Cloud Distributed File Systems: a Benchmark of HDFS, Ceph, GlusterFS, and XtremeFS

[*]Luca Acquaviva, [†]Paolo Bellavista, [†]Antonio Corradi, [†]Luca Foschini, [†]Leo Gioia, [†]Pasquale Carlo Maiorano Picone
[†]Dipartimento di Informatica – Scienza e Ingegneria, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
{paolo.bellavista, antonio.corradi, luca.foschini, pasquale.maiorano4}@unibo.it, leo.gioia@studio.unibo.it
[*] Imola Informatica, Via Selice, 66/A, 40026 Imola (BO), Italy
lacquaviva@imolinfo.it

***Abstract*—Cloud computing nowadays is the cornerstone for all the business applications, mainly because of its high fault tolerance characteristic. High resilience and availability typical of cloud-native applications are achieved using different technologies. Regarding the file system, the main fault tolerant application examples are distributed file systems, such as HDFS, Ceph, GlusterFS, and XtremeFS. These file systems have different architectures and deployment models than the Traditional Distributed File Systems (TDFSs), such as NFS. The primary goal of this work is to analyze and compare different Cloud Distributed File Systems (CDFSs) in terms of characteristics, architecture, reliability, and components. As a key feature, the paper benchmarks them considering as use case an IaaS platform.**

Keywords—Distributed File Systems; Cloud Distributed File Systems; performance analysis; performance benchmarking

I. INTRODUCTION

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 humans but are assigned, quickly and conveniently, through automated procedures, starting from a pool of shared resources with other users, without the involvement of the user in the configuration process. When the user releases the resource, it is reconfigured in the initial state and made available in the shared pool of resources. This work focuses only on a part of the cloud computing world: the storage area that represents hardware devices, storage media, infrastructures, and non-volatile storage software of large amounts of electronic information. The concept of storage is tightly tied to the concept of the file system. The exploitation of a Cloud Distributed File System (CDFS) in a complex architecture, such as an Infrastructure as a Service (IaaS) cloud environment, leads to a stable and reliable system, which helps the cloud provider to achieve fault tolerance and to avoid all problems regarding storage for all purposes.

Although distributed file systems (called in the following Traditional Distributed File Systems - TDFSs) existed since long, such as the Network File System (NFS) that allows a unified vision of traditional Storage Area Network (SAN), they are typically not suitable for cloud environments in terms of performances, reliability, and level of automation. Moreover, TDFS were not designed to guarantee multi-site global redundancy and replication as they typically relied on concentrated storage and assumed the presence of VPN and controlled access.

Conversely, CDFSs used in modern IaaS systems provide high global resiliency and reliability. These components also enable a "uniform" vision of storage area in the multi-datacenter scenario, because they offer mechanisms to enforce synchronization between different sites taking always into account security. These kinds of systems are now widely used in all cloud scenarios, from private to public ones because of their capabilities, among the others, the uniform visibility of underlying storage space for IaaS application levels such as Object and Block storage.

The need of a uniform vision for multi-node (and even also multi-site) data and storage was a well-known problem for a long time. Previously with SANs and TDFSs, we had "giant Network Attached Storage" which run a multitude of services for storage, from NFS servers to iSCSI initiator in order to provide all possible services and storage to other running servers. This kind of machines had different power supplies and a fallback plan in case of hard disk break down, but, notwithstanding that, constituted a single point of failure.

CDFSs offers high availability and redundant data across multiple sites, enabling also a seamless cross-site migration for components like Virtual Machines and Containers. In order to be deeply integrated with their specific purpose platform, the CDFS offer a first class driver which implements an interface used by the platform. For instance, Ceph offers a driver for OpenStack Object Storage, Block Storage, and Virtual Machine Storage.

Our work goal is to provide an innovative comparison of CDFS. The originality of our analysis and comparison is manifold. First, we propose possible benchmark standardization guidelines by identifying common parameters for future releases and new CDFS products. Second, we present a qualitative comparison where all aspects related to CDFS characteristics are analyzed: the benchmark has been designed and implemented considering a normal IaaS infrastructure as a use case. Third, we provide an extensive and fully expressive quantitative performance benchmark that covers all the possible aspects of a typical IaaS use case. Fourth, we make available to the community a benchmark prototype that can be used in order to test if a deployment is efficient [16].

