File Systems: GFS vs HDFS vs Ceph Bluestore

Nicolas Arteaga and Raphael Schleithoff

Summer term 2020

Abstract

We describe three different distributed file systems, the Google File System, the Hadoop Distributed File System and Ceph Bluestore, that strive to handle the growing amounts of data that is produced by modern algorithms. We will have a look at the basic operations and the different trade offs they implement in their system architecture. Finally we compare all three and recommend one of the distributed file systems.

1 Introduction

Computer Science is all about abstractions, which means that complexity of an implementation is hidden and that new ideas are built on top of already existing structure. Compilers or specifically in distributed systems the OSI model provide levels of abstraction on top of which new protocols are created. etc. Distributed file systems act in a similar way, they want to abstract where data is stored exactly and only want to make it accessible to the client through an API.

In this paper we want to have a look at the way a set of distributed servers are orchestrated and networked to emulate a single file system.

Without file systems data inside a storage medium would be one large blob without any way of separating or retrieving them. A file system therefore defines the structure and logic for managing data (files).

When talking about file systems it's important to keep the context in mind. In the context of disks a file system is the way in which files are actually organized in the physical storage medium. Formats include ext2, FAT. In the context of operation systems a file system usually means the hierarchical directory structure. Here the UNIX file system is a great example. Distributed file systems are close to the latter kind. They provide interface for clients to interact with a file system as if it were stored on a single computer. But by making it distributed a lot of functionality and performance improvements.

2 The Google File System

The Google File System (GFS) was developed out of the need to process large amounts of data for the search engine. It is Google's proprietary software, therefore there is only a white paper about its functionality and not any code to look at. Google's search algorithms functions in a highly distributed manner but all the programs running on differ-

ent machines have to access the same data to work on. The many petabytes of data can't be stored on one huge server because one server can't provide fault tolerance or high and easily scalable storage capacity.

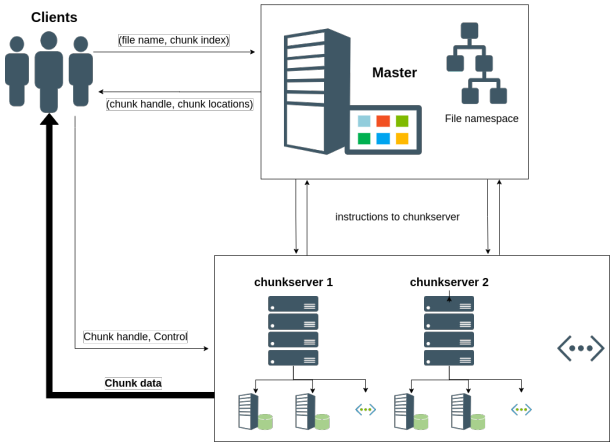

Figure 1: GFS Architecture Model

The GFS was designed to be optimized for reading and appending, which means that writing/overwriting is possible but comparatively costly. In contrast to what one might think Google doesn't operate high-end servers with triple redundancy, but cheap commodity servers running Linux. This enables easy **scalability**, low costs but also leads to higher probability of any one of the servers crashing. GFS was therefore designed with high **fault tolerance** as a central characteristics.

From the user point of view we have a classic client-server architecture. A client, some program running on a search query for example, makes a request to the GFS and the GFS hands it the contents of that file. If we dive a little deeper we see that the GFS is made up of clusters of servers operating in a master-slave architecture. In the GFS every file is split into chunks of 64MB, which are assigned unique 64-bit chunk handles and then replicated and distributed to the chunkservers by a master server. So the file example.txt with a size of 130MB is split into two chunks of 64MB and one chunk of 2MB, which are then replicated three times across the cluster of chunkservers, resulting in a total of nine chunks. This may seem overly redundant but replication not only increases the fault tolerance but also the bandwidth of the overall system by making the same chunks accessible for reading in parallel.

On a high level view the client sends a request to the master server wanting to access /data/example.txt with a byte offset if need be. The master server, knowing how that file is distributed amongst the chunkservers, tells the client on which chunkservers to find file. The client then makes a request to the chunkservers, which send the data directly to the client without going through the master server. This ensures that the master server doesn't become the bottleneck of the system. If many clients send requests to the master server this could of course still lead to higher latency but by letting the data transfer happen directly between the client and the chunkservers overall bandwidth is increased, which is far more important metric for the file system than the latency. Message size is kept small.

Autonomic computing is another key concept important in the design of the GFS; it strives to automate administrative duties that keep the system running. Examples of this include detection of dead/crashed chunkservers through heartbeat logging and the maintenance of the operation logs (mapping of chunks to file). In order to serve clients as fast as possible the master server keeps the whole op-

eration log in RAM, which means that if the master server crashes the whole log is lost. In order to solve that, a second (shadow) master server continuously copies the master slaves operation logs.

