Performance Evaluation of Distributed Storage Systems

Sogand Shirinbab, Lars Lundberg, David Erman
School of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden
Sogand.Shirinbab@bth.se, Lars.Lundberg@bth.se, David.Erman@bth.se

Abstract

Distributed storage systems offer reliable and cost-effective storage of large amounts of data. We have evaluated three distributed storage systems: Compuverde, Gluster and OpenStack's Swift, using the same hardware, consisting of 24 storage nodes and a total storage capacity of 768 TB of data. Compuverde offers both structured (file based) and unstructured (block based) storage. The read/write/delete performance of Compuverde Structured is compared to Gluster, and the read/write/delete performance of Compuverde Unstructured is compared to OpenStack's Swift. These evaluations show that Compuverde outperforms the other two systems in both the structured and unstructured case. We also measure the recovery time when a storage node goes down. These evaluations show that Compuverde Unstructured recovers approximately 30 times faster than OpenStack's Swift, and Compuverde Structured recovers approximately 50 times faster than Gluster. We identify some architectural and implementation based differences that could explain the performance difference between these three systems.

1. Introduction

A number of recent studies show that the demand for storage capacity has increased rapidly during the last years. The International Data Corporation (IDC) reports: "World Wide Disk Storage systems finished 2010 with double-digit growth, and year-over-numbers reaching around 16%, but total disk storage systems capacity reached 5,127 petabytes, growing 55.7% year over year" [1]. Users want to access their data from any location. Therefore, users are moving their data to online services like Dropbox and Facebook that allow them to share and access their data from anywhere; in addition, these services provide higher data reliability than local storage disk drives. The main problem with local disk drives is that data losses are very common due to hardware errors or user mistakes. A solution which is more expensive and more reliable is Redundant Array of Independent Disks (RAID) storage. The advantages of RAID storage systems are their reliability (through redundancy), flexibility, capability to automatically manage faulty disks without losing data, and their scalability by attaching new drives. However, the scalability of RAID systems is still limited; this limitation is the main reason for designing distributed storage systems.

Distributed storage systems should solve two problems: they should be capable of sustaining rapidly growing storage demands, and they should provide efficient distribution of the stored content [2]. Two examples of distributed storage systems are OpenStack's Swift[1] and Gluster[2].

In this report we evaluate the performance of three distributed storage systems: Compuverde, OpenStack's Swift, and Gluster. Openstack's Swift and Gluster are both open-source distributed storage systems that are available for downloading and testing.

[1] http://openstack.org/
[2] http://www.gluster.org/

2. Background

In a distributed storage system, data redundancy is usually introduced by spreading multiple replicas of data over a number of storage nodes in order to guarantee that stored data is available even when some storage nodes are unavailable. Some distributed storage systems use Distributed Hash Tables (DHTs) for mapping data to physical servers. Two examples of systems that use DHTs are Gluster and OpenStack's Swift [22].

In distributed storage systems, the most common interfaces are Web Service APIs (Application Programming Interface) like Internet Small Computer System Interface (iSCSI) [25]; REpresentational State Transfer (REST)-based [23, 24] and Simple Object Access Protocol (SOAP)-based [27]. iSCSI is an end-to-end protocol that transport I/O storage blocks (fixed-size) over IP networks and uses indexes to address data. SOAP is a simple XML-based protocol to let applications exchange information over HTTP. REST is a HTTP-based architectural style to build networked applications that allows access to stored objects by an Object Identifier (OID) (see Figure 1), i.e., no file or directory structures are supported [4]. We will refer to object-based storage system as *unstructured storage systems*.

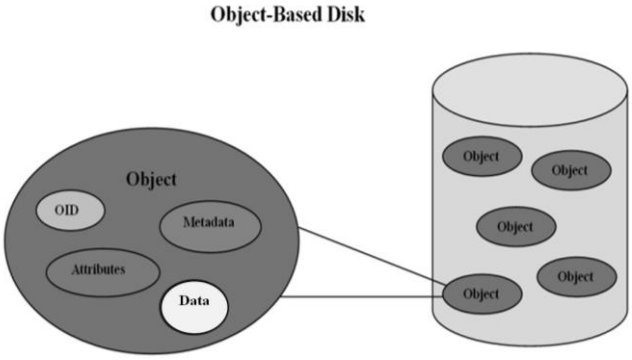


Figure 1: Object-Based Disk

There are other file storage access methods like Network File System (NFS) and Common Internet File System (CIFS) which are used for accessing storage on a private network or LAN and Web-based Distributed Authoring and Versioning (WebDAV) which is based on HTTP. These APIs are file-based (variable-size) and use a path to identify the data; we denote these as *structured storage systems*. The architecture of structured storage systems is similar to Network Attached Storage (NAS) which provide file system functionality (see Figure 2), i.e., structured storage systems support variable file and directory structures [3, 21].

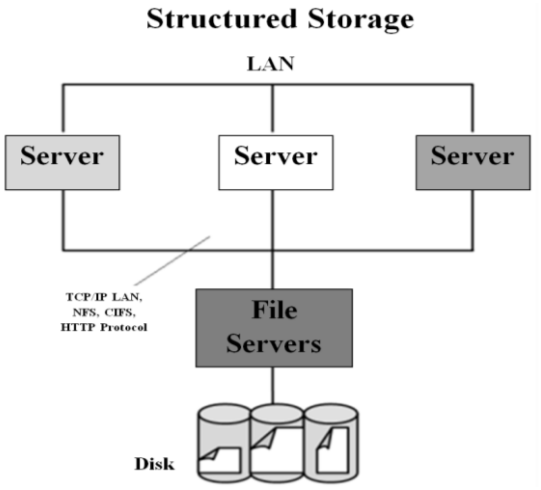


Figure 2: Structured Storage System Architecture

The most well-known distributed storage systems are Amplistor [5, 30], Caringo's CAStor [6, 31], Ceph [7], Cleversafe[3] , Compuverde[4] , EMC Atmos [8], Gluster [9], Google File System [10], Hadoop [11, 20], Lustre [12], OpenStack's Swift [19], Panasas [13], Scality[5] and Sheepdog[6]. Some of the distributed file systems could be used by other applications, i.e., BigTable is a distributed storage for structured data and it uses GFS to store log and data files [36].

As shown in Table 1 AmpliStor, CAStor, Ceph, Cleversafe and Scality are all unstructured distributed storage systems. Amplistor is designed to work with HTTP/REST. Just as in Amplistor, CAStor's Simple Content Storage Protocol (SCSP) is based on HTTP using a RESTful architecture [34]. Ceph provides an S3-compatible REST interface that allows applications to work with Amazon's S3 service. Unlike the above unstructured distributed storage systems, Cleversafe provides an iSCSI device interface, which enables users to transparently store and retrieve files as if they were using a local hard drive.

EMC Atmos is a structured distributed storage system that provides CIFS and NFS interfaces, as well as web standard interfaces such as SOAP and REST. Other distributed file systems such as Google File System, Hadoop Distributed File System (HDFS), Lustre and Panasas provide a standard POSIX API. Sheepdog is the only distributed storage system which is based on Linux QEMU/KVM and is used for virtual machines. The distributed storage systems use either Striping, Multicast or Distributed Hash Tables (DHTs).

Some of the distributed file systems are also used for computing purposes, e.g., the Hadoop Distributed File System (HDFS) which is designed based on the Data Intensive Scalable Computing (DISC) architecture in order to distribute storage and computation across many servers. HDFS stores file system metadata and application data separately and users can reference files and directories by paths in the namespace (a HTTP browser can be used to browse the files of an HDFS instance) [35]**.** Lustre is an object-based file system used mainly for computing purposes. The Lustre architecture is designed for High Performance Computing (HPC) and is composed of three components: Metadata Servers (MDSs), Object Storage Servers (OSSs) and Clients. It uses striping to distribute data across a certain number of objects**.** In addition to Hadoop and Lustre, Panasas is also used for computing purposes and similar to Lustre, it is designed for HPC. It provides parallel and redundant access to object storage devices (OSDs), per-file RAID, distributed metadata management, consistent client caching, file locking services, and internal cluster management**.**

Scality uses a ring storage system which is based upon a Distributed Hashing Mechanism with transactional support and failover capability for each storage node**.** The Sheepdog architecture is fully symmetric and there is no central node such as a meta-data server (Sheepdog uses the Corosync cluster engine [32] to avoid metadata servers). Sheepdog provides an object (variable-sized) storage and assigned a global unique id to each object. In Sheepdog's object storage, target nodes calculated based on consistent hashing algorithm which is a schema that provides hash table functionality and each object replicated to 3 nodes to avoid data lost [33].

The remaining distributed storage systems in Table 1 are Compuverde, Gluster and OpenStack's Swift. We could port these three systems to the same hardware platform (see Section 3), thus making it possible to compare their performance (see sections 4 and 5). In subsections 2.1, 2.2, and 2.3, we discuss these three systems in detail.

[3] http://www.cleversafe.com/
[4] http://compuverde.com/
[5] http://www.scality.com/
[6] http://www.osrg.net/sheepdog/

	INTERFACE				SOLUTION			METADATA	
	Unstructured		Structured						
	Web Service APIs (REST, SOAP)	*Block-based APIs (iSCSI)*	*File-based APIs (CIFS, NFS)*	*Other APIs (WebDAV, FTP, Proprietary API)*	DHT	Multicast	Striping	Centralized	Distributed
AmpliStor	X	-	-	-	-	-	X		X
Caringo's CAStor	X	-	X	-	-	X	-	X	-
Ceph	X	-	-	-	-	-	X	-	X
Cleversafe	-	X	-	-	-	-	X	X	-
Compuverde	X	-	X	X	-	X	-	-	X
EMC Atmos	X	-	X	-	-	-	X	-	X
Gluster	-	-	X	X	X	-	X	-	-
Google File System (GFS)	X	-	X	-	-	-	X	X	-
Hadoop	-	-	X	-	-	-	X	X	-
Lustre	-	-	X	-	-	-	X	X	-
OpenStack's Swift	X	-	-	-	X	-	-	-	X
Panasas	-	-	X	-	-	-	X	-	X
Scality	X	-	-	-	X	-	-	-	X
SheepDog	-	X	-	-	X	-	-	-	X

Table 1: Overview of different distributed storage systems

2.1. Compuverde

Compuverde has no single point of failure, and no separate metadata or metadata servers. Compuverde uses its own proprietary caching mechanism (SSD Caching that employs Write-back policy) [26] in the storage nodes (see Figure 3). The solution utilizes multicasting, and the supports geographical dispersion, heartbeat monitoring, versioning support, self-healing and self-configuring. Compuverde supports a flat 128 bit addresses space (for unstructured storage) and NFS/CIFS (for structured storage). The system supports Linux and Windows. Compuverde's storage solution consists of two parts: The first part is unstructured and it contains all storage nodes (clusters). The other part is the structured part of the storage solution. This part contains gateways (this corresponds to what OpenStack calls proxy servers) to communicate with storage nodes. The communication is based on TCP unicast and UDP multicast messages. Structured data storage is achieved by storing information about the structure in envelopes. An envelope is an unstructured file that is stored on the storage nodes and contains information about other envelopes and other files. The storage cluster provides mechanisms for maintaining scalability and availability of the structured data by replicating the envelopes a (configurable) number of times within the cluster as well as providing access to them by the use of IP-multicast technology.

The communication between the structured and the unstructured layers starts with an IP-multicast of a key from the gateway; this key identifies the requested envelope. Then all nodes that have the requested envelope reply with information about that envelope and what other nodes contain the requested envelope with the current execution load on the storage node. The gateway collects this

information and waits until it has received answers from more than 50% of the listed storage nodes that contains the identifier before it makes a decision on which one to select for retrieval of the file[7].