978-1-5386-4727-1/18/$31.00 ©2018 IEEE

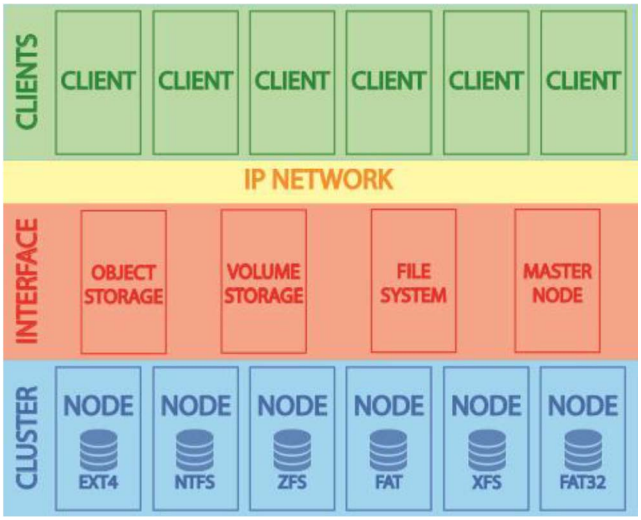

Fig. 1. Cloud Distributed File Systems Architecture.

II. Cloud Distributed File Systems

There are several notable examples of TDFSs that have been proposed in the last three decades, mainly in the eighties, such as NFS by SUN [5], Samba by IBM (SMB) [6] and Andrew File System (AFS) by Carnegie Mellon [7]. They shared the common goal of providing functional characteristics as closer as possible to that of the UNIX file system, and supported wide accessibility up to an entire university campus as for AFS, but they were typically designed for geographically limited networks and without multi-site global (worldwide) reliability.

CDFS has been recently proposed as a new generation of DFSs that allow users to store and share data as simply as if the data were stored locally. In CDFS data can be stored on several nodes to guarantee replication for achieving better system performance and/or for achieving reliability of the system. Compared to TDFSs, in CDFSs data are more protected from a node failure. If one or more nodes fail, other nodes can provide all functionality. Files can be moved among nodes for achieving load-balancing. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should be transparent. If the capacity of the nodes is not enough for storing files, new nodes can be added on-the-fly to the existing CDFS to increase its capacity by scaling horizontally.

Fig. 1 shows a typical layered architecture of a distributed file system:

- *Cluster Layer* contains cluster nodes that represent the storage units of the CDFS; each of them can use a different local file system for saving the various files.
- *Interface Layer* allows the high-level use of CDFS and uses the nodes of the underlying cluster in high-level storage pools; this level exposes APIs to allow the interaction between CDFS and clients.
- *IP Network Layer* provides network connectivity to CDFS.
- *Clients Layer* allows clients to interface with CDFSs. The direct use of APIs is possible.

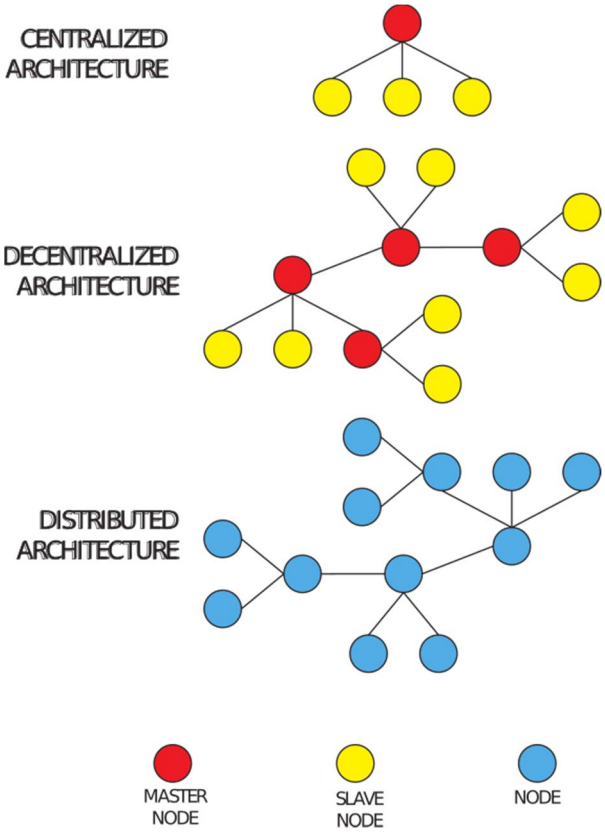

Fig 2. Distribution of CDFS nodes.

