Evaluating Cassandra as a Manager of Large File Sets

Leander Beernaert, Pedro Gomes, Miguel Matos, Ricardo Vilaça, Rui Oliveira

High-Assurance Software Laboratory INESC TEC & Universidade do Minho
Braga, Portugal
{lbb,pedrogomes,miguelmatos,rmvilaca,rco}@di.uminho.pt

Abstract

All companies developing their business on the Web, not only giants like Google or Facebook but also small companies focused on niche markets, face scalability issues in data management. The case study of this paper is the content management systems for classified or commercial advertisements on the Web. The data involved has a very significant growth rate and a read-intensive access pattern with a reduced update rate.

Typically, data is stored in traditional file systems hosted on dedicated servers or Storage Area Network devices due to the generalization and ease of use of file systems. However, this ease in implementation and usage has a disadvantage: the centralized nature of these systems leads to availability, elasticity and scalability problems.

The scenario under study, undemanding in terms of the system's consistency and with a simple interaction model, is suitable to a distributed database, such as Cassandra, conceived precisely to dynamically handle large volumes of data.

In this paper, we analyze the suitability of Cassandra as a substitute for file systems in content management systems. The evaluation, conducted using real data from a production system, shows that when using Cassandra, one can easily get horizontal scalability of storage, redundancy across multiple independent nodes and load distribution imposed by the periodic activities of safeguarding data, while ensuring a comparable performance to that of a file system.

Categories and Subject Descriptors D.4.2 [*Operating Systems*]: Storage Management; C.2.4 [*Computer Communication Network*]: Distributed Systems :Distributed databases; D.4.7 [*Operating Systems*]: Organization and Design : Distributed systems

General Terms Design, Performance, Evaluation

Keywords Non-relational databases, Cassandra, File systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CloudDP '13, April 14, 2013, Prague, Czech Republic.
Copyright © 2013 ACM 978-1-4503-2075-7. . . $15.00

1. Introduction

Companies developing their business on the Web need content management systems for classified or commercial advertisements on the Web, able to handle the sheer volumes of data generated by very large scale Internet services. Moreover, most of these systems require elasticity properties, the capacity to add more storage capacity without disrupting the service.

The discussion about the best persistence method of static content (e.g. image files) is recurrent. Since this is the core function of file systems, they are usually the first and most appropriate choice for the implementation of such service, as its use does not pose difficulties and the user interface is very simple.

Currently, the contents of these systems are easily accessed from HTTP servers taking advantage of calls to the operating system that allow a direct and asynchronous reading of the file to the network device. These solutions are typically placed into production on centralized storage servers such as NAS (Network-Attached Storage) or SAN (Storage Area Network), which are sometimes limited in terms of scalability and system availability. Moreover, most of these systems need to be taken offline in order to increase capacity thus compromising availability.

In order to circumvent the limitations in terms of scalability and system availability of traditional file systems several distributed file systems exist, such as Lustre[1], Ceph [9], and GFS [4]. However, they are designed to handle large files and thus provide poor performance on workloads such as content management systems that access many small files.

One alternative approach could be the use of relational databases. In terms of availability, a database can offer a more consolidated solution and increased performance regarding replication. However, they have limited scalability [7] and even distributed and parallel databases, that have been around for decades [6], build on the same architecture

[1] http://wiki.lustre.org/index.php/Main _Page

and thus have the same scalability limitations. Moreover, the strong consistency model associated with traditional relational databases poses a significant overhead and performance penalty in a scenario where content is mostly static. In hindsight, these are some of the reasons underlying the emergence of non-relational databases such as Cassandra.

In this paper, we evaluate Cassandra [5], for the storage of a large number of files of reduced size whose access pattern is essentially composed of reads.

With multiple physical replicas we can achieve a system capable of horizontally scaling, coping with the increased volume of data, and in some cases of high load, while simultaneously offering a better protection against possible failures of nodes and disk corruptions.