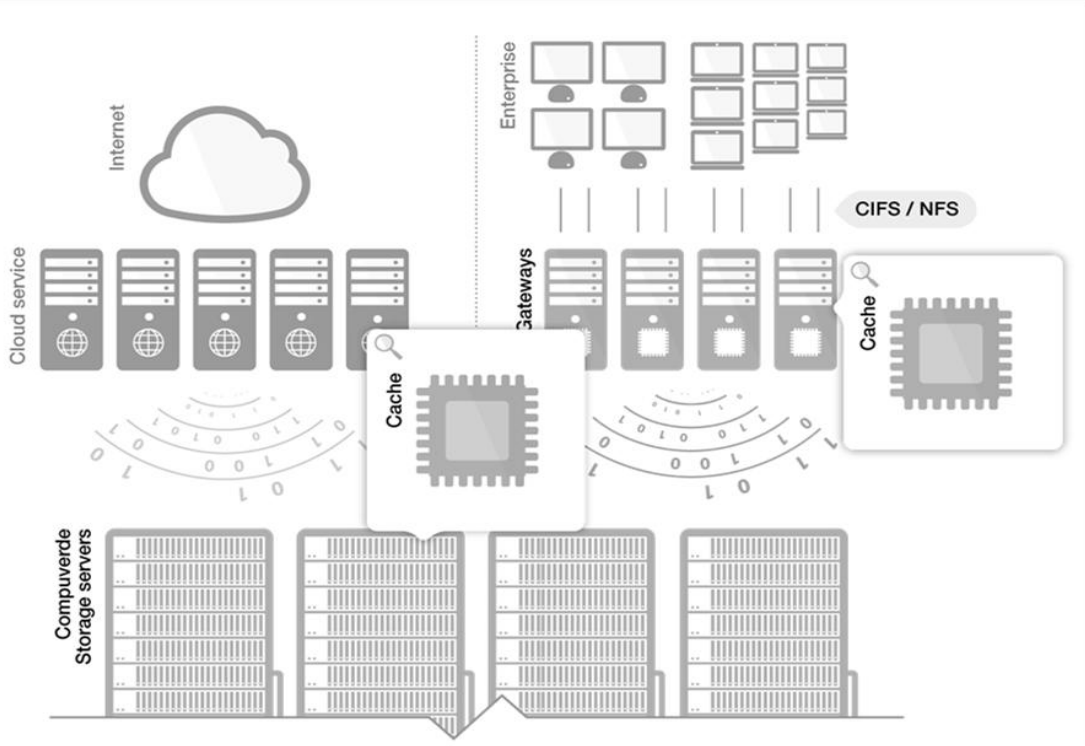


Figure 3: Compuverde System Overview

2.2. Gluster

Gluster is a structured distributed storage system. Storage servers in Gluster support both NFS and CIFS. Gluster does not provide a client side cache in the default configuration [28]. Gluster recommends hardware RAID. Software RAID works, but there are performance issues with this approach. Gluster only provides redundancy at the server level, not at the individual disk level. For data availability and integrity reasons Gluster recommends RAID 6 or RAID 5 for general use cases. For high-performance computing applications, RAID 10 is recommended. Distribution over mirrors (RAID 10) is one common way to implement Gluster (see Figure 4). In this scenario, each storage server is replicated to another storage server using synchronous writes. In this strategy, failure of a single storage server is transparent, and read operations are spread across both members of the mirror.

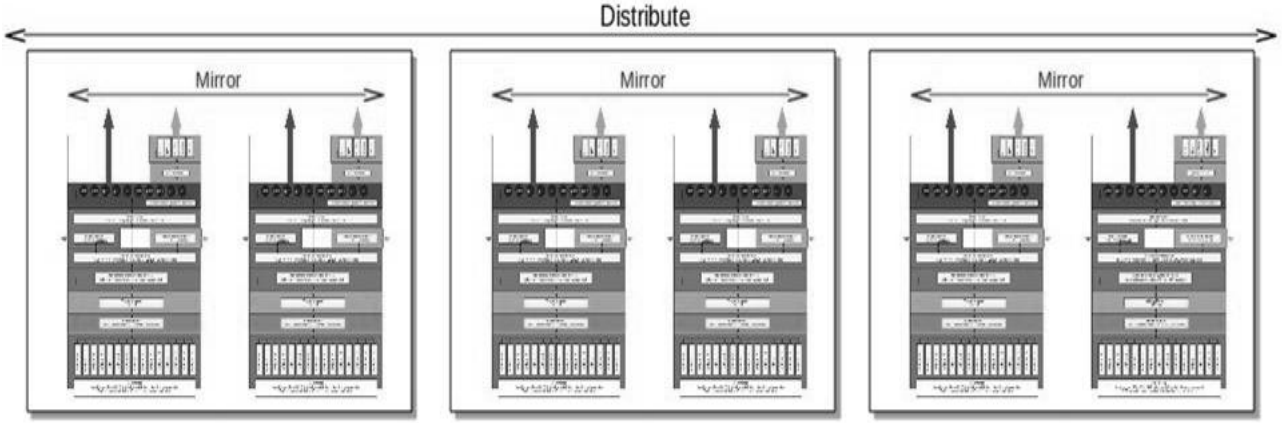


Figure 4: Gluster Distribute Over Mirrors (RAID 10)

[7] http://ilt.se/

Figure 5 shows the Elastic Hash Algorithm (EHA) used by Gluster. EHA determines where the data are stored and is a key to the ability to function without metadata. A pathname/filename is run through the hashing algorithm. After that, the file is placed on the selected storage. There are no metadata in Gluster. When accessing the file, the Gluster file system uses load balancing to access replicated instances. Gluster offers automatic self-healing [9, 14].

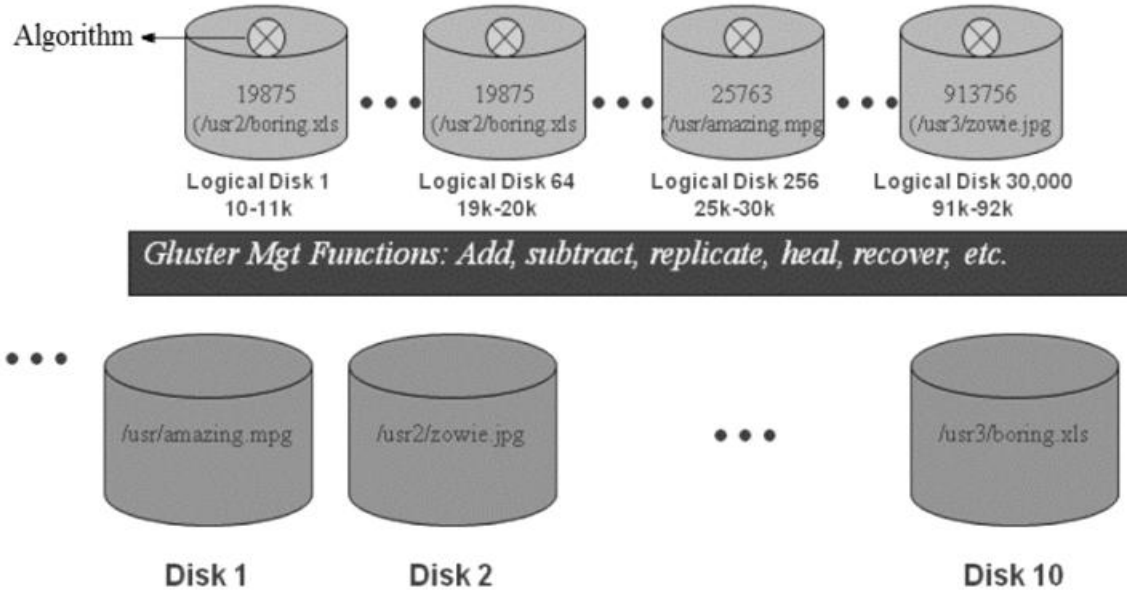


Figure 5: Gluster's Elastic Hash Algorithm (EHA)

2.3. OpenStack's Swift

OpenStack's Swift is an unstructured distributed storage system that is developed in the Python programming language and supports Get/Put/Delete methods in the same way as all other distributed storage systems which use the HTTP protocol. These methods can be used over the REST API, while containers (folders) and objects (files) are available via an HTTP API. In OpenStack's Swift, a number of "zones" are organized in a logical ring which represents a mapping between the names of entities stored on disk and their physical location. Swift is configurable in terms of how many copies (called "replicas") are written, as well as how many zones that are used. Swift tries to balance the writing of objects to storage servers so that the write and read load is distributed. The mapping of objects to zones is done using a hash function. Swift does not do any caching of actual object data but Swift-proxy can optionally work with a cache (Memcached[8]) to reduce authentication, container, and account calls [29]. In Swift, there are separate rings for accounts, containers, and objects. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine its location in the cluster. OpenStack's Swift's ring is used by the proxy server and several background processes (like replication), Swift's ring is also responsible for determining which devices are used for handoff in failure scenarios [15, 16, 17, 18, 19]. OpenStack's Swift divides the storage space into partitions. In the case that we will study, 18 bits of the GUID is used to decide on which partition a certain file should be stored. This means that there are 2^{18} = 262 144 partitions in this case. These partitions are divided into 6 zones. Zone 0 is mapped to storage nodes 0 to 3, zone 1 is mapped storage nodes 4 to 7, and zone 5 is mapped to storage nodes 20 to 23. Storage nodes 0 to 7 are handled by one switch, nodes 8 to 15 by one switch and nodes 16 to 23 by one switch (see Figure6). There are 24*16 = 384 disks in the system and the 262 144 partitions are

[8] Memcached is a distributed memory object caching system

spread out with 682 or 683 partitions on each disk (262144/384 = 682.666…). The 18 bits from the GUID will decide a partition and thus also a disk and a storage node for a file. If a file is stored on partition X, the two extra copies of the file (there are three copies of each file) are stored on partitions (X + 87 381) mod 262 144, and (X + 2 * 87 381) mod 262 144 (262 144 / 3 = 87 381.333…).

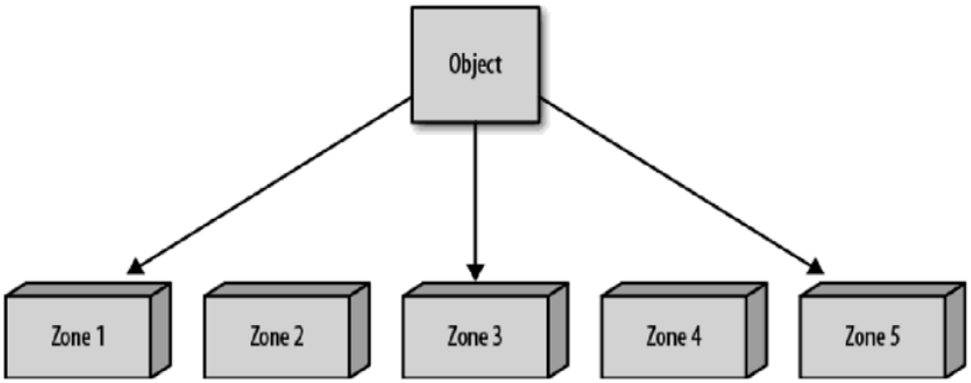


Figure 6: Swift Replicas and Zones

3. Experimental Setup

3.1. Test Configurations

Four different storage system configurations have been evaluated:

1. Compuverde Unstructured
2. Compuverde Structured
3. OpenStack's Swift (an unstructured storage system)
4. Gluster (a structured storage system)

The entire measurement takes place in two "Load Generating Clients" (see Figure 7), by running the same code for each configuration; the only part of the code that has been changed was the interface. The clients work synchronously and report the result to the "Master Controlling the Clients", which is responsible for monitoring the throughput.
The same hardware is used in each configuration. The storage system consists of 24 storage nodes, each containing sixteen 2 TB disks, i.e., a total of 32 TB for each node and 768 TB storage for all 24 nodes. With the exception of configuration 1 (Compuverde Unstructured), all accesses to the storage system are routed through four proxy (gateway) servers. In configuration 1 the clients communicate directly with the storage system.
Each proxy server has an Intel Quad processor, 16 GB RAM, and two 10 Gbit network cards. Each storage node has an Intel Atom D525 processor, 4 GB RAM, and a 1 Gbit network card. All storage nodes and proxy servers run the Linux operating system. There are four switches that are used to transmit data from four proxy servers and two load generating clients to the 24 storage nodes. The central switch is a Dell 8024F and the other three switches are Dell 7048Rs. Four Proxy Servers are connected to the Central Switch via four 2*10 Gbit Fibers. Two Load Generating Clients connected to a Central Switch via two 10 Gbit Fibers and Central switch connected to other three stand alone switches via three 4*10 Gbit Fibers.

The physical structure of the system is the same for all four cases and all measurements have been done in Load Generating Clients (see Figure 7). In configuration 1, the proxy servers are not used.