It is important to remember that some elements may not be present in some CDFSs because for design choices they are implemented differently.

In order to have a clearer vision of how CDFSs work and to evaluate their pros and cons, we have decided to take an experimental approach consisting in deploying various types of CDFSs. The chosen parameters that drove our choice were diffusion, ease of use, and architectural differences. Based on these criteria we have decided to choose Hadoop Distributed File System [8], GlusterFS [9], Ceph [10] and XtreemFS [11]. In fact, these are very good examples of CDFSs in terms of facility of access and use and wide diffusion in the cloud market. In addition, we purposely selected these four CDFSs because they adopt different distributed architectures, each with its own pros and cons. HDFS implements a *centralized architecture* (see top of Fig. 2) where a node (Master Node) manages and coordinates the other nodes of the cluster (Slave Nodes). GlusterFS adopts a *decentralized architecture* (middle of Fig. 2) where a series of nodes deal with the management and coordination of the other nodes. Finally, Ceph and XtreemFS use a (fully) *distributed architecture* (bottom of Fig. 2) where all the nodes are involved both in the management of the file system and in the management and coordination of the cluster. Fig. 2 shows all different architectures in terms of node hierarchy and distribution. In centralized and decentralized architectures, the red nodes are master nodes, i.e., nodes responsible of control and management decisions, while yellow nodes are the slave nodes responsible for managing the file system content such as files, objects, and other relative data.

The decentralized architecture instead relies on a multi-master environment and partitioned load and data distribution, defining also a protocol for master coordination. In distributed architectures, instead, all nodes have the same importance and decisions are made through the use of coordination protocols.

III. Cloud Distributed File System Benchmark

This section introduces our CDFS benchmark framework that consists of a first part addressing qualitative distinctive aspects and a second one more focused on quantitative performance assessment dimensions.

A. Qualitative Analysis Dimensions

For the qualitative analysis, we focus on several relevant characteristics, typically considered as non-functional aspects, namely: POSIX API compliance, deployment support, usability, scalability, consistency, and additional capabilities such as snapshot, stripping, and caching support.

POSIX API (Portable Operating System Interface API) compliance consists in the portability of a given interface for data access on the various systems, which is one of the most important features of CDFS. *Deployment* consists in the methodology for delivering and executing a particular application or service. In order to deploy a service or an application, there is a set of operations to perform; the deployment parameters aim to analyze if these operations for each CDFS could be automated or not. *Usability* aims to grab the degree of satisfaction in the interaction between the user and the CDFS such as making available certain end-user features to simplify its use. *Scalability* is the ability of a system to grow to meet the needs of an increasing number of users; in particular, we are interested in evaluating the level of automation support that allows to dynamically and elastically modify the CDFS configuration by adding/removing nodes to scale-out/-in. *Consistency* in a CDFS means that following various operations you can continue to ensure data integrity by avoiding any loss or error on the files. Moreover, in a CDFS 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 considered 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. As regards additional capabilities, *snapshot* represents the possibility to freeze and store an object's state at a given instant of time. A snapshot could be crash-consistent if it remains consistent after a power outage or similar random events or application-consistent if the application can enter in a "freeze state" (while the application is a freeze, it cannot respond to requests) during the snapshot's process. This is especially useful in cloud IaaS, for instance, when we want to periodically save the state of a Virtual Machine (VM). *Striping* is the capability to spread data segments across multiple devices which can be accessed concurrently with the goal to increase total data throughput. It is also a useful method for balancing I/O load across an array of disks. Finally, *caching* is the possibility of storing data in a local temporary storage; different CDFSs provide different possibilities to manage caching, with regard to general configurability as well as management of the consistency of cached data.

B. Quantitative Performance Assessment Indicators

The performance assessment of a distributed file system has to be evaluated on multiple critical dimensions. Moreover, to enable accurate evaluation of the performance of a CDFS, it is necessary to monitor both client-side and server-side behaviors and to keep in mind that they might be strongly influenced by the network. In particular, the server-side plays a pivotal role in taking care of data storage in memory, providing recovery in case of failures, replication support, and so forth. In the following, we report a list of main indicators we considered for our performance tests, namely, deployment time, write time, write propagation time, and fault tolerance. For a wider taxonomy and formal definition of evaluation criteria that are used for distributed file systems benchmarking, we also refer readers to [12].