The rest of this paper is organized as follows. In Section 2 we present the related work, and in Section 3 we describe the details of Cassandra relevant to this paper. In Section 4 we present our approach to the management of large file sets on Cassandra. Section 5 presents an evaluation of a real case study, and a comparison of our approach with traditional file systems. Finally, Section 6 concludes the paper.

2. Related work

The persistence of static or seldom updated content is a common need in Web companies. Many companies choose specific file systems for this purpose, which are supported by storage devices with redundancy to ensure the durability of the data. However, such services are limited to the capacity of the server on which they are placed and, as such, an increase of the load beyond its rated capacity is problematic. In this scenario there are two options: scaling vertically and replacing, if possible, the server with a larger one, or scaling horizontally and adding new servers and distributing the system's load.

The horizontal scalability of file systems is still an area of ongoing developments with new solutions emerging in the market. Many distributed file systems have evolved as commercial products and others as internal solutions in companies like Google. In fact, this is the case of GFS[4](Google File System), a system built to hold large volumes of the company's data using current general purpose hardware. This depends on a node coordinator to ensure consistency in some operations, such as a locking and election mechanism in case of failures.

Other systems are also designed for larger scales, but directed to the common user. In Ceph[9], we have mechanisms of replication and fault tolerance that, however, require synchronization among all replicas. Other systems, such as Lustre, assume that each node provides its own redundancy with different persistence schemes on disk.

Unfortunately, all distributed file systems are not optimized for the workloads we target, namely content management systems that access many small files. This stems in part from the need to perform many operations over metadata [4, 9] yielding poor performance under these workloads.

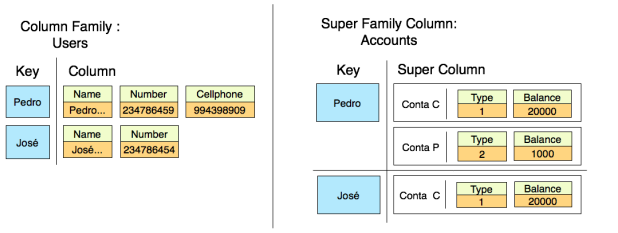


Figure 1. Example of a family of columns and another of super columns.

Moreover, Cloud storage services such as Amazon's S3 service[2], or open-source alikes such as OpenStack's SWIFT[3] are designed for large files and thus have the same performance issues with many small objects.

3. Cassandra

Cassandra is a distributed solution that tries to combine the architecture with no single points of failure of Amazon's Dynamo [3] with the data structure of Google's BigTable [1]. This database, originally created by Facebook, is now used in several companies in the Web due to its recognized ability to reasonably scale up to several hundred nodes. It is chiefly characterized by its relaxed consistency model based on quorums, where different replicas may differ in time, and finally reconcile based on temporal labels that accompany each of the writes [8]. Such errors are immediately detected and corrected, thus preserving integrity.

Cassandra gives the user the flexibility in selecting the level of consistency of reads, providing settings for the number of replicas and their location, when reading and writing data.

Regarding the model, the data is grouped into structures called column families. In these structures, similar to the tables of the relational model, each line has a set of arbitrary columns, thus allowing more dynamism in data modeling. There is also the ability of each column being itself a list of columns called Super column. However, this option is seldom used, because it imposes a penalty database. These structures are exemplified in Figure 1.

4. An alternative on file persistency

Regarding the problem of scalability and integrity of data in common file systems, we now present a new alternative. The goal is not to develop a system that can be fully compared to a file system, whether in its POSIX interface or management of metadata. In fact, when working with rarely updated data, there are some restrictions which can be relaxed, re-

[2] http://aws.amazon.com/s3/

[3] http://swift.openstack.org/

vealing new deployment options. We present a new approach based on the use of a large scale non-relational database which presents advantages regarding scalability and durability. However, in order to be adapted to a traditional environment, we chose to keep the file and directory concepts in the data structure, because it is easier for the administrator to view the content in accordance with these structures.

