GFS Overview, Analysis and Challenges

UIUC | CS 410: Text Information Systems | Fall 2022 Abhinav Abhay(aabhay3)

Intro:

This topic is of interest in the Text Information System class because distributed storage and computation is one of the key factors or enablers for scalable and efficient Web Indexing leading to a fast and reliable web search engine. The two most important requirement of a Web scale indexing is (a) Storing the index on multiple machines (Example framework, Google File System - GFS) and (b) Creating the index in parallel (Map reduce). It is important to note both are general infrastructure. However, in this paper I will scope the discussion only to GFS.

In this article I will try to present the challenges as to why distributed storage is hard and how Google File System tackle these problems. I will also spend some time to write about challenges or bottlenecks of Google File System. In the end I will provide a brief description on the Hadoop Distributed File System and what is a key difference between GFS and HDFS.