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. *Write time* is the total time used in order to write a new file on the CDFS; this is influenced by the dimension of the file and also by the driver efficiency for the target CDFS. *Write propagation time* defines how much the CDFS needs to reach a consistent state spreading the modification or creation of a file across all replicas. Finally, to assess scalability and fault tolerance, we measured, respectively, the average time to serve *multiple clients* under a stress situation (close to overload) and the *re-balancing time* as the time taken by a distributed file system to return to a consistent state after a fault. .

IV. Benchmark Result

In order to implement a reliable CDFS benchmark, we used and integrated some traditional tools for host monitoring and for monitoring changes within a file system. These tools depict exactly how a specific CDFS handles a specific test and helps us to better understand the CDFS behavior. For host machines monitoring, we chose *Ganglia* by exploiting its reliable storage on a database and its complete web interface. In order to monitor file system changes, instead, we preferred to use low-level system tool that allowed us monitoring all events that occur on the inode. For this reason, we have opted for *Inotify*.

CDFSs present a lot of features that need to be tested. This section first discusses all qualitative aspects according to the qualitative analysis dimensions introduced in the previous section. Then, we present experimental results collected to quantitatively assess CDFS performance indicators for the considered IaaS use case.

A. Qualitative Analysis and Comparison

Starting with POSIX API, HDFS is not POSIX compatible, CephFS aims to adhere to POSIX semantics wherever possible, while GlusterFS is a fully POSIX compliant file system and 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.

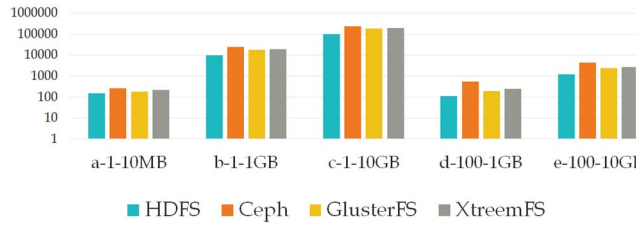

Fig 3. Write Time Test.

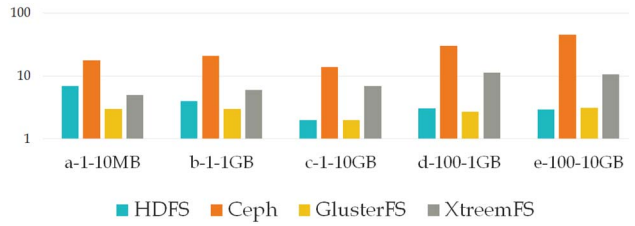

Fig 4. Write Propagation Time Test.

Regarding deployment and usability, HDFS provides a code library that allows users to use file system 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 File system. GlusterFS is a userspace file system that makes use of File System in Userspace (FUSE). Finally, XtreemFS supports the use of FUSE similar to GlusterFS.

Focusing on scalability, 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 nodes 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 nodes. When you add an node, XtreemFS re-balances the data according to the parameters set for each replication factor.

As for consistency, HDFS is eventually consistent. It means that it can take some time before changes to objects (during an operation of creation, deletion or updates) become visible to all clients. In Ceph, reads must reflect an up-to-date version of data and writes must reflect the 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. Consistency in GlusterFS is achieved with the use of the Automatic File Replication (AFR) translator that maintain replication consistency. XtreemFS provides the user with a consistent view of the entire file system; the consistency of replicas is handled by each node according to the distributed architecture.

With regard to snapshot support, 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 that are exposed as read-only volumes. Connected to the snapshot support, CDFSs also support fault tolerance. 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 replaces the fault tolerance of Redundant Array of Inexpensive Disks (RAID) with replication allows you to spread that vulnerability between 2 or more complete servers. Finally, XtreemFS provides fault tolerance (and scalability) by supporting replication and striping of files: file replica management is an autonomous and automated service.

Focusing on striping, in HDFS it is connected to the Erasure Coding (EC) concept. In fact, in storage systems the most notable usage of EC is RAID. The RAID type most similar to Ceph striping is RAID 0, or a 'striped volume'. Ceph striping offers the throughput of RAID 0 striping, the reliability of n-way RAID mirroring and faster recovery. In GlusterFS, striping relates to the volume type, so if you want to stripe you must use a striped volume. XtreemFS also supports striping of file content and it handles different parts of a striped file by different nodes.