These systems are mainly updated off-peak during hours with the operation operation performed either by a single operator or an automatic process. Therefore, competing writes are a problem that we can largely ignore. Reading data is the core operation and is characterized by accesses to single or small groups of files. These characteristics fit in Cassandra's consistency model benefiting from its data distribution that, if partitioned based on keys' hash, also avoids contention points.

4.1 System's Architecture

Having Cassandra as the base element, the architecture proposed here is simple in terms of components but requires the creation of new interface components for integration with traditional systems. In fact, this is the biggest disadvantage of this proposal, because we lose the modules and interfaces that are commonly used to communicate with web servers and clients. Apart from its database on Cassandra, the system still depends on a dedicated connector for a web server, and an interface where the client can manipulate the files to be stored. We consider these issues to be out of scope of the current paper and focus only on the core system.

4.2 Model Adaptation

One of the first tasks has the adaptation of the data model to Cassandra. At its base, each file is treated as an array of bytes with a name associated to it as identification. This is the base unit of the system, excluding for now additional metadata. The notion of directory as a container of files or other directories is also fundamental. This hierarchical structure can be used or ignored in the organization of the data model, as only the files themselves are vital to the system. There are thus two options: a) transform the directory structure that the client creates while inserting files into unique names resulting in a model of file per line; or b) incorporate the directory concept in the system, inserting a notion of association between files, thereby, creating a directory structure per line.

File per line: To implement the first option, the file name is concatenated to the directory structure and acts as a primary key in a single column family in Cassandra (Figure 2 a). In each line there is a column with the file content. In this model, each line can also contain posteriorly other information with a column that encodes the metadata of the file in question. Each reading of files is done by name and directory, and there is no direct way to read a group of files associated with each other (usually organized on a directory). Actually, the directory structure can be preserved in a column family aside so that simultaneously reading multiple files common to an application or web page is possible; although, this is not currently an essential requirement to the system.

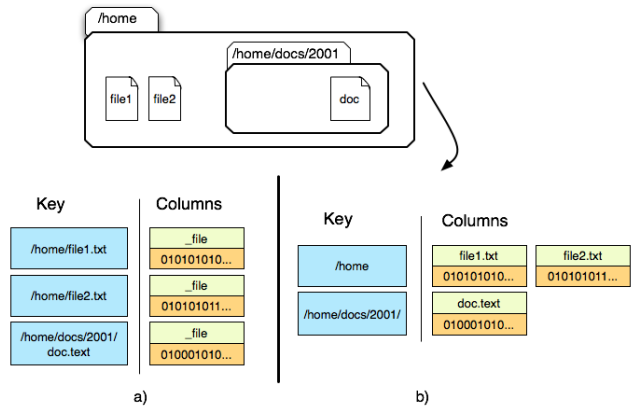


Figure 2. Alternative data models: a) file per line, b) directory per line

Directory per line: In this model each line has as key the name of a directory, or concatenation of the names in the hierarchy forming a path, Figure 2 b). For each line, columns have the value and name of the file. In this option there is some notion of directories and groups of files that allow reading several related files without knowing their name. From the client's point of view, the overall structure of the directories may or may not be represented in the database, because accesses may require the full path of the file or just the containing directory.

These options imply different access patterns and internal representations in the database, thus existing an inherent difficulty of knowing which of these will have a better performance or variability in terms of access time. We evaluate both approaches in the next section.

5. Experimental analysis

This section evaluates our approach to the management of large file sets on Cassandra. We have done some preliminary experiments to confirm whether distributed file systems have performance issues with workloads consisting mostly of many small objects. To this end we tried to populate our case study over GLuster FS[4], Ceph[9] and SWIFT[5] but in all systems the populate has very slow, taking several weeks to finish.

5.1 Case study

Our case study, based on a production system, is a list of files associated with a portal that currently hosts content for various websites of classified or commercial advertisements. It consists of thirteen million images, with size distribution as

[4] http://www.gluster.org

[5] http://swift.openstack.org/

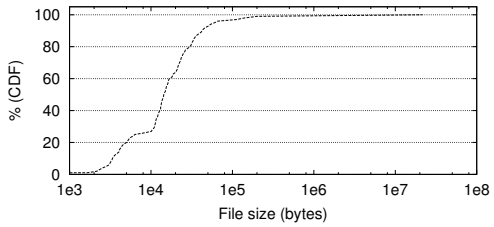


Figure 3. File size distribution.