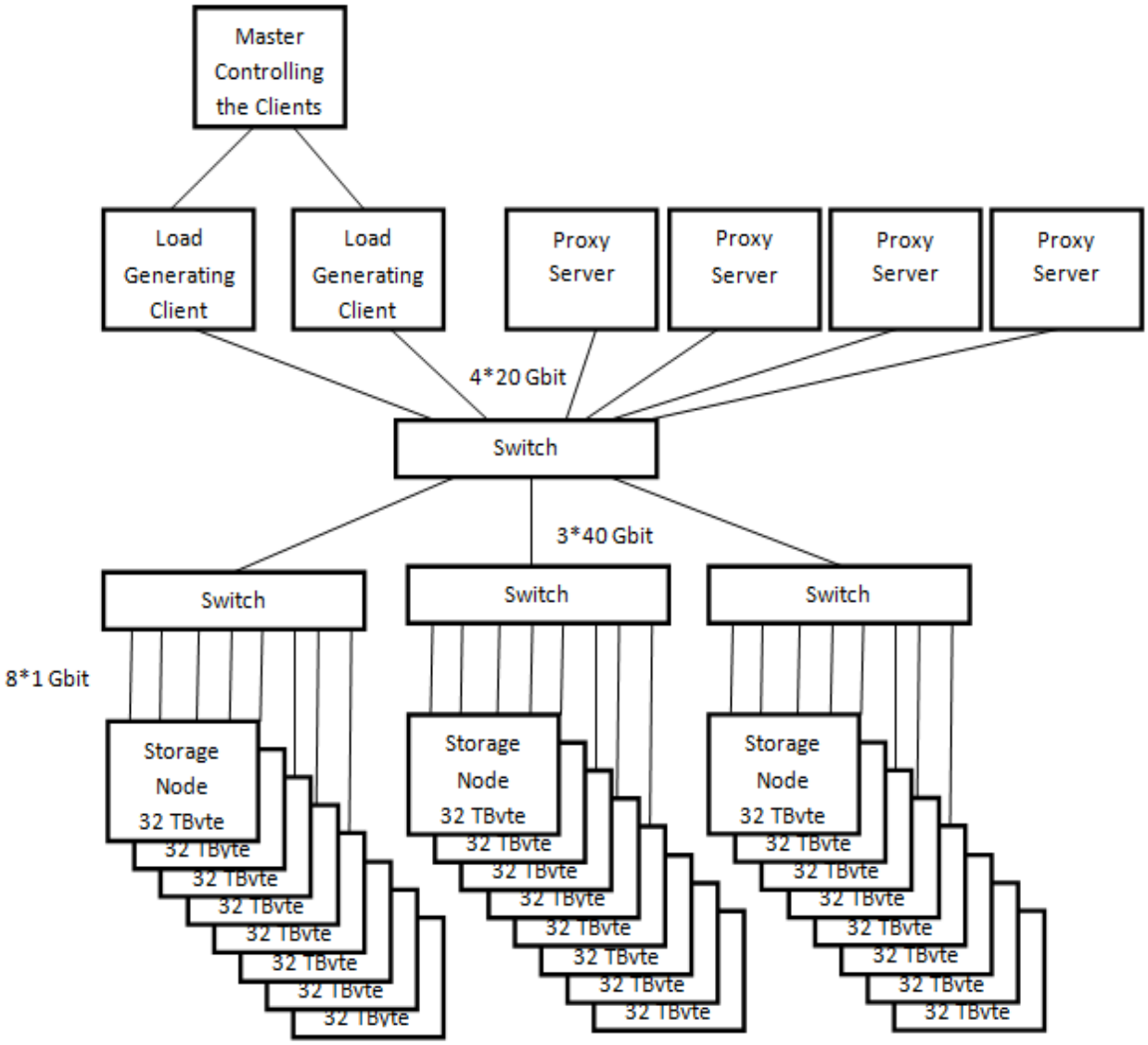


Figure 7: The physical structure of the test configurations.

The four test configurations will now be described.

3.1.1. Compuverde Unstructured

In this configuration three copies of each file are created. The proxy servers are not used, and the load generating clients communicate directly with the storage nodes.

3.1.2. Compuverde Structured

In this case two copies of each file are created. The reason for this is that this case will be compared with Gluster, and Gluster only supports two copies of each file. The two load generating clients communicate with two proxy servers each. The communication protocol between the load generating clients and the proxy servers is NFS/CIFS.

3.1.3. OpenStack's Swift

OpenStack's Swift has three copies of each file, and the two load generating clients communicate with two proxy server each.

3.1.4. Gluster

Gluster dedicates a volume to the lock file. In Gluster the storage nodes are arranged in pairs to obtain fault tolerance. This means that there are only two copies of each file. The communication protocol between the load generating clients and the proxy servers is NFS/CIFS.

3.2. Test Cases

Two kinds of tests are considered in this study: performance tests and recovery tests.

3.2.1. Performance Tests

In these test cases the read, write and delete performance are measured:

There are four test cases for each test configuration:

1. We measure write performance. In these tests, a number of clients (implemented as full speed threads, i.e., as threads that issue write requests in a tight loop without any delay and with only minimal processing done between each request) running on two servers (see Figure 7) create files of size 0 KB, 10 KB, 100 KB, 1 MB and 10 MB, respectively. Writing 0 KB corresponds to creating a file and will be reported separately. We vary the number of clients from 2 to 256, using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. A write operation is a combination of Open, Write and Close. We measure MB/s and operations/s.

2. We measure read performance. In these tests, a number of clients (implemented as full speed threads) running on two servers (see Figure 7) read files of size 0 KB, 10 KB, 100 KB, 1 MB and 10 MB, respectively. Reading 0 KB corresponds to opening a file and will be reported separately. We vary the number of clients from 2 to 256, using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. A read operation is a combination of Open, Read and Close. We measure MB/s and operations/s.

3. We measure delete performance. In these tests, a number of clients (implemented as full speed threads) running on two servers (see Figure 7) delete files of size 10 KB, 100 KB, 1 MB and 10 MB, respectively. We vary the number of clients from 2 to 256, using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. We measure operations/s.

4. For the structured storage case, we use the SPECsfs2008 performance evaluation tool from the Standard Performance Evaluation Corporation[9]. The tool can be configured to issue a number of I/O Operations per Second (IOPS), and it then measures the actual achieved throughput in terms of IOPS and the average response time.

[9] http://www.spec.org/sfs2008

During the read/write/delete performance tests for small file sizes (0KB and 10KB), test has been done by writing/reading/deleting 1,000,000 files to/from the storage nodes, but for larger file sizes (100KB, 1 MB and 10 MB) the test has been continued by writing/reading/deleting files (between 50,000 and 100,000 files) until the results become stable and then the number of operations per second and MB/s were measured.
For Gluster and OpenStack's Swift no caching is used. In order to get fair results, the test has been done for Compuverde for two cases: caching and No Caching (NC) (Gluster and OpenStack do not support caching). We just limited the caching and No Caching tests to only 1 MB files. We have recorded the CPU utilization for the storage nodes (S CPU %) and for proxy servers (P CPU %) during the tests. Compuverde Unstructured does not use any proxy servers, so there are no recordings of proxy servers CPU usage for this case.

3.2.2. Recovery Tests

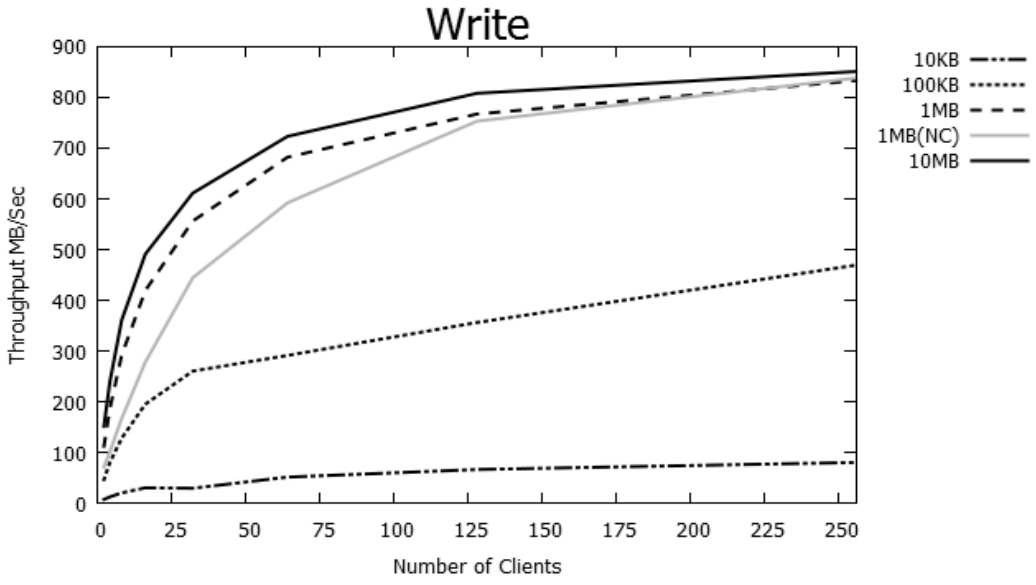
In these tests we measure how long it takes for the storage system to reconfigure itself after a node failure. We will measure recovery performance by reformatting one storage node. When a storage node is reformatted the file copies stored on that node are lost. We measure the time until the system has created new copies corresponding to the copies that were lost.

4. Read and Write Performance

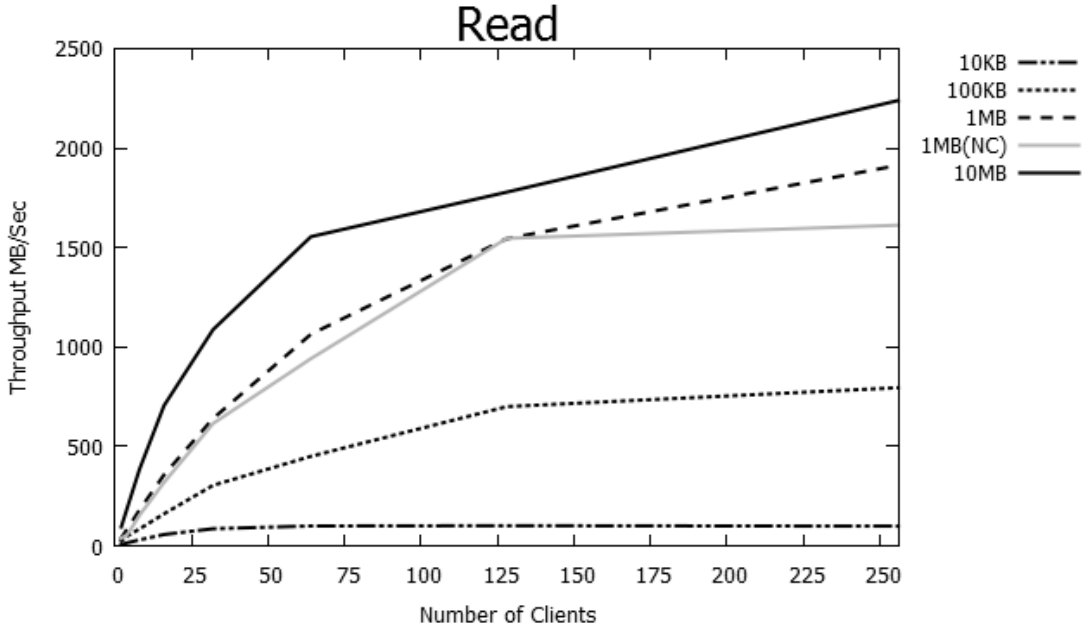
In this section we will look at the read and write performance of each of the four configurations. In Section 5 we will compare the different configurations with each other. The exact values that correspond to the figures in sections 4 and 5 can be found in Appendix A.

4.1. Compuverde Unstructured

Figures 8(a) and 8(b) show that the throughput is low when number of clients and the size of the files are small; the throughput increases when number of clients and the size of the files increase. It can also be noted that the performance in case of using cache in the storage nodes, e.g., 1 MB files, does not differ so much compared to the case that using no cache, i.e., 1 MB (NC).