Finally, regarding caching, in HDFS cache is called Centralized cache management and it is available as 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, it is possible to use cache in a volume and the user can specify the cache size. In XtreemFS, the read-only replication helps to quickly build a caching infrastructure on top of XtreemFS in order to reduce latency and bandwidth consumption between datacenters.

B. Performance Assessment Experimental Results

After the qualitative evaluation, we have performed a thorough quantitative assessment performing several performance tests. In our benchmark, we considered the IaaS use case considering various input parameters and performance indicators. For the tests, we used a four machines cluster: three to host CDFSs and one to emulate clients and for monitoring. The machines were equipped with Intel(R) Xeon(R) CPU E5-2603 v4 (1.70GHz), 40 GB 1866 MHz DDR3 SDRAM and 2TB HDD (WD Red Pro 2TB NAS Hard Disk Drive - 5400 RPM Class SATA 6 Gb/s 64MB Cache 3.5 Inch - WD20EFRX). The network between the machines was a 100MB/s. Each server we installed Ubuntu Server 16.04. Shown results (apart Fig. 5-b, Fig. 6, and Fig. 7) are average time values expressed in milliseconds and shown in log scale collected over 33 runs; we are not reporting standard deviations that that are typically rather limited (always below 6% across all tests).

In the first result set, we focused on the deployment time. HDFS took 88 minutes, Ceph 63 minutes, GlusterFS 18 minutes, and XtreemFS 27 minutes, and thus the last two CDFSs are faster to be deployed from scratch.

In our second result set, shown in Fig. 3, we assessed the write time with one client in five different configurations from 'a', that loaded the CDFS writing one file of 10 MB (i.e., a-1-10MB), to 'e', that loaded it with 100 files of 10 GB (i.e., e-

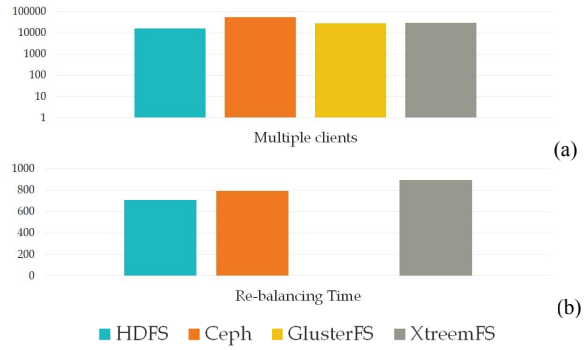

Fig. 5. (a) Multiple Clients and (b) Re-Balancing Time Tests.

100-10GB). Write times were measured on all nodes where replicas are placed and are expressed in milliseconds. First of all, we note that write times with multiple file are typically lower than those for single file. We believe this is due to the fact that CDFSs are typically tuned for fast management of many files leveraging multiple threads. In other words, when writing a single file fewer threads are allocated, while when there are more files the thread pool is larger. In addition, taking into account that all CDFSs before saving a file they save the metadata, when the thread pool is larger this work is faster. Finally, before propagating the changes to the other nodes, the chunk must be consistent and this process takes less time if more threads are employed. At a finer level, for each configuration, Ceph needs more time then HDFS and GlusterFS. XtreemFS times, instead, are between GlusterFS and Ceph times. We believe this is due to different architectures of the CDFSs and to the presence of Placement Groups (PGs) in Ceph that need to be updated. HDFS has a centralized point where metadata are stored while 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.

Our third result set shows write propagation times, one of the most important parameters to evaluate replication efficiency (see Fig. 4). For example, we suppose that at time t_0 $file_1$ is consistent on $server_1$, at time t_1 $file_1$ is consistent on $server_2$ and at time t_2 $file_1$ is consistent on $server_3$. The propagation time is the milliseconds 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. We believe again this is due to the presence of PGs in Ceph.

Our fourth result set reports the results for a write test with multiple clients; we considered 10 clients, each one storing 10 files of 10GB to bring the system to a saturation state. As Fig. 5-a shows, applying the same configurations as our for our second result set, all CDFSs present (as expected) higher times. Looking deeper into differences in Fig. 5-a, multiple clients Ceph write time is the highest, GlusterFS, and XtreemFS have higher times than the previous tests, so we believe they present a higher overhead in managing multiple client connections. Finally, HDFS, always has the best write times with a lower overhead.