3 Hadoop Distributed File System

If one were to assign some kind of genealogy to distributed file system then the Hadoop Distributed File System (HDFS) would be the open-source clone of the GFS. The HDFS, licensed under the Apache License 2.0, forms part of the Apache Hadoop project and can therefore be used by us to demonstrate how a distributed file system works in practice. HDFS provids an efficient data store and retrieval for Big Data applications such as MapReduce.

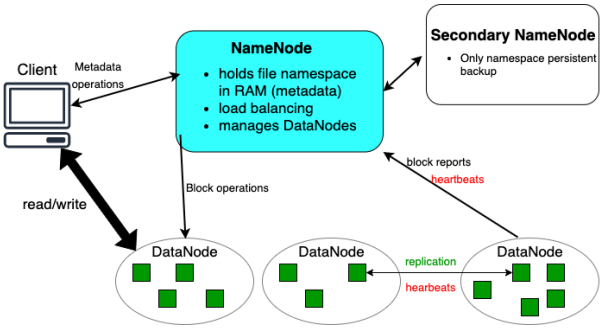

Figure 2: HDFS System Architecture

Even though the HDFS is heavily modelled after the GFS, there exist some differences. The master server becomes the NameNode and the chunkservers are called DataNodes. Furthermore files are split into blocks of 128MB, instead of 64Mb chunks.

HDFS implements a centralized architecture, where namespace information (block location of file) is communicated by the **NameNode** but actual file data is transferred between the client and the **DataNode**. This makes **load balancing** possible but can also lead to lower ***availability** of the system.

In order to increase the **availability** of the system the NameNode holds all the namespace information of the file system in RAM. This makes the system less faul tolerant because if the NameNode crashes it losed that information. The **SecondaryNameNode** therefore periodically saves it in its memory to make it persistent.

Fault detection and node statuses are propagated through the system with **heartbeats** and **block reports**. By default every three seconds a heartbeats and every 10 minutes a block report from each DataNode. This makes the system fully connected, which means that every node in the file system is aware of faulty behavior.

4 Ceph Bluestore

Ceph Bluestore is an open source totally distributed file system. The philosophy from Ceph is an Open Source Project for evolving in the needs of the future, that focuses on the community and has no single point of failure. Meaning that a Failure from a server is the norm and not the exception. It's Design concentrates on being scalable, that's why is totally distributed and has not a Master server. It's software based, because all kinds of different hardware can run with the same software and work together seamlessly. Another important aspect from Ceph is, that it's Self-

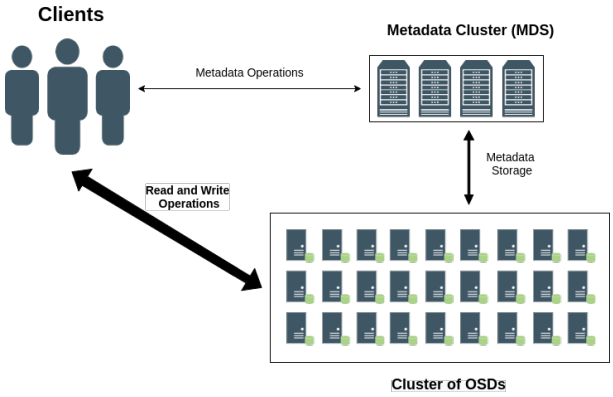

Figure 3: Ceph Architecture Model

Managing, meaning that it heals automatically whenever a problem arises.

Ceph doesn't use a Master Slave Architecture, so to substitute a Master it provides a dynamic distributed metadata management using a metadata cluster (in short MDS). Then stores data and metadata in Object Storage Devices (OSDs). These are simply a CPU attached to a Network Card and to an SSD or Harddisk. OSDs are used to take advantage from horizontal scalability (Having more computers, rather than one super computer). The MDS manage the namespace, consistency and security of the systems and Clients access the data directly using an algorithm named CRUSH, to find the Blockfiles. [Figure 3]

4.1 CRUSH

The algorithm CRUSH stands for Controlled Replication Under Scalable Hashing and is a pseudo-random placement function, that determines which OSDs to store files on, instead of the centralized way of storing the placement information in a table, like with the GFS. The volume of metadata stored per file is reduced, boosting metadata accesses and shrinking the load on metadata servers. Clients directly calculate the proper file placement for read and write operations, and only need to connect with the MDS servers for metadata operations.