(a) Write Performance Test Results (Compuverde Unstructured)

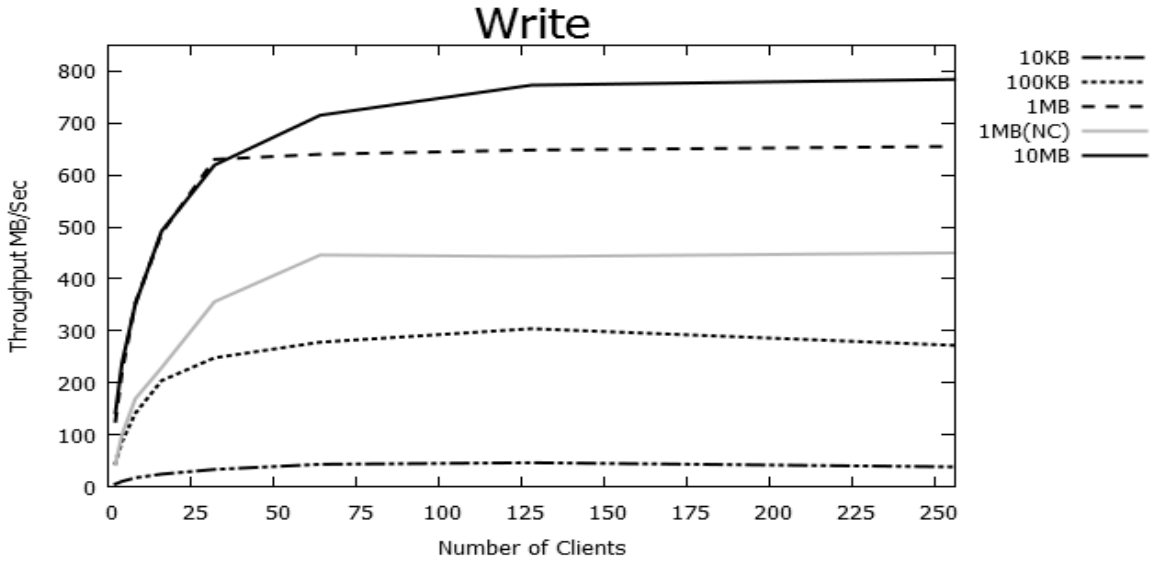


(b) Read Performance Test Results (Compuverde Unstructured)

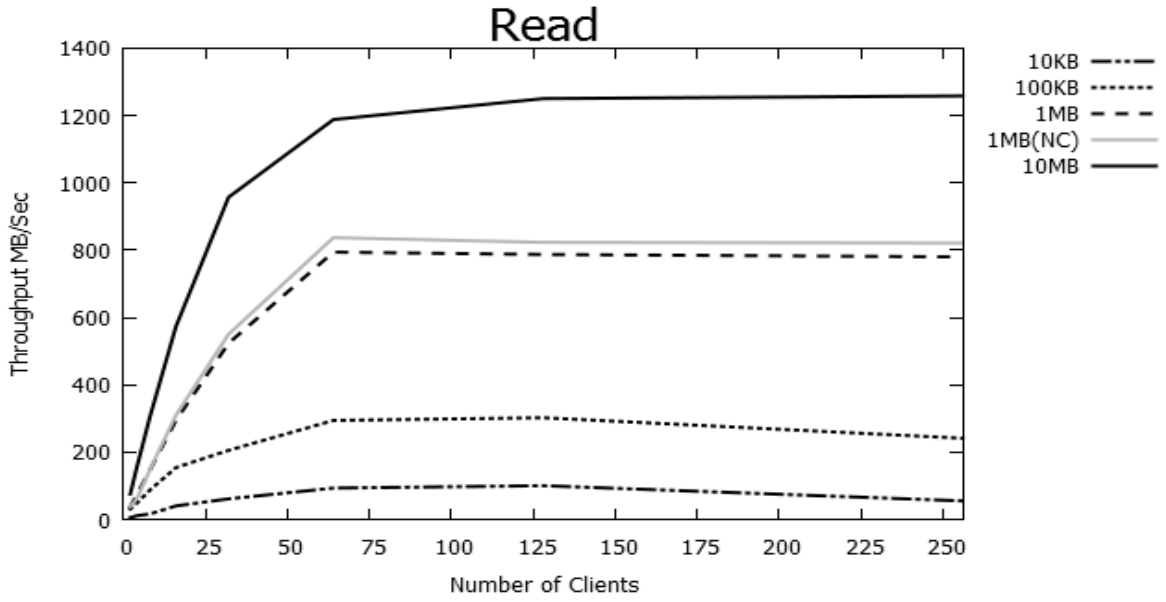
Figure 8: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously. Figure (a) shows the write performance results, and Figure (b) shows the read performance results for Compuverde Unstructured according to Appendix A, part 1, Write/Read test results.

4.2. Compuverde Structured

Figures 9(a) and 9(b) show that the data transfer rate is low when the number of clients and the size of the files are small and it increases when number of clients and size of files increase. It can also be noted that the performance difference between using caching in the storage nodes, e.g., 1 MB files, and using no caching, i.e., 1 MB (NC), is approximately a factor of 1.5 when writing; there is no significant difference between caching and no caching when reading.



(a) Write Performance Test (Compuverde Structured)

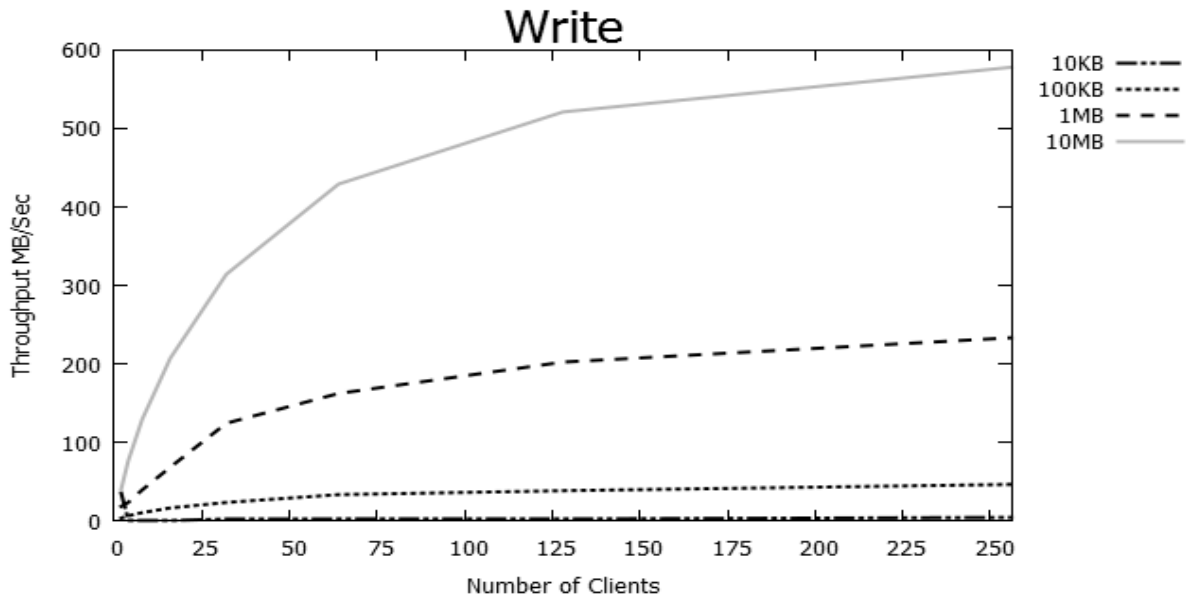


(b) Read Performance Test (Compuverde Structured)

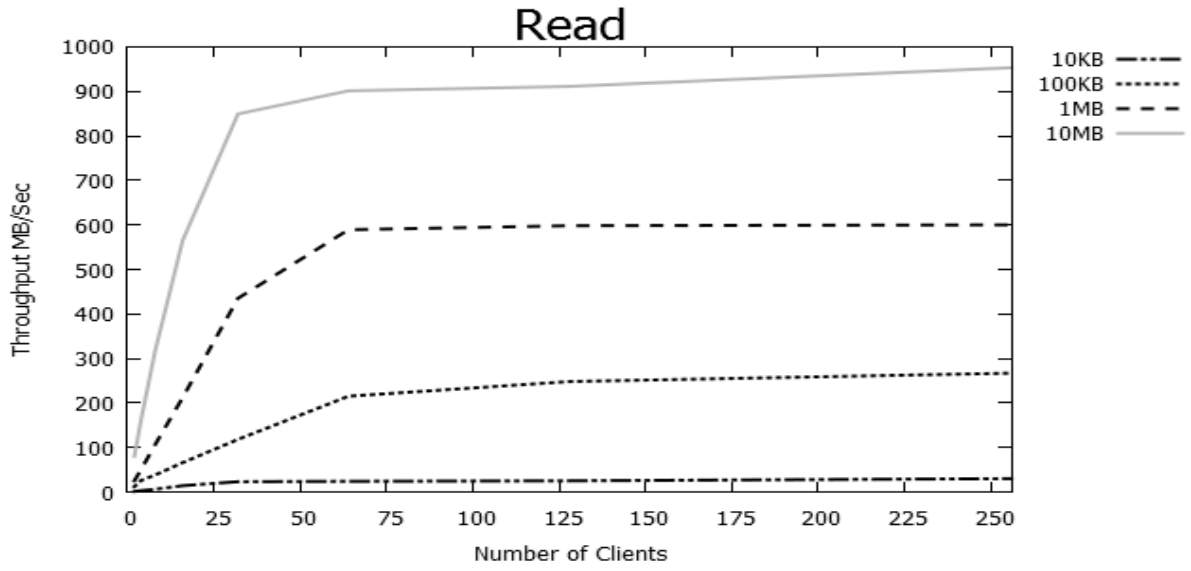
Figure 9: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously. Figure (a) shows the write performance results, and Figure (b) shows the read performance results for Compuverde Structured according to Appendix A, part 2, Write/Read test results.

4.3. OpenStack's Swift

Figures 10(a) and 10(b) show that in cases of writing/reading the files of large size (10MB), the data transfer rate increases rapidly when the number of the clients increases. While in case of writing files with size of 1MB and less the curve is quite stable.



(a) Write Performance Test (OpenStack)

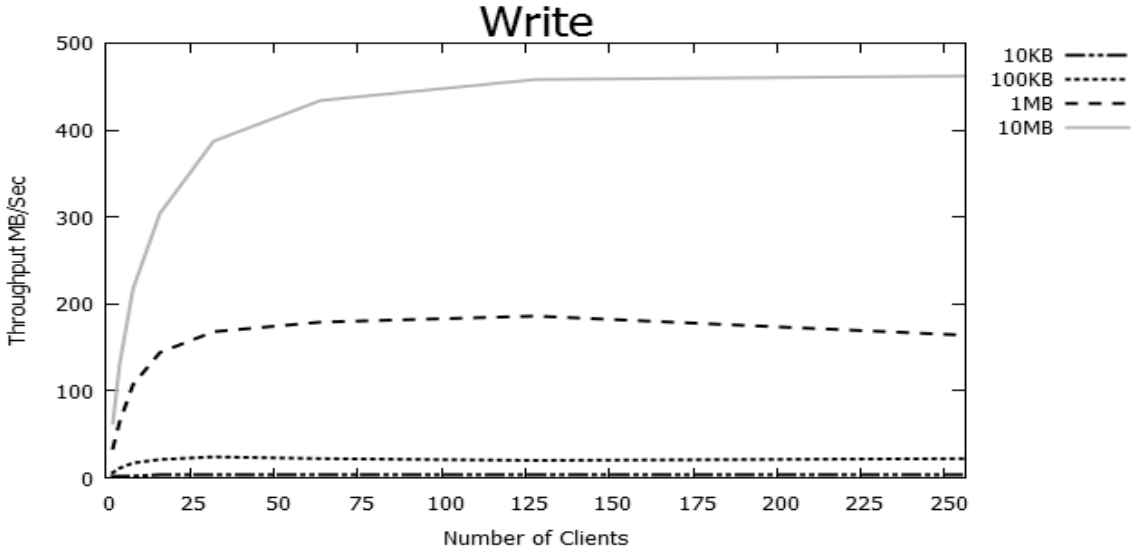


(b) Read Performance Test (OpenStack)

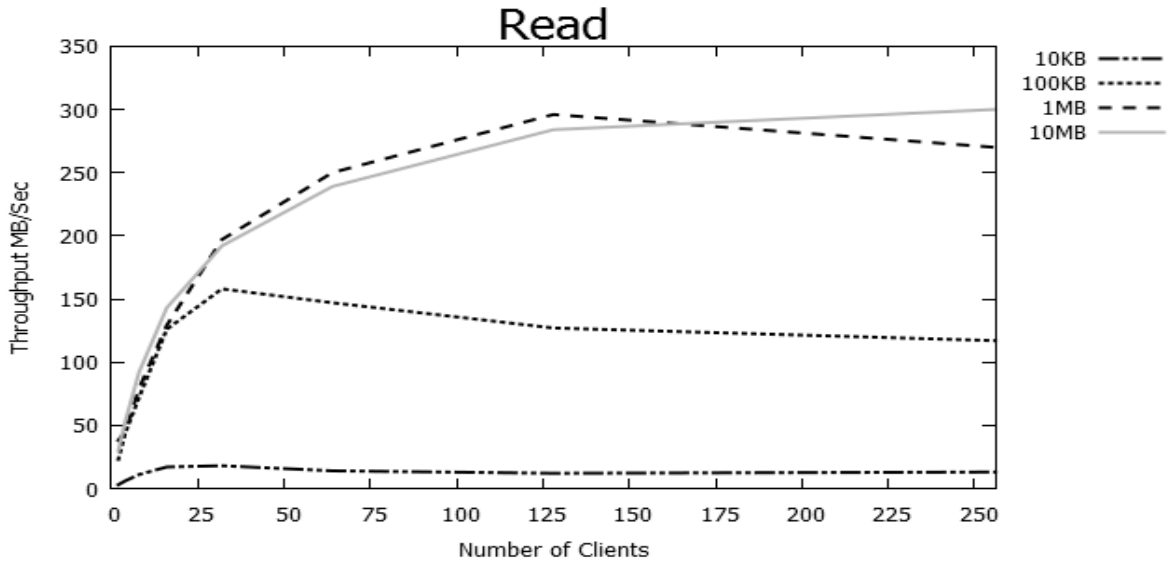
Figure 10: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously. Figure (a) shows the write performance results, and Figure (b) shows the read performance results for OpenStack according to Appendix A, part 3, Write/Read test results.