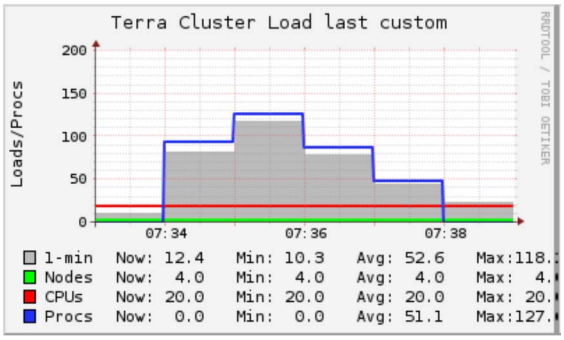

Fig. 6. Load Information of Cluster.

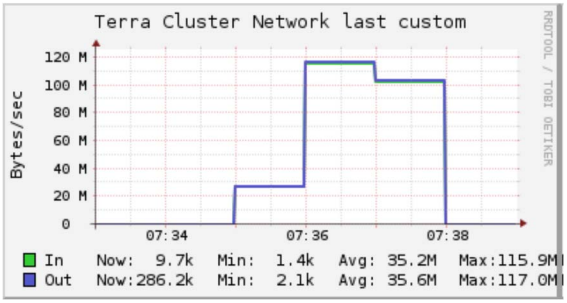

Fig. 7. Network Information of Cluster.

Our fifth result set focuses on fault tolerance, and more precisely and re-balancing times when we turn off one node. Fig. 5-b reports (differently from other figures) the times in linear scale and in seconds to better appreciate the differences. For each CDFS there is a *reaction time* after a node has been marked as unreachable and then a *recovery time* to rebalance data over all nodes, we show in Fig. 5-b the sum of these two terms that we define overall re-balancing time. HDFS re-balancing time is the best one with 712s (respectively, 630s and 82s reaction and recovery times), Ceph follows with 794s (130s and 664s), then XtreemFS presents the worst behavior with 900s (300s and 600s). Finally, as regards GlusterFS, it immediately marks the peer of the failed node as disconnected, but it does not launch any recovery process that is out-of-the-scope of this CDFS (this is the reason why there is no result for it).

Finally, during the entire execution of the benchmark, Ganglia was running and providing data on the state of the system. In general the load of the system is minimal and the network presents peaks during file transmission especially for large files. Fig. 6 and Fig. 7 show the information about Load and Network for one test in the fourth result set about Ceph.

V. Related Work

Without pretense of being exhaustive, in the following we report a few significant some efforts made in realizing novel CDFS, and then a review of works that, similar to ours, survey and analyzed different CDFS.

The literature has many examples of CDFS. For instance, at CERN adopts a CDFS called CernVM-FS that is completely decentralized and enables a uniform vision of resources using Filesystem in USErspace (FUSE) [1]. CernVM-FS is for internal CERN use and Jacob Blomer et al. [2] have remarked how it could be helpful for the scientific community in order

to have a reliable and fast storage support for experimental purposes. Focusing on the internal architecture and organization of CDFSs in general, Jeffrey Shafer et al. pointed out that they typically require complex configurations (of the multiple coordinators and high availability functions) and there might be many drawbacks in misconfiguration, from poor performances to a complete failure in high availability if the deployment choice is completely wrong [3]. Furthermore, CDFSs (precisely in IaaS) must support multi-tenancy in order to be more configurable and useful. This feature should be defined at configuration level or enforced using a middleware like the one proposed by Giorgos Kappes et al. that enables multi-tenant access control at the distributed file system level, that exploits also the opportunities offered by multi-datacenter deployments [4].

Focusing on comparisons and analyses on CDFSs, ours is not the only one. Indeed, there are others ranging from a simple taxonomies to performance assessments, in particular involving Hadoop. However, other contributions are mainly focused on the qualitative comparison from model level point of view. Our comparison, instead, goes beyond the simple qualitative comparison defining also a benchmark for assessing and comparing performances.

D. Sathian et al. [13] have defined a complete taxonomy of main characteristics of CDFSs, analyzing from architecture to consistency and taking into account also naming and metadata. This work is pretty complete for model point of view, but there is no comparison in terms of performance. Yuduo Zhou made a comparison between some CDFSs considering also Google File System (GFS), Hadoop and many more [14] and similar to the previous one also this comparison is focused only on technical characteristics without any benchmark. Finally, Daravath Ramesh et al. also made a very detailed qualitative comparison between CDFSs [15]; however, although very in-depth, this survey considers only GFS and HDFS and takes into account different characteristics without comparing system performances.