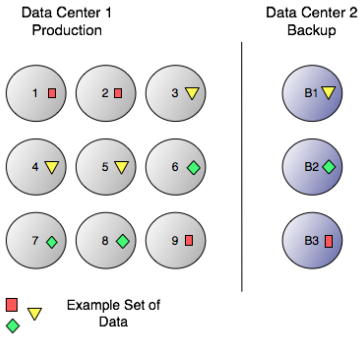


Figure 4. Structural model of the nodes.

depicted in Figure 3 and where 90% of these are smaller than 50 KB, occupying about 300 GB of space. Most images are inserted by the client and written overnight and then served in different websites via requests to the Apache server, which is connected to the file system served by a centralized storage system.

Currently, the total size of the data is not yet a problem in terms of access to the system, as the average latency for a single client to read local files is of 1.8 ms.

However, it is already problematic whenever a full backup is needed. In fact, the reading of all files on disk (without resorting to *cache*) or the simple traversal of the directory structure, is not only slow, taking several hours, but also affects the performance of the system during the process. This problem makes it impossible to carry out differential backups, forcing the creation of backup strategies defined together with the application. This problem together with the single point of failure of a centralized storage, is a scenario suited to the solution we propose in this paper.

In terms of access patterns, there are two concurrent operations. The first are regular read operations that follow a power-law distribution. The second are sequential traverses that read all the data which model backup operations. Following the behavior of a real system, we don't want the system to be taken offline (don't allow regular read operations) to realize backup operations.

5.2 Testing Platform

To test our implementation, we use Yahoo!'s YCSB[2] that is designed to test large scale non-relational databases. It is an adaptable platform whose value is recognized both within academic and business communities to test new solutions in the area of databases.

Nevertheless, the platform has some limitations to our case study. In fact, YCSB is prepared for reading and collecting small amounts of information that are uniform in size, and whose keys are integers so they are easy to generate during execution. Yet, this is a mechanism that does not conform to the scenario in question, and it is, therefore, necessary to create a new request client.

This client can make requests by file name or directory thus allowing to test the different models of data organization described in Section 4.2. Another important factor for the creation of this new component is the fact that we can inject into the platform sets of pre-existing keys which we previously inserted in the database. In fact, in databases like Cassandra the dynamic nature of the data model, and more specifically the naming of the keys and columns represents a real cost in terms of storage and performance.

Further modifications to the platform adapter for Cassandra were required due to the limited configuration options of operations performed in Cassandra, thus inserting a more dynamic control of the consistency levels to be used. Accordingly, it is now easy to define what number of nodes are involved in each read in the database. Similarly, we assigned node categories to each type of reads. An example of the usage of this option is the routing of sequential reads of data to nodes isolated from the rest of the system. All modifications done to YCSB and to the Cassandra adapter are available at `https://github.com/PedroGomes/YCSB`.

In short, we use YCSB for the execution of requests and collection of obtained results, receiving as input different Cassandra settings and types of operations used. The platform also makes use of an external in-memory database (Redis [6]) with the names of all files to be accessed. Thus, this avoids reading millions of names through the platform on each execution with minimum cost in generating requests.