4.4. Gluster

Figures 11(a) and 11(b) show that the data transfer rate for large files increases when the number of clients increases. However, for smaller files the transfer rate does not increase so much when the number of clients increases. In fact, when the number of clients exceeds a certain values the transfer rate starts to decrease. The reason for this is probably that Gluster contains contention bottlenecks internally. The tables in part 4 of Appendix A show that the utilization for the storage nodes never exceeds 50% for Gluster. For the other test configurations we get much higher utilization values. This is an indication that there are internal performance bottlenecks in Gluster.



(a) Write Performance Test (Gluster)



(b) Read Performance Test (Gluster)

Figure 11: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously. Figure (a) shows the write performance results, and Figure (b) shows the read performance results for Gluster according to Appendix A, part 4, Write/Read test results.

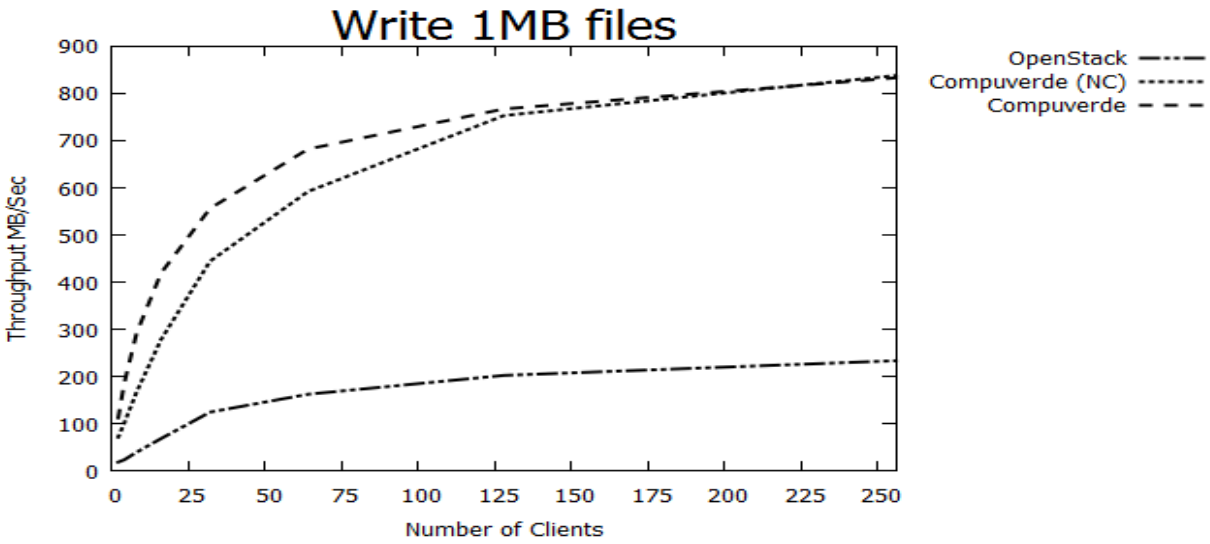
5. Comparing the Distributed Storage Systems

We have evaluated two unstructured storage systems (OpenStack's Swift and Compuverde Unstructured) and two structured storage systems (Gluster and Compuverde Structured). In Section 5.1 we compare the performance of the two unstructured systems with each other and in Section 5.2 we compare the performance of the two structured systems with each other. In Section 5.3 we compare the time to recreate all the file copies in a storage system in case one of the storage nodes fails.

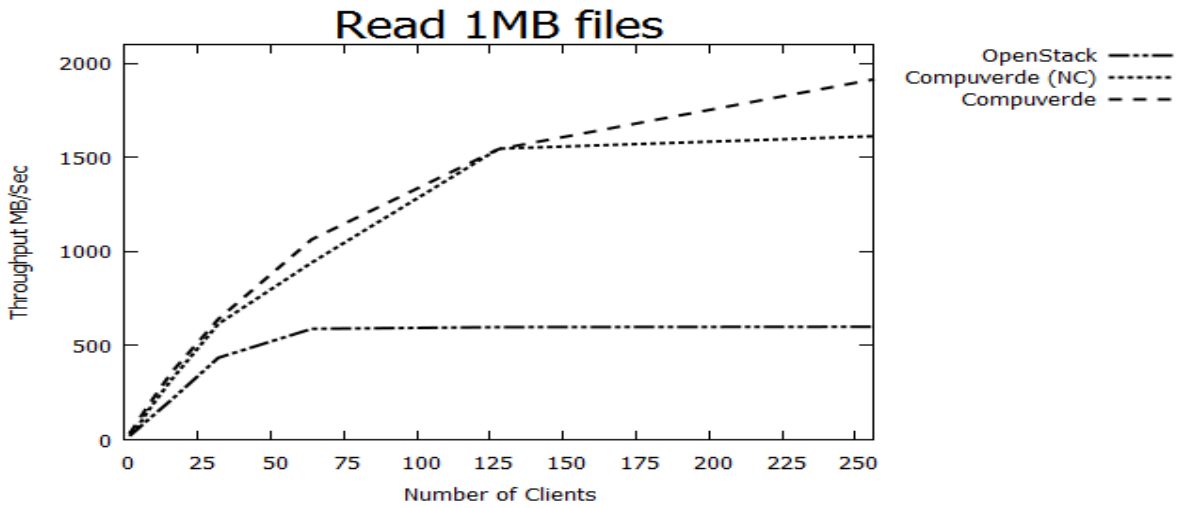
5.1. Compuverde Unstructured vs. OpenStack's Swift

We talked to several online storage provider companies and it turned out that most of their users store small files with an average size of 1 MB. Therefore, the performance tests (Write/Read/Delete) are compared only for 1 MB. Figure 12(a) shows that, the throughput of Compuverde Unstructured for 256 clients (both when using caching and no caching (NC)) was roughly 800 MB/s, while for OpenStack's Swift it was around 200 MB/s. Figure 12(b) shows that, the throughput of Compuverde Unstructured for 256 clients (both when using caching and no caching (NC)) was roughly 1600MB/sec to 1900 MB/sec, while for OpenStack's Swift it was around 600 MB/s. The create files performance test has been done by creating (writing) 0 KB files. The tables in Appendix A that correspond to Figure 12(c) show that the performance of Compuverde Unstructured for 256 clients was 10118 operations/s in case of caching and 6500 operations/s in case of no cache (NC); for OpenStack's Swift it was 600 operations/s. The open files performance test has been done by opening (reading) 0 KB files. Performance test results listed in the tables in Appendix A that correspond to Figure 12(d) show that the performance of Compuverde Unstructured for 256 clients was 11153 operations/s in case of caching and 12826 operations/s in case of no cache (NC); for OpenStack's Swift it was 4500 operations/s. The delete files performance test has been done by deleting files with a

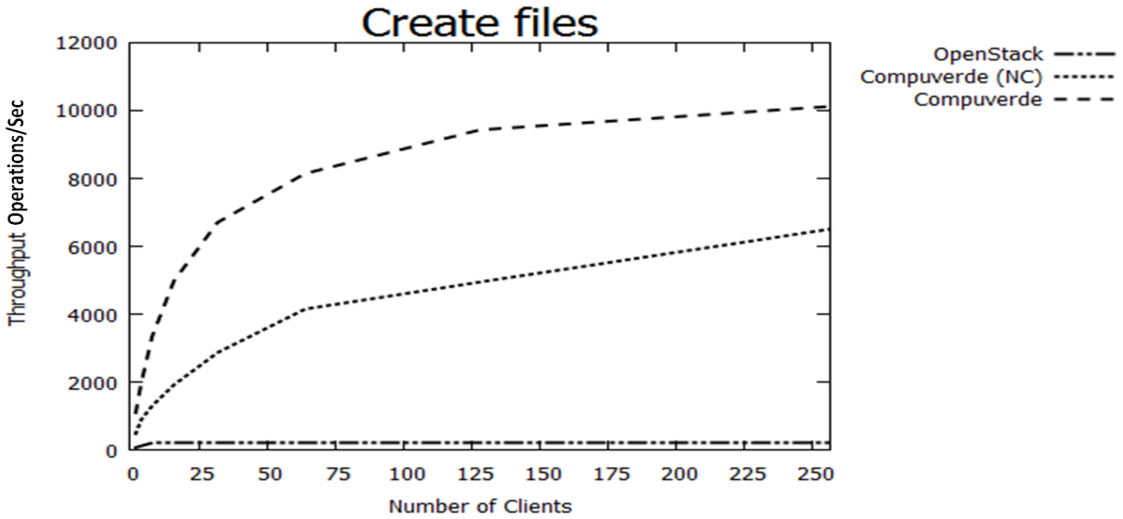
size of 1 MB. The tables in Appendix A that correspond to Figure 12(e) show that the performance of Compuverde Unstructured for 256 clients was 9956 operations/s in case of caching and 8145 operations/s in case of no cache (NC); for OpenStack's Swift it was 498 operations/s.



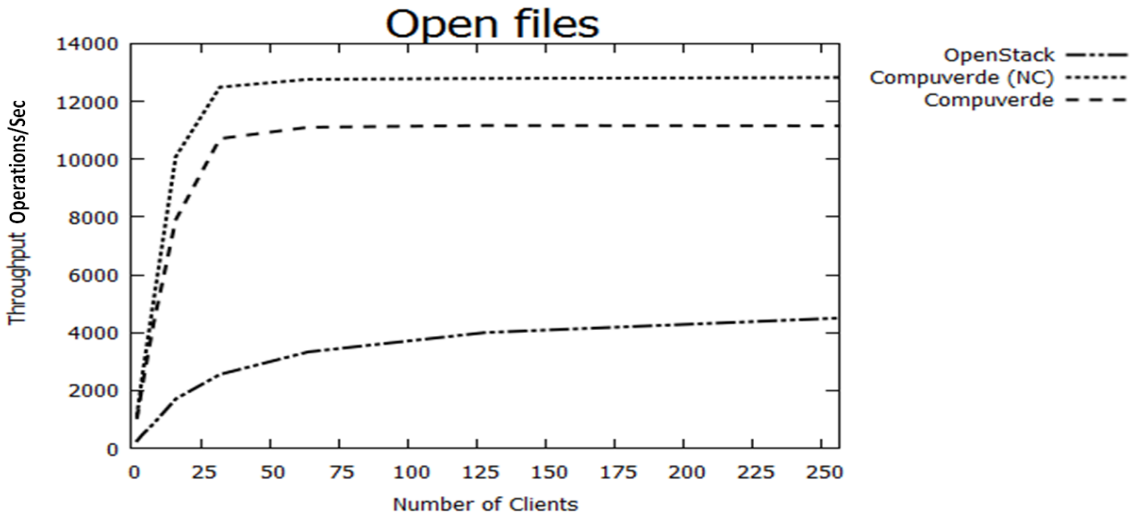
(a) Write performance Compuverde Unstructured vs. OpenStack's Swift



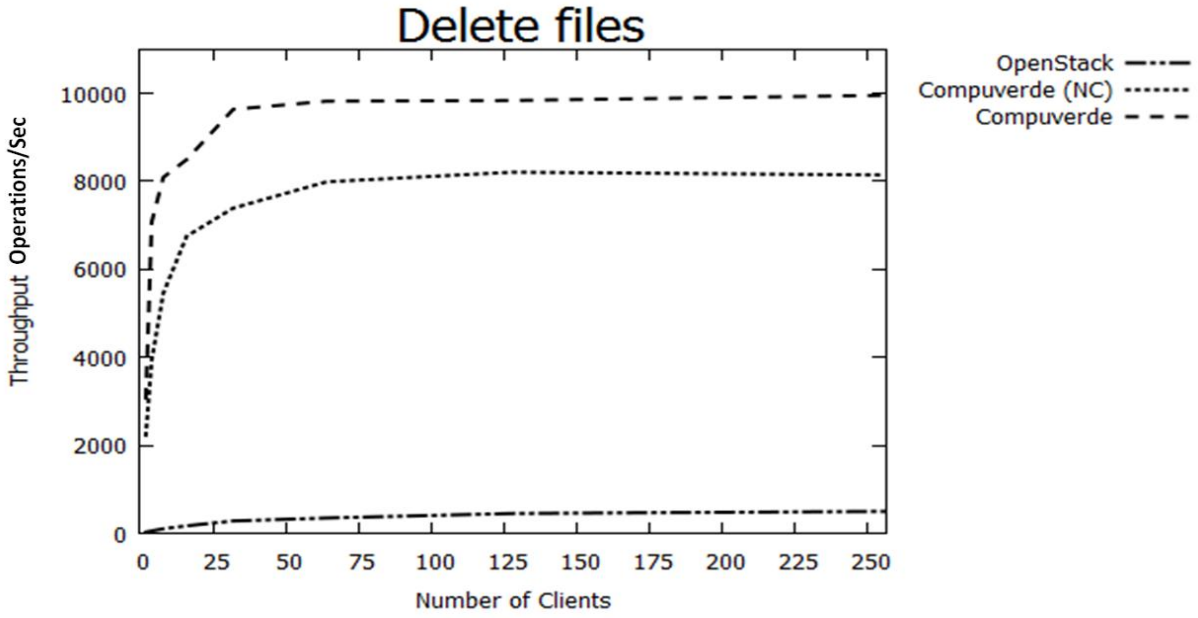
(b) Read performance Compuverde Unstructured vs. OpenStack's Swift