VI. Conclusion

This work explores the various methodologies that can offer storage in a cloud environment. We have conducted a comparison on four of the most relevant industrial CDFS implementations at the time of writing, namely, HDFS, Ceph, GlusterFS, and XtreemFS, by defining qualitative and quantitative tests.

Obtained results showed that there are some differences between the four distributed file systems. Each of the four CDFSs ensures transparency and fault tolerance using different methods. Some differences emerged in the design; for instance, decentralized architecture allows scaling easier and faster thanks to greater load distribution. Another important point is the choice of using asynchronous replication, which allows managing a large amount of data dynamically. About fault tolerance, it should be noted that all the systems have immediately noticed the fault (lack) of a node, but GlusterFS does not support automatic recovery of the faulty node.

In conclusion, this work has shown that in a cloud environment it is much more convenient to use CDFSs as they provide highly reliable and high availability features that should otherwise be implemented by hand over traditional TDFSs. Among benchmarked CDFSs, HDFS showed good performances when it is not necessary to save different data types, but there is the need of a simple and fast distributed file system. Ceph, instead, looks very promising in terms of a wide set of supported APIs and functions at the price of a limited additional overhead.

Acknowledgment

This research was supported by the SACHER (Smart Architecture for Cultural Heritage in Emilia Romagna) project funded by the POR-FESR 2014-20 (no. J32I16000120009) through CIRI.

References

[1] CernVM FS - https://cernvm.cern.ch/portal/filesystem *(Accessed August 2018).*

[2] J. Blomer *et al.*, "The Evolution of Global Scale Filesystems for Scientific Software Distribution", *IEEE Computing Science & Engineering*, vol. 17, no. 6, pp. 61-71, 2015.

[3] Jeffrey Shafer, Scott Rixner, and Alan L. Cox "The Hadoop Distributed Filesystem: Balancing Portability and Performance", *IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, 2010.

[4] G. Kappes, A. Hatzieleftheriou, and S.V. Anastasiadis "Multitenant Access Control for Cloud-Aware Distributed Filesystems", *IEEE Transactions on Dependable and Secure Computing*, doi: 10.1109/TDSC.2017.2715839, 2017.

[5] O. Kulkarni, D. Gawali, S. Bagul, B. B. Meshram, "Study of Network File System (NFS) And Its Variations", *International Journal of Engineering Research and Applications (IJERA)*, vol. 1, no. 3, pp. 721-729, 2011.

[6] Samba Definition - https://www.samba.org/cifs/docs/what-is-smb.html *(Accessed August 2018).*

[7] Monali Mavani, "AFS Definition, Comparative Analysis of Andrew Files System and Hadoop Distributed File System", *Lecture Notes on Software Engineering*, vol. 1, no. 2, pp. 122-125, 2013.

[8] Hadoop Distributed File System, www.hadoop.apache.org/docs/r1.2.1/hdfs_design.html *(Accessed August 2018).*

[9] GlusterFS - www.gluster.readthedocs.io/en/latest/Install-Guide/ Overview *(Accessed August 2018).*

[10] Ceph - www.ceph.com *(Accessed August 2018).*

[11] XtreemFS - www.xtreemfs.org/all_features.php *(Accessed August 2018).*

[12] Liu Aigui, "Test method and test tool for the distributed file system", 2012 - www.prog3.com/article/1970-01-01/311573 *(Accessed August 2018).*

[13] D. Sathian *et al.*, "A comprehensive Survey on Taxonomy and Challenges of Distributed File Systems", *India Journal of Science and Technology*, vol. 9, no. 11, pp. 1-8, 2016.

[14] Y. Zhou, "Large-Scale Distributed File System Survey" - http://dsc.soic.indiana.edu/publications/Large%20Scale%20Distributed%20File%20System%20Survey.pdf *(Accessed August 2018).*

[15] D. Ramesh, N. Patidar, G. Kumar, T. Vunnam, "Evolution and Analysis of Distributed File Systems in Cloud Storage: Analytical Survey", *International Conference on Computing, Communication and Automation (ICCCA)*, 2016.

[16] Unibo CDFS Benchmark website: http://lia.disi.unibo.it/research/CDFSBench/index.html *(Accessed August 2018).*