of file systems in a cloud environment. After, there is a comparison of the most important technologies present in the State of the Art. Thanks to this comparison, we chose four distributed file systems to deepen: HDFS, Ceph, GlusterFS, and XtreemFS. We discussed the most important features of these file systems, such as the architecture, the fault tolerance support, the snapshot and backup support, etc. with a theoretical comparison, but this is not enough. In order to understand which distributed file system was the most suitable for certain cases, it is necessary a specific distributed file system benchmark. After searching the evaluation criteria to be taken into account for a proper analysis and the tools needed to make the benchmark, we have built a benchmark plane.

The benchmark consisted of two phases: a theoretical one and a practical one. The first phase discusses features that do not require practical tests, such as architecture or deployment modes. The second

phase, instead, was finalized to perform extensive practical tests to assess some main performance measurements, such as propagation times and writing times.

The results showed that there are some differences between the four distributed file systems. The first phase showed that the four file systems offer snapshot support, but Ceph, GlusterFS, and XtreemFS have more convenient and easy to use access modes. In addition, consistency support is developed in several ways on the four distributed file systems. In the second part, however, some differences are emerged in writing, propagating, and rebalancing times. These differences are probably due to the different architectures of the four file systems, Ceph shows slightly higher times (few milliseconds) due probably to the presence of the placement groups.

Each of the four DFSs ensure transparency and fault tolerance using different methods that provide the same results. Some differences emerged in the design. A decentralized architecture allows to scale easier and faster thanks to greater load distribution. Another important point is the choice of using asynchronous replication, which allows to manage a large amount of data more dynamically. About fault tolerance, it should be noted that all the systems have immediately noticed the lack of a node, but GlusterFS has not automatically proceeded to recapture the right number of replicas.

In conclusion, this work has shown that in a cloud environment it is much more convenient to use distributed file systems as they provide highly reliable and high availability features that should otherwise be implemented by hand. Using HDFS is recommended when you do not need to save different data types, but you need a simple and fast distributed file system. Ceph, on the other hand, provides many benefits at the price of a few milliseconds. Finally, GlusterFS and XtreemFS provides a storage similar to Ceph, but without placement groups.

References

[1] File System Definition of Margaret Rouse, www.searchstorage.techtarget.com/definition/file-system.

[2] D. Peric, T. Bocek, F.V. Hecht, D. Hausheer, and B. Stiller, "The Design and Evaluation of a Distributed Reliable File System," in Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on, Higashi Hiroshima, 2009.

[3] File Allocation Table Definition of Margaret Rouse, www.searchexchange.techtarget.com/definition/file-allocation-table.

[4] NTFS Definition of Margaret Rouse, www.searchwindowsserver.techtarget.com/definition/NTFS.

[5] ZFS Definition, www.oracle.com/storage/nas/index.html.

[6] BTRFS Official Definition, www.btrfs.wiki.kernel.org/index.php/Main_Page.

[7] NFS Definition of Pavel Bžoch, Distributed File Systems: The State of the Art and concept of Ph.D. Thesis, University of West Bohemia.

[8] Samba Definition of Pavel Bžoch, Distributed File Systems: The State of the Art and concept of Ph.D. Thesis, University of West Bohemia.

[9] AFS Definition of Pavel Bžoch, Distributed File Systems: The State of the Art and concept of Ph.D. Thesis, University of West Bohemia.

[10] Coda Definition of Pavel Bžoch, Distributed File Systems: The State of the Art and concept of Ph.D. Thesis, University of West Bohemia.

[11] Hadoop Distributed File System, www.hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[12] Ceph, www.ceph.com.

[13] GlusterFS, www.gluster.readthedocs.io/en/latest/Install-Guide/Overview/.

[14] HekaFS, www.fedoraproject.org/wiki/Features/HekaFS.

[15] OrangeFS, www.www.orangefs.org/.

[16] GridFS, www.docs.mongodb.com/manual/core/gridfs.

[17] XtreemFS, www.xtreemfs.org/all_features.php.

[18] Amazon Simple Storage Service, www.docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html.

[19] XtreemFS, www.xtreemfs.org.

[20] Test method and test tool for distributed file system of Liu Aigui, www.prog3.com/article/1970-01-01/311573.

[21] Ganglia Monitoring System, www.ganglia.sourceforge.net.

[22] Bonnie, www.textuality.com/bonnie.

[23] IOzone, www.iozone.org/.

[24] FIO, www.freecode.com/projects/fio.

[25] Inotify Python Version, www.pypi.python.org/pypi/inotify.