(c) Create files performance Compuverde Unstructured vs. OpenStack's Swift



(d) Open files performance Compuverde Unstructured vs. OpenStack's Swift



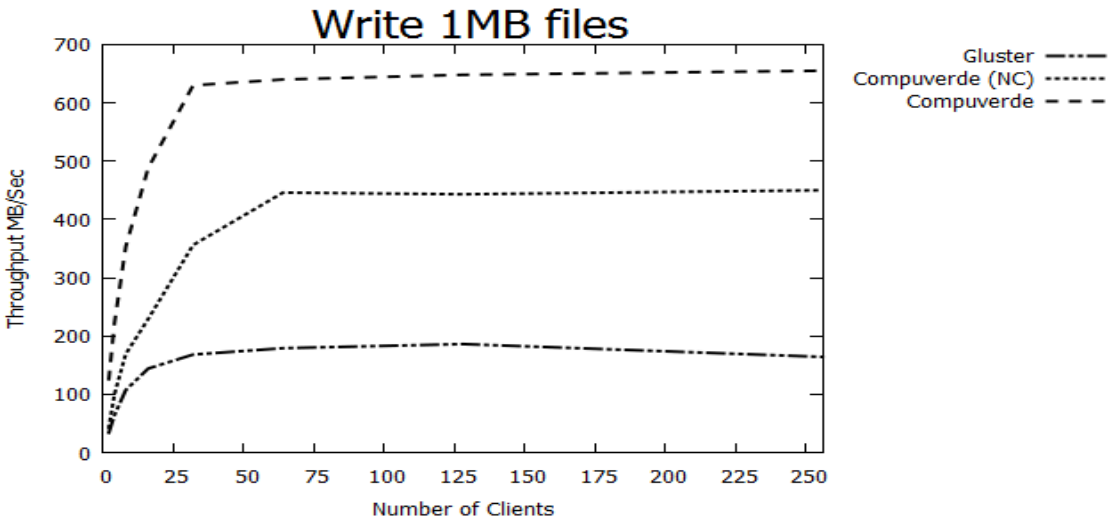
(e) Delete file performance Compuverde Unstructured vs. OpenStack's Swift

Figure 12: Figure (a) shows the comparison between write performance of Compuverde Unstructured and OpenStack's Swift for files of 1 MB. Figure (b) shows the comparison between read performance of Compuverde Unstructured and OpenStack's Swift for files of 1 MB. Figure (c) shows the comparison between create file performance of Compuverde Unstructured and OpenStack's Swift. Figure (d) shows the comparison between open file performance of Compuverde Unstructured and OpenStack's Swift. Figure (e) shows the comparison between delete performance of Compuverde Unstructured and OpenStack's Swift for files of 1 MB. All values are provided in Appendix A.

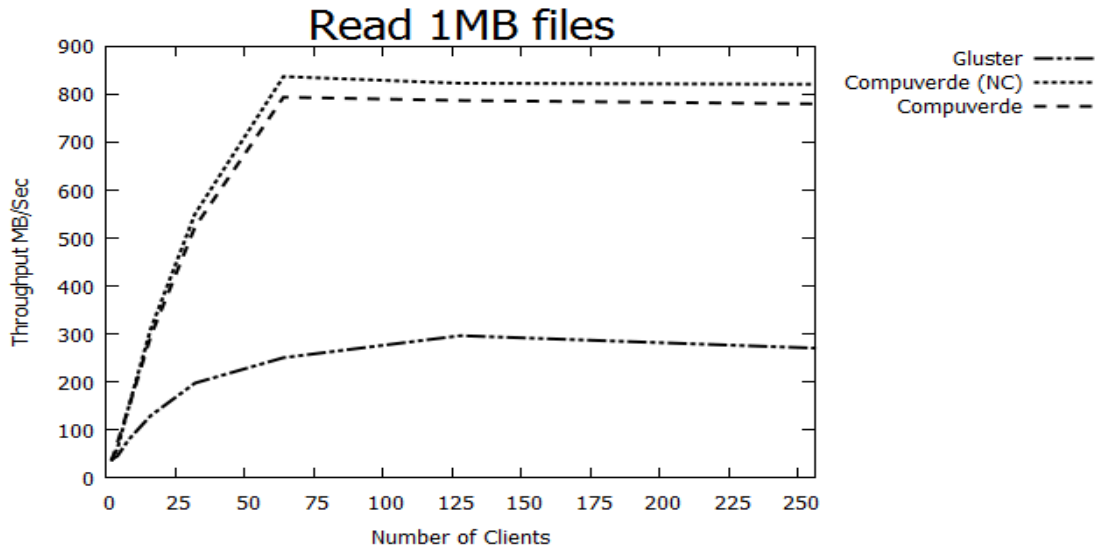
5.2. Compuverde Structured vs. Gluster

The write/read/delete performance tests have been done only for 1 MB according to the conducted interviews with the on-line storage providers. The tables in Appendix A that correspond to Figure 13(a) show that the throughput of Compuverde Structured for 256 clients was 655 MB/s in case of caching and 450 MB/s in case of no cache (NC); for Gluster it was 164 MB/s. The tables in Appendix A that correspond to Figure 13(b) show that the throughput of Compuverde Structured for 256 clients

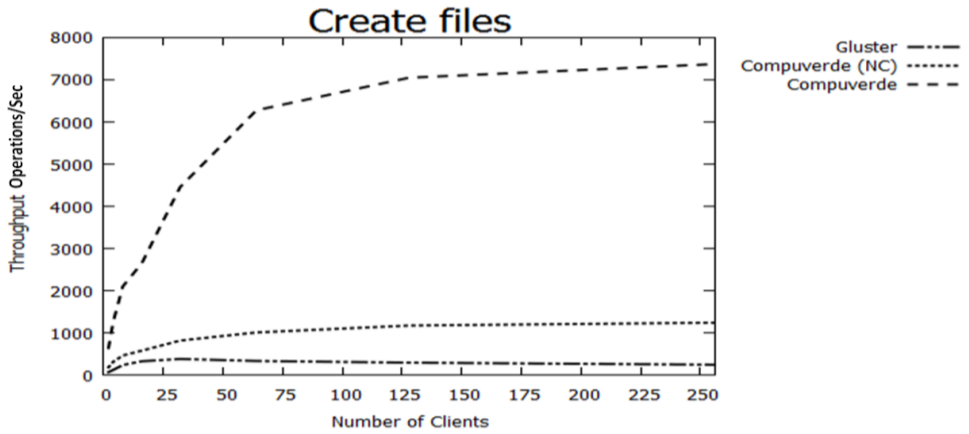
was 780 MB/s in case of caching and 821 MB/s in case of no cache (NC); for Gluster it was 270 MB/s. The create files performance test has been done by creating (writing) 0 KB files. The tables in Appendix A that correspond to Figure 13(c) show that the performance for Compuverde Structured for 256 clients was 7370 operations/s in case of caching and 1239 operations/s in case of no cache (NC); for Gluster it was 241 operations/s. The open files performance test has been done opening (reading) files of 0 KB size. The tables in Appendix A that correspond to Figure 13(d) show that the performance for Compuverde Structured for 256 clients was 11116 operations/s in case of caching and 12458 operations/s in case of no cache (NC); for Gluster it was 1029 operations/s. The delete files performance test has been done by deleting files of 1MB size. The tables in Appendix A that correspond to Figure 13(e) show that the performance for Compuverde Structured for 256 clients was 3548 operations/s in case of caching and 3367 operations/s in case of no cache (NC); for Gluster it was 441 operations/s.



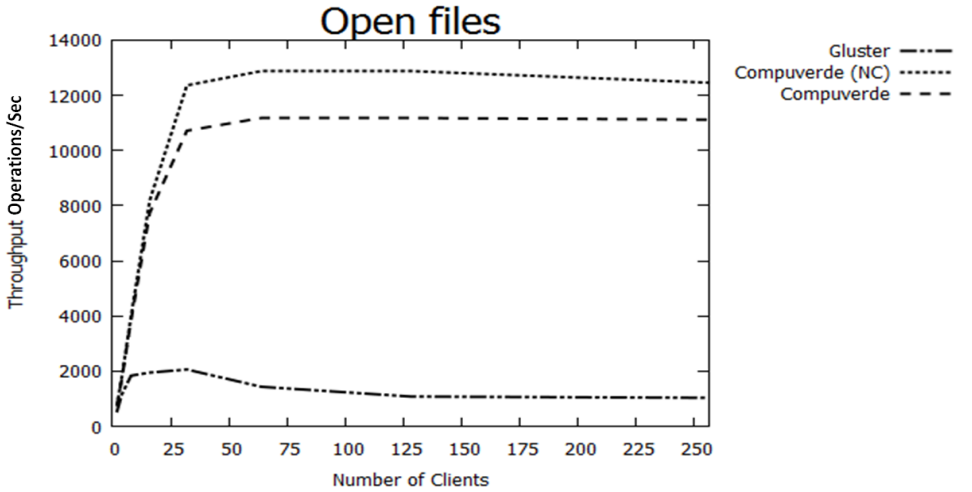
(a) Write performance Compuverde Structured vs. Gluster



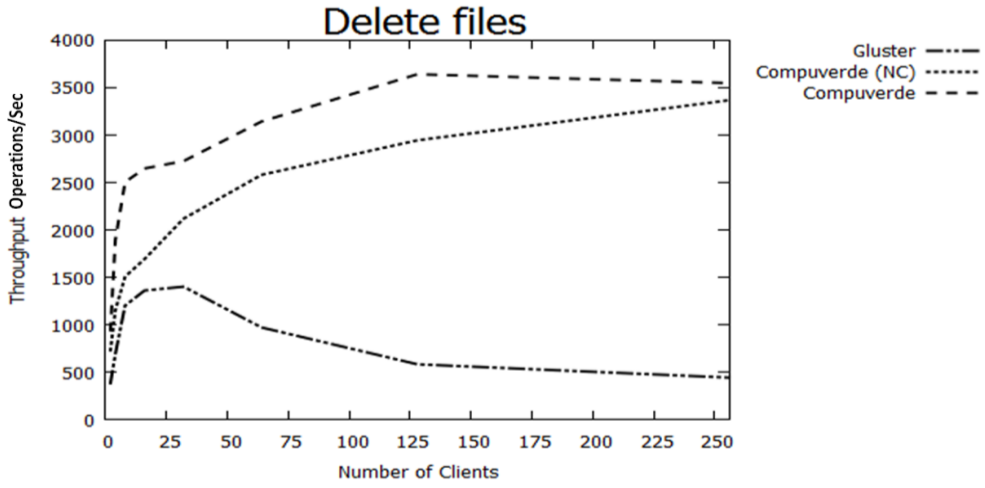
(b) Read performance Compuverde Structured vs. Gluster