4.2 Differences from Ceph Filestore

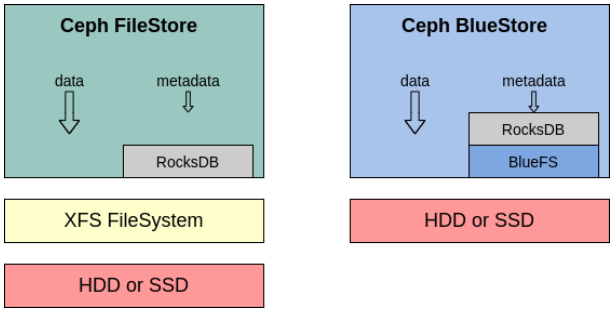

Figure 4: Ceph FileStore vs Ceph BlueStore

The Motivation from Bluestore is improving the performance of the cluster. The previous object store, FileStore, uses the journaling File System XFS on top of raw block devices. Instead, BlueStore stores objects directly on the block devices. And since the metadata in Ceph is organized with RocksDB, a Database that cannot write directly to the raw disk device. BlueFS was created, a file system only for RocksDB with the minimal functioning feature set, to store the metadata.

The results make BlueStore roughly about twice as fast as FileStore, leaving a boosted performance (almost 2x for writes) with a lower latency.

5 Conclusion

As an overview, we summarized the most important points from the three File Systems

	HDFS	Ceph
Architecture	Centralized	Distributed
Namespace	Index	CRUSH
API	CLI, REST API	CLI, REST API
Fault detection	Fully connected	Fully connected
Replication	Async.	Sync.
Load balancing	Automatic	Manual

Table 1: Analysis of Six Distributed File Systems (GFS is almost the same as HDFS)

described in this Report. In a first examine it's easy to spot the similarities between the Google File System and the Hadoop File System, since HDFS is based from the other. That said, as a conclusion we will only concentrate in HDFS vs Ceph. The Architecture from HDFS is centralized, using an Index Namespace in the Master to map all files in the servers, while Ceph is completely distributed and uses it's CRUSH algorithm to be able to find all Blockfiles. Both use a Command Line Interface and an Application Programming Interface, but since GFS is a white paper, there is no info what Google uses to control it's File System. All three File Systems provide strong fault detection, since they use heartbeats to check between the Master and the Server or the individual OSDs. More significant differences come with the Replication and Load Balancing. HDFS is asynchronous, this being, that when a File is written, it is still possible to request it, although the replicas aren't written in the disks. This changes with the synchronous replication form Ceph where, when data is written, the data is locked until the replicas are committed to the disk. In HDFS there is an automatic Load Balancing between the Datanodes, but in Ceph, because of it's pseudo random placement of the files by CRUSH, the Load Balancing commands have to be run manually.

	HDFS		Ceph	
Write / Read	**R**	**W**	**R**	**W**
1 x 20GB	401s	**407s**	**342s**	419s
100 x 1MB	17s	72s	21s	76s

This table compares the performance of HDFS and Ceph in Read and Write operations from one big 20 GB File and a hundred 1MB Files. And reflects, that HDFS is more applicable to store small quantity of big files (Writes faster), while Ceph can dominate both small and big data.

5.1 Our Recommendation

We would recommend using CEPH because of the performance improvements. This being that decentralized architectures seem to scale better than a centralized one thanks to the distributed workload management. That and the fact that Ceph is an Open Source Project, makes it the future of storage.

6 Sources

Ghemawat, Sanjay, Gobioff, Howard and Leung, Shun-Tak. "The Google File System." Google. 2003. Harris, Robin. "Google File System Evaluation." StorageMojo. June 13, 2006.

http://storagemojo.com/?page$_id = 15$

"HDFS Architecture Guide." Hadoop, hadoop.apache.org/docs/r1.2.1/hdfs$_design.html$.

Big-Data-Europe. "Big-Data-Europe/Docker-Hadoop." GitHub, 19 May 2020, github.com/big-data-europe/docker-hadoop.

"Apache Hadoop." Wikipedia, Wikimedia Foundation, 10 May 2020, en.wikipedia.org/wiki/Apache$_Hadoop$.

"List of File Systems." Wikipedia, Wikimedia Foundation, 9 Apr. 2020, en.wikipedia.org/wiki/List$_of_file_systemsDistributed_file_systems$.

Ceph Documentation https://docs.ceph.com/docs/master/

https://www.usenix.org/legacy/event/osdi06/tech/full$_papers/weil/weil_html/index.html$

Benjamin Depardon, Gaël Le Mahec, Cyril Séguin. Analysis of Six Distributed File Systems. [Research Report] 2013, pp.44. ffhal-00789086

https://ceph.io/community/new-luminous-bluestore/