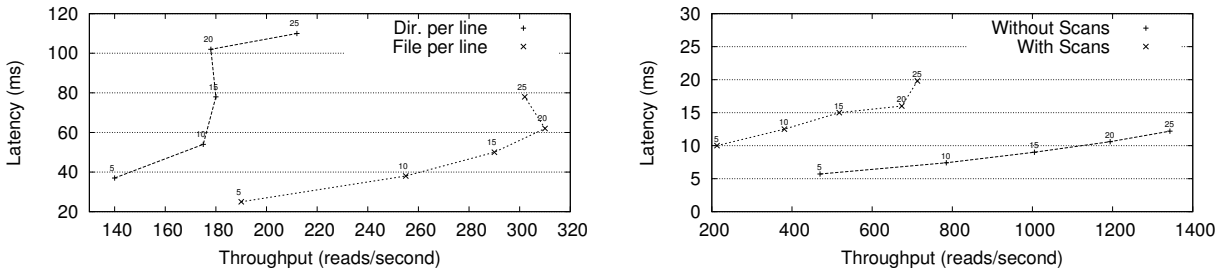
5.3 Experimental setting

In terms of hardware, all nodes used are commodity machines with an Intel dual-core i3 at 3.1GHz, 4GB of memory and a storage capacity of 250 GB. All machines are connected by a local network of 1Gb/s.

As our workload is composed of unique and sequential read operations (for backup), we divided 12 Cassandra nodes into two groups that are configured as two different data centers as depicted in Figure 4.

From Cassandra's perspective these two groups are two distinct physical locations, allowing to perform, in isolation, different operations on each group of nodes. In the first of these groups, which would be the data center to serve most requests, there are nine nodes configured with a replication factor of 3. In the second group, which works as a backup,

[6] http://redis.io/



(a) Cassandra deployment

(b) File system deployment

Figure 5. Cassandra and filesytem comparison. Results for Cassandra include the scan operation. The numbers close to the line points are the number of clients.

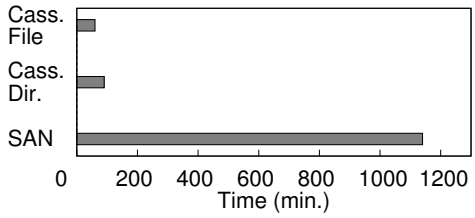


Figure 6. Full traversal/backup performance.

there are 3 nodes that replicate data that exist in the other data center. In this scheme, the data of node 3 is replicated in nodes 4 and 5, and in node $B1$ of the auxiliary data center, as can be observed in Figure 4.

Thus, such structure provides a degree of redundancy which allows the system to remain online even if a portion of the nodes fails. Other advantages of this scheme is that the replicas in the second data center can function as a data exploration site or backup without interfering with the normal operation of the system. Moreover, since all writes will ideally go to both places, there is no problem in serving requests from this site at times of great load on the system. In short, in its typical operation, each direct read of the data will always be made in a node quorum in the main data center, and the read requests for backups directed to the alternate data center. As for writes, a level of consistency that guarantees writings in node quorums of both sides is enough (in which case the quorum in the auxiliary data center is only a node).

In order to compare our proposal based on Cassandra with a typical environment, as the one used in production for the case study, we setup a RAID-0 volume with 350GB on a HP storage Works Eva 4400, and formatted it with the ReiserFS file system.

5.4 Models analysis

In this section, we compare the performance of the two alternative data models proposed in Section 4.2 with the performance of a traditional file system deployment, using all the data from the case study for unique and sequential read operations (for backup).

Results for our proposal for regular read operations are depicted in Figure 5(a). As it is possible to observe, the file per line model has lower latency and higher throughput as the number of clients increases and thus is preferable to the directory per line model. Despite Cassandra's single row read operation allowinging to filter which column to retrieve (for the directory per line model, only fetch the column with the requested file), it has additional overhead that adds up to latency and thus decreases throughput.

For sequential traversal operations, the requests are run in parallel to each of the replicas that are at the backup data center, thus not interfering with regular reads. For these operations, the directory per line model is faster, taking on average one hour to complete, while the file per line model takes about an hour and a half (Figure 6). This can be explained by the number of files per directory that exist in each line. In fact, the number of directories in the system is 8 thousand contrasting with the 13 million files, and the number of files per directory is highly variable. Thus, this difference affects the reads of individual columns, but according to the results, the lowest number of lines helps the sequential reading of the data.

Considering that the backup operation is not frequent, the time for backup the file per line model is still acceptable and that the overhead of the files per directory model for regular read operations is not huge, the file per line is the most appropriate model for this workload.