(c) Create files performance Compuverde Structured vs. Gluster



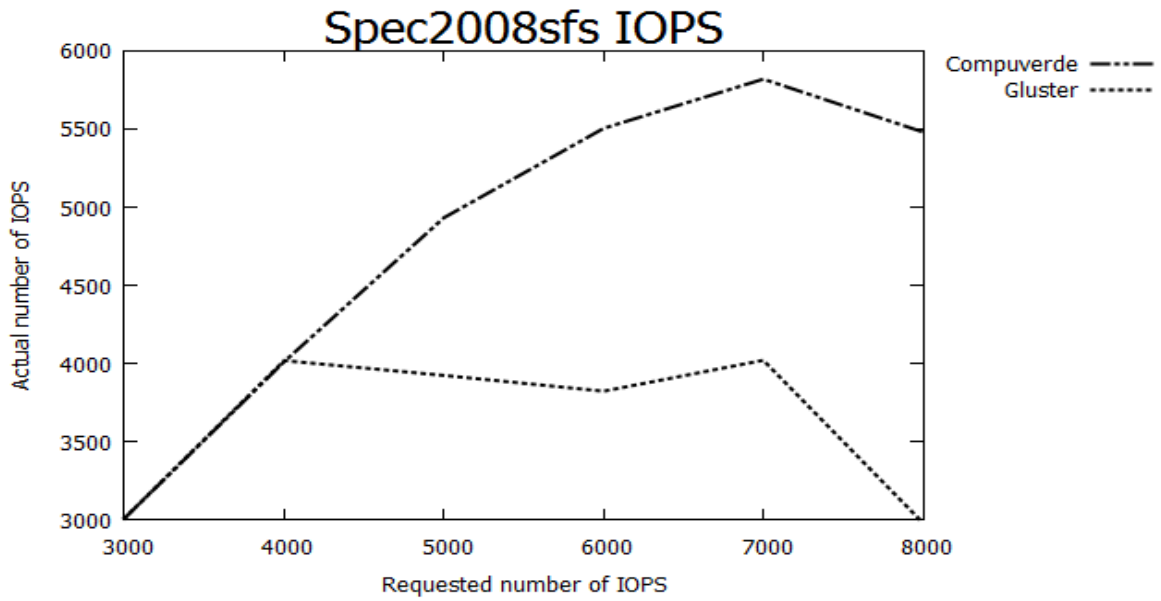
(d) Open files performance Compuverde Structured vs. Gluster



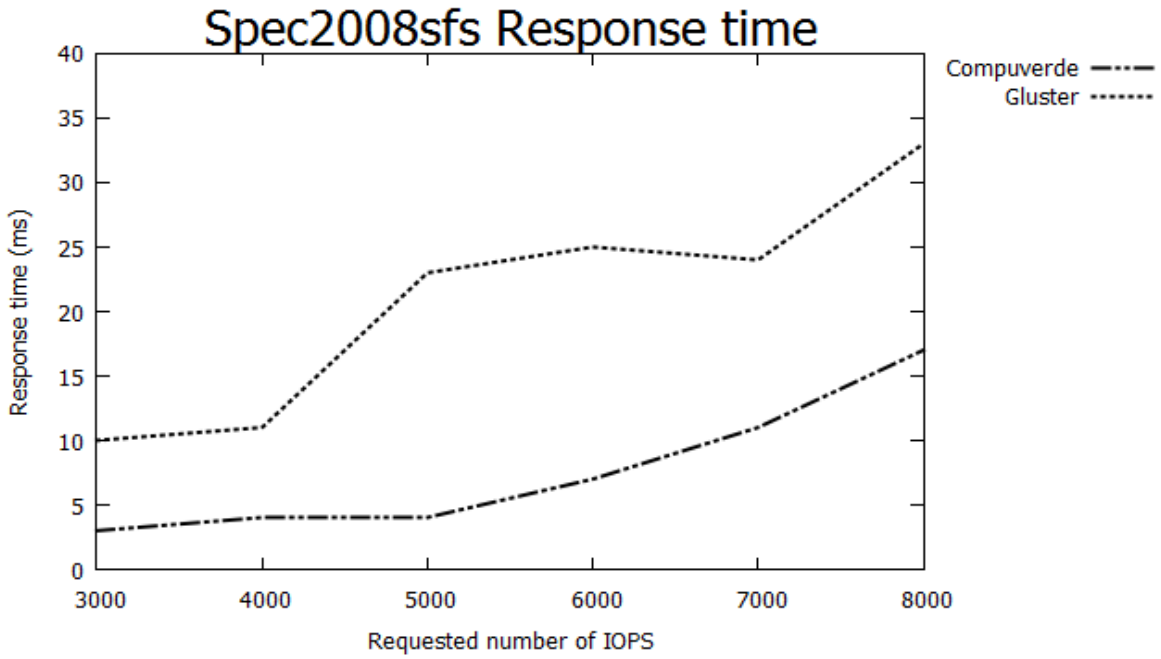
(e) Delete files performance Compuverde Structured vs. Gluster

Figure 13: Figure (a) shows the comparison between write performance of Compuverde Structured and Gluster for files of 1 MB. Figure (b) shows the comparison between read performance of Compuverde Structured and Gluster for files of 1MB. Figure (c) shows the comparison between create file performance of Compuverde Structured and Gluster. Figure (d) shows the comparison between open file performance of Compuverde Structured and Gluster. Figure (e) shows the comparison between delete performance of Compuverde Structured and Gluster for files of 1 MB. All values are provided in Appendix A.

The test results using the Spec2008sfs tool are shown in Figures 14(a) and 14(b). Figure 14(a) shows that both Compuverde Structured and Gluster meet the number of requested IOPS for the first two cases, i.e., for 3000 IOPS and 4000 IOPS. However, when the requested numbers of IOPS increased to 5000 and above, Compuverde Structured delivered a number of IOPS relatively near to the requested one, while Gluster delivers a number of IOPS that is significantly smaller than the requested number of IOPS. Figure 14(b) shows the result of response time test that has been obtained using the Spec2008sfs performance evaluation tool. Compuverde's response time is in the range of 3.5 ms to 17 ms, while for Gluster the response time is between 10.1 ms and 33.3 ms.



(a) Performance evaluation Compuverde Structured vs. Gluster



(b) Performance Evaluation Compuverde Structured vs. Gluster

Figure 14: Figure (a) and (b) show the comparison between the performance of Compuverde Structured and Gluster by using the Spec2008sfs tool.

5.3. Recovery Test

We did the recovery test for all four different configurations (Compuverde Unstructured, Compuverde Structured, OpenStack's Swift and Gluster). The same recovery test has been run twice for each configuration.

Compuverde Unstructured	19 minutes (1140 s)	18 minutes (1080 s)
Compuverde Structured	22 minutes (1320 s)	22 minutes (1320 s)
OpenStack	9 hours 27 minutes (34020 s)	10 hours 16 minutes (36960 s)
Gluster	18 hours 27 minutes (66420 s)	18 hours 29 minutes (66540 s)

Table 2: Recovery Test Results

As been shown in Table 2, the recovery time for Compuverde Unstructured was 18-19 minutes and the recovery time for OpenStack's Swift was 9 hours and 27 minutes. This means that the recovery time for Compuverde Unstructured is almost 30 times faster than that of OpenStack's Swift. One reason could be that OpenStack is written in Python language which is considered being a programming language that does not perform so fast. The other reason is that OpenStack uses the rsync[10] command that is responsible for maintaining object replicas, consistency of objects and perform update operations. It seems that using rsync command introduces a significant overhead which causes a performance decrease. The situation is similar for Compuverde Structured with a recovery time of 22 minutes compared to Gluster with recovery time of 18 hours and 27 minutes. Compuverde Structured recovery time is 50 times faster than Gluster recovery time. Gluster also uses rsync for replication, and also in this case we believe that this is one reason for the performance reduction. Another reason for the low performance of Gluster compare to Compuverde Structured is the architecture that used by Gluster for replication. In Gluster the proxy servers are doing the self-healing while in Compuverde Structured storage nodes are performing the self-healing by themselves without involving any proxy servers which results in many-to-many replication pattern. Consequently, it seems that it is much more efficient to build the synchronization protocols from scratch than to base them on standard software applications such as rsync (which is used in both OpenStack and Gluster). Also, the centralized recovery approach in Gluster (i.e., using the proxy servers) does not seem to be as efficient as the distributed approach (i.e., using the storage nodes themselves) in Compuverde. The use of the Phyton programming language is probably a contributing factor to the relatively low performance of OpenStack.

6. Discussion

Compared to conventional centralized storage systems, a distributed storage system allows for more flexible scaling in several dimensions. It allows for not only increased performance and redundancy, but also affords improved energy efficiency and lowering the carbon footprint of the system. For instance, by removing the need for a central, very high-powered storage controller, replacing it with low-cost and low wattage storage nodes, such as the ones used in the experiments presented in this paper, the power used by the system can be decreased.

[10] rsync is a file transfer program for Unix-like systems.

Furthermore, by decoupling the intelligence for providing the storage service, and placing it in the storage nodes, not only is redundancy improved, but it also allows for exchanging the individual nodes for nodes with a lower carbon footprint as technology advances. This makes the system more flexible with respect to the hardware used to build the system, and makes it possible to take immediate advantage of improvements in sustainable technologies. One example would be exchanging conventional hard drive for solid state drives as prices decrease. Another advantage of moving the system logic to the storage nodes is that this allows system performance to scale linearly with the number of nodes. This is particularly apparent when employing multicast.

7. Conclusions

The performance evaluations of the unstructured storage systems show that the open, read, write, and delete performance of Compuverde Unstructured is significantly higher than for OpenStack's Swift. The performance advantage of Compuverde Unstructured is particularly clear when the load is high, i.e., when the number of clients that issue simultaneous accesses to the system is high. The evaluations show that the performance advantage of Compuverde Unstructured is not a result of caching in the storage nodes, i.e., the performance difference using cache and no cache (NC) is relatively small. One possible explanation to the low performance of OpenStack's Swift is that the OpenStack system is written in the Python programming language, which is not a compiled language and is slow compared to C/C++. The other explanation for OpenStack's low performance could be the proxy servers. In the OpenStack architecture, data has to flow through proxy servers, which is a performance bottleneck; Compuverde Unstructured does not use any proxy servers and clients are connected directly to the storage nodes.

The performance evaluations of the structured storage systems show that the open, read, write, and delete performance of Compuverde Structured is significantly higher than for Gluster. The performance advantage of Compuverde Structured is particularly clear when the load is high, i.e., when the number of clients that issue simultaneous access to the system is high. The performance in terms of throughput goes down for Gluster when the load increases. This behavior indicates that there are internal bottlenecks (e.g., file level locking) and contention problems in Gluster. The evaluation using the Spec2008sfs tool show that same behavior, i.e., the performance of Compuverde Structured, in terms of throughput and response times, is better than that of Gluster and Gluster suffers from contention problems when the load increases. Compuverde Structured has some contention problems when the load increases, but not to the same extent as Gluster. The evaluations show that the performance advantage of Compuverde Structured is not a result of caching in the storage nodes. In some cases caching in the storage nodes adds a significant advantage, but the performance of Compuverde Structured is better than Gluster also without caching in the storage nodes.

The recovery test show that Compuverde recovers from a storage node failure much faster than OpenStack's Swift and Gluster. One reason for Gluster to perform slower than Compuverde Structure could be its architecture which involves proxy servers in self healing while Compuverde uses the many-to-many replication pattern and only involves storage nodes in self healing. A major factor for OpenStack's Swift to perform slower than Compuverde Unstructured could be the Python programming language. Another reason could be that Compuverde has built its own recovery protocol from scratch, whereas OpenStack and Gluster base their protocols on existing applications (e.g., rsync).

Finally, our study provides quantitative performance values measured on a large real world distributed storage system. Such values can be used for comparisons by other researchers and practitioners.