Regarding the comparison with the traditional file system deployment, as can be observed in Figure 5(b), for regular read operations the file system deployment has lower latency accompanied by an higher read throughput. However, the time for sequential traversal operations is unbearable, taking on average nineteen hours to complete (Figure 6). Moreover, we can see that when a sequential traversal operation is running concurrently to regular read operations, their latency almost doubles. This confirms the observations in the production system.

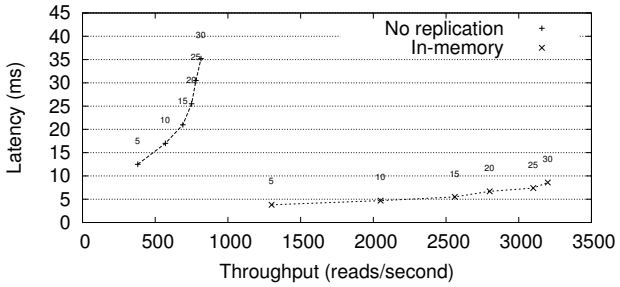


Figure 7. Scalability results

5.5 Scalability

Based on the results obtained obtained above, we decided to continue with the model of file per line and explore Cassandra's scalability in this scenario. For this purpose, and in the absence of a larger number of machines, we used 10 of the nodes previously used to recreate the same scenario but without replication. Analyzing this transformation, from the 300 GB of existing data, each node would always be responsible for one third of the data in the above scheme (i.e. approximately 100 GB), contrasting with this new scenario where each node is only responsible for 30 GB (i.e. less than one third of the original load). Compared to the original results, we can see from Figure 7 that latency has fallen by half while the throughput of read operations rose considerably. Actually, considering the data distribution and replication factors in both scenarios each machine effectively serves on average about 30 GB of data. This difference in performance is justified by the fact that, containing much less data, the cache of these nodes becomes more efficient, especially if we take into account that the reads are executed according to a power-law distribution.

Then we inquired what is the real performance of the system when using machines with a more favorable ratio between RAM and data volume. This is a scenario in which much of the data can be in memory. For this, and given the constraints regarding the available hardware, we chose to reduce the number of persisted data so that each machine does not hold more data than what it can fit in memory. As can be observed in Figure 7, there is a substantial decrease in latency accompanied by an increase in the read throughput. In this setting, the latency of our proposal is slightly lower of traditional file systems, Figure 5(b), and the throughput is much higher.

6. Conclusion

This paper presents a new approach to content management systems for classified or commercial advertisements on the Web, containing large amounts of seldom updated small files.

Applied to a real scenario, we noted that our proposal has comparable performance with a common file system with clear advantages in scalability, durability and data availability, allowing the system to work regularly while performing maintenance tasks such as backups. Complementing the tools that for this purpose are already offered by Cassandra, we show that a global reading mechanism of data implemented to run separately on selected nodes can also be used. In fact, being a system with redundancy in multiple data centers, one of these sites can be regarded as a functional backup of the other. This allows the client to reduce the number of times this operation is performed and, as a consequence, reduce its cost.

Acknowledgments

This work is in part-funded by; ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project Stratus/FCOMP-01-0124-FEDER-015020; and European Union Seventh Framework Programme (FP7) under grant agreement n^o 257993.

References

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI'06*, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *SOCC '10*, pages 143–154, New York, NY, USA, 2010. ACM. doi: 10.1145/1807128.1807152.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07*, pages 205–220, New York, NY, USA, 2007. ACM. doi: http://doi.acm.org/10.1145/1294261.1294281.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003. doi: http://doi.acm.org/10.1145/1165389.945450.

[5] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, 2010. doi: http://doi.acm.org/10.1145/1773912.1773922.

[6] M. T. Özsu and P. Valduriez. *Principles of distributed database systems (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[7] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB'07*, pages 1150–1160, 2007.

[8] W. Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, Jan. 2009. doi: 10.1145/1435417.1435432.

[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *OSDI'06*, pages 307–320, 2006.