References

1. IDC (International Data Corporation), (2011, March 04). "Worldwide Disk Storage Systems Finishes 2010 with Double-Digit Growth on Strong Fourth Quarter Results, According to IDC" [Online]. Available: http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22723811&sectionId=null&elementId=null&pageType=SYNOPSIS
2. Liuis Pamies I Juarez, "On the Design and Optimization of Heterogeneous Distributed Storage Systems", University Rovira in Virgili, Department of Engineering Information in Mathematic, PHD thesis dissertation, (2011, July 19).
3. Garth A. Gibson, Rodney Van Meter, "Network Attached Storage Architecture", Magazine, Communications of the ACM, New York, (2000, November).
4. Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, Julian Satran," Object storage: The future building block for storage systems", published in: in 2nd International IEEE Symposium on Mass Storage Systems and Technologies, Sardinia,2005.
5. Santa Clara, "Amplidata Demonstrates Highly Scalable and Reliable Storage Solutions for Massive Cloud Deployments at Intel Development Forum", Article at PRNewswire, Calif. (2011, September 16).
6. Caringo CAStor (2011, September 15). "Castor the Market Leading Object Storage Engine" [Online]. Available: http://www.caringo.com/downloads/datasheets/Caringo-CAStor-Object-Storage.pdf
7. Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System", University of California, Santa Cruz, Appeared in Proceeding of the 7th Conference on Operating Systems Design and Implementation (OSDI'06), (2006, November).
8. EMC Atmos, "EMC Atmos Cloud Optimize Storage for Web Services" Whitepaper, (2010, April).
9. Gluster Inc. "An Introduction to Gluster Architecture" Whitepaper,(2011).
10. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System", Appeared in 19th ACM Symposium on Operating Systems Principles, Lake George, NY (2006, October).
11. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System", Yahoo! Sunnyvale, California USA, 2010.
12. Feiyi Wang, Sarp Oral, Galen Shipman, "Understanding Lustre FileSystem Internals", OAK RIDGE, 2009.
13. Brent Welch, Marc Unangst, Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Jim Zelenka, Bin Zhou. "Scalable Performance of The Panasas Parallel File System", FAST'08 proceedings of the 6th USENIX Conference on File and Storage Technologies, USENIX Assosiation Berkeley, CA, USA, 2008.
14. Drew Robb, "Gluster Brings Open Source to Unstructured Data", Storage Technology Features Article Published (2010, August 12).
15. OpenStack, "OpenStack Compute Admin Manual",Manual, (Nov14,2011).
16. Joe Arnold, Dr. Jinkyung Hwang, Dr. Jaesuk Ahn, "Commercialization of OpenStack: Object Storage", OpenStack conference commercialization of object storage, Korea (2010, April 26).
17. Pepple Ken,"Deploying OpenStack". O'Reilly Media. ISBN 1449311059, (August 2011).
18. OpenStack, "OpenStack Object Storage: An Overview" white paper, 2010.
19. OpenStack, LLC, "Welcome to Swift's documentation!", Swift v1.4.8-dev documentation,2011.
20. Shunsuke Mikami, Kazuki Ohta, Osamu Tatebe, "Using the Gfarm File System as a POSIX Compatible Storage Platform for Hadoop MapReduce Applications", Published in: GRID'11 Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing, IEEE Computer Society Washington, DC, USA, 2011.
21. Ranjit Noronha, Lei Chai, Thomas Talpey, Dhabaleswar K.Panda,"Designing NFS with RDMA for Security, Performance and Scalability", Technical Report OSU-CISRC-6/07-TR47, The Ohio State University, 2007.
22. Julian Dymcek, "Survey of Distributed Hash Tables", Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown WV,2011.
23. Roy T.Fielding, Richard N. Taylor, "Principles design of the modern Web architecture", published in ICSE'00 Proceedings of the 22nd international conference on software engineering, ACM New York, NY, USA, 2000.
24. Michael Jakl,"REST: Representational State Transfer", University of Technology Vienna, 2008.
25. Wang, P., "IP SAN- from iSCSI to IP-addressable Ethernet disks" Appears in: Mass Storage Systems and Technologies. Proceedings, 20th IEEE/11th NASA Goddard Conference, 2003.
26. Thanos Makatos, Yannis Klonatos, Manolis Marazkis, Michail D. Flouris, Angelos Bilas, "Using transparent compression to improve SSD-based I/O Caches",Published in EuroSys'10 Proceedings of the 5th European conference on Computer systems, ACM NewYork, NY, USA, 2010.
27. Curbera F., Duftler M., Khalaf R., Nagy W., Mukhi N., Weerawarana S. "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI", published in Internet Computing, IEEE, NY, USA, 2002.

28. Ranjit Noronha, Dhabaleswar K. Panda “IMCa: High Performance Caching Front-end for GlusterFS on InfiniBand” Network-Based Computing Laboratory, Computer Science and Engineering, The Ohio State University, 2008.
29. OpenStack Object Storage Admin Manual, OpenStack, “Consideration and Tunning”, 2011.
30. Amplidata,“Amplistor: Unbreakable Object Storage for Petabyte-Scale Unstructured Data” Whitepaper, 2011, (April 13)
31. Caringo CAStor, “CAStor: The Market Leading Object Storage Engine” Product Brief, (2011, September 15)
32. Steven C. Dake, Christine Caulfield, Andrew Beekhof. “The Corosync Cluster Engine”, Proceedings of the Linux Symposium, Ottawa, Ontrio, Canada, 2008 July 23.
33. George Parisis, “DHTbd: A Reliable Block-based storage system for High Performance clusters”, Proceedings of the IEEE/ACM CCGRID, UK, 2011.
34. Roberto Lucchi, Michel Millot, “Resource Oriented Architecture and REST”, JRC Scientific and Technical Reports, European Communities, Luxembourg, 2008.
35. Bin Fan, Wittawat Tantisiriroj, Garth Gibson, Lin Xiao, ”DiskReduce: Replication as a Prelude to Erasure Coding in Data-Intensive Scalable Computing”, Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, 2011.
36. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, “BigTable: A Distributed Storage System for Structured Data”, Journal: ACM Transactions on Computer Systems (TOCS), New York, USA, June 2008.

APPENDIX A, MEASURED PERFORMANCE VALUES

1. COMPUVERDE (UNSTRUCTURED)

Write	0 KB		0 KB (NC)		10 KB		100 KB		1 MB		1 MB (NC)		10 MB	
Clients / File size	Op/s	CPU %	Op/s	CPU %	MB/s	CPU %	MB/s	CPU %	MB/s	CPU %	MB/s	CPU %	MB/s	CPU %
2	1096	12	478	3	7	12	46	14	112	14	71	5	151	12
4	1965	24	890	5	12	24	80	23	186	20	101	8	239	23
8	3341	27	1296	8	20	36	128	27	292	25	167	12	361	25
16	4982	50	1916	13	30	70	195	50	419	35	278	23	491	37
32	6702	73	2866	18	29	90	260	75	556	54	444	27	611	48
64	8141	77	4146	21	51	95	291	73	682	72	592	49	723	60
128	9428	95	4944	30	66	97	356	74	767	76	753	58	808	73
256	10118	98	6500	55	80	98	469	77	833	89	838	72	851	75

Read	0 KB		0 KB (NC)		10 KB		100 KB		1 MB		1 MB (NC)		10 MB	
Clients / File size	Op/s	CPU %	Op/s	CPU %	MB/s	CPU %	MB/s	CPU %	MB/s	CPU %	MB/s	CPU %	MB/s	CPU %
2	1057	9	1152	4	7	5	24	4	39	4	38	3	95	3
4	2210	15	2565	15	15	12	50	5	86	4	57	3	193	3
8	4327	25	5110	25	30	24	85	8	181	5	154	4	386	5
16	7901	46	10086	45	57	30	161	12	357	7	319	5	705	7
32	10709	53	12490	50	86	50	304	26	640	15	615	13	1087	15
64	11103	55	12761	50	100	60	450	46	1064	26	941	24	1554	24
128	11165	55	12792	53	101	68	700	73	1544	40	1546	28	1778	27
256	11153	56	12826	54	99	72	795	77	1913	50	1612	43	2239	28

Delete	1 MB		1 MB (NC)	
Clients / File size	Op/s	CPU %	Op/s	CPU %
2	3078	53	2230	30
4	7076	62	3936	45
8	8092	70	5460	47
16	8495	73	6754	50
32	9636	74	7391	45
64	9822	74	7990	35
128	9840	73	8209	49
256	9956	73	8145	49

2. COMPUVERDE (STRUCTURED)

Write	0 KB			0 KB (NC)			10 KB			100 KB			1 MB			1 MB (NC)			10 MB		
Clients / File size	Op/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %
2	640	24	12	182	4	4	5	25	13	45			126	20	10	43	5	5	142		
4	1266	38	19	313	7	5	10	46	18	87			222	26	16	100	11	10	238		
8	2098	75	25	459	9	5	17	75	23	141			353	58	31	169	20	20	351		
16	2650	100	32	576	15	8	24	95	34	204			488	80	52	229	26	25	492		
32	4453	100	44	814	22	13	33	100	40	248			630	90	74	356	45	39	619		
64	6276	99	58	1007	20	13	43	96	62	278			640	93	81	446	48	60	715		
128	7048	96	63	1171	23	16	46	97	42	304			648	98	90	443	51	70	773		
256	7370	97	71	1239	25	25	38	94	39	272			655	98	93	450	53	68	784		

Read	0 KB			0 KB (NC)			10 KB			100 KB			1 MB			1 MB (NC)			10 MB		
Clients / File size	Op/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %
2	779	5	7	786	4	6	5	4	9	31			37	3	12	38	3	19	75		
4	1700	12	10	1769	11	10	11	6	15	48			73	4	26	61	4	22	154		
8	3851	24	18	3985	24	19	16	11	39	87			146	4	46	147	5	45	304		
16	7733	28	32	8189	30	34	40	25	54	154			295	6	75	310	6	72	574		
32	10716	50	47	12359	49	53	61	35	90	205			523	13	90	550	12	92	957		
64	11171	50	62	12859	50	57	93	54	96	294			794	17	98	837	23	98	1188		
128	11172	51	72	12857	51	63	100	70	95	302			787	18	99	823	22	98	1250		
256	11116	55	84	12458	50	61	55	68	82	241			780	20	99	821	23	99	1258		

Delete	1 MB			1 MB (NC)		
Clients / File size	Op/s	S CPU %	P CPU %	Op/s	S CPU %	P CPU %
2	942			730		
4	1892			1150		
8	2503			1506		
16	2647			1691		
32	2725			2117		
64	3142			2580		
128	3641			2944		
256	3548			3367		

Spec2008sfs		
Req IOPS	IOPS	Resp (ms)
2000		
3000	3007	3,5
4000	4010	4,4
5000	4931	4,8
6000	5504	7,3
7000	5819	11,9
8000	5480	17
9000		

3. OPENSTACK

Write	0 KB			10 KB			100 KB			1 MB			10 MB		
Clients / File size	Op/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %
2	67	35	34	35 op/s	29	8	3	30	4	18	49	12	40	29	6
4	115	38	10	1	38	8	6	33	7	23	49	12	76	29	19
8	190	41	14	1	42	11	10	35	11	39	53	12	129	31	22
16	292	43	18	1,9	43	15	16	46	16	68	80	26	207	56	28
32	375	46	26	2,6	47	23	23	49	23	124	87	27	314	62	41
64	467	74	35	3,5	54	30	33	55	23	162	85	25	429	87	74
128	528	87	44	3,9	60	37	38	86	23	202	84	47	521	95	81
256	600	82	50	4,2	60	60	46	76	44	233	92	60	578	96	92

Read	0 KB			10 KB			100 KB			1 MB			10 MB		
Clients / File size	Op/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %
2	253	32	6	1,5	32	7	12	30	4	25	48	6	80	22	6
4	477	35	13	3,4	33	13	26	30	10	51	48	11	164	24	14
8	850	35	23	6,4	40	19	38	33	17	106	51	17	316	29	30
16	1703	38	42	14	42	47	66	38	35	212	62	38	565	35	55
32	2550	47	75	23	45	65	118	41	53	435	65	70	849	38	78
64	3332	70	28	24	39	30	215	53	82	589	67	90	901	48	93
128	4000	42	35	25	62	23	248	64	98	598	69	80	911	51	98
256	4500	43	95	30	45	40	267	69	99	600	86	100	953	55	99

Delete	1 MB		
Clients / File size	Op/s	S CPU %	P CPU %
2	27	51	3
4	54	52	3
8	103	54	6
16	170	56	15
32	280	56	17
64	347	65	22
128	450	87	22
256	498	89	28

4. GLUSTER

Write	0 KB			10 KB			100 KB			1 MB			10 MB		
Clients / File size	Op/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %
2	65	7	12	1	6	17	6	6	9	34	6	9	63	5	10
4	120	12	17	1	10	17	11	11	18	64	13	19	130	12	20
8	233	19	28	2	44	44	17	18	35	107	18	42	217	24	45
16	323	31	42	3	30	49	21	19	46	144	22	44	304	30	52
32	377	35	39	3	40	42	24	24	38	168	24	48	387	42	56
64	330	35	38	3	30	29	22	34	32	179	24	50	434	43	59
128	290	40	33	3	32	29	20	22	34	186	35	51	458	41	69
256	241	45	33	3	35	37	22	37	39	164	31	44	462	41	67

Read	0 KB			10 KB			100 KB			1 MB			10 MB		
Clients / File size	Op/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %	MB/s	S CPU %	P CPU %
2	548	8	9	3	5	27	23	4	10	38	6	11	30	6	9
4	1151	14	20	6	9	11	44	8	11	45	8	18	53	12	20
8	1835	25	40	11	20	24	72	15	49	79	12	43	92	14	38
16	1945	36		17	33	62	126	24	68	129	24	58	143	18	46
32	2059	43	34	18	39	69	158	32	72	197	28	71	192	22	53
64	1431	28	21	14	39	68	147	35	80	250	36	77	239	24	55
128	1080	31	27	12	31	60	127	30	83	296	27	78	284	28	64
256	1029	31	26	13	36	59	117	45	62	270	40	80	300	32	66

Delete	1 MB		
Clients / File size	Op/s	S CPU %	P CPU %
2	383	8	9
4	688	12	10
8	1200	23	37
16	1359	34	39
32	1400	35	42
64	970	16	34
128	581	38	41
256	441	25	28

Spec2008sfs		
Req IOPS	IOPS	Resp (ms)
2000		
3000	3008	10,1
4000	4017	11
5000	3922	23,9
6000	3821	25,9
7000	4019	24,6
8000	2977	33,3
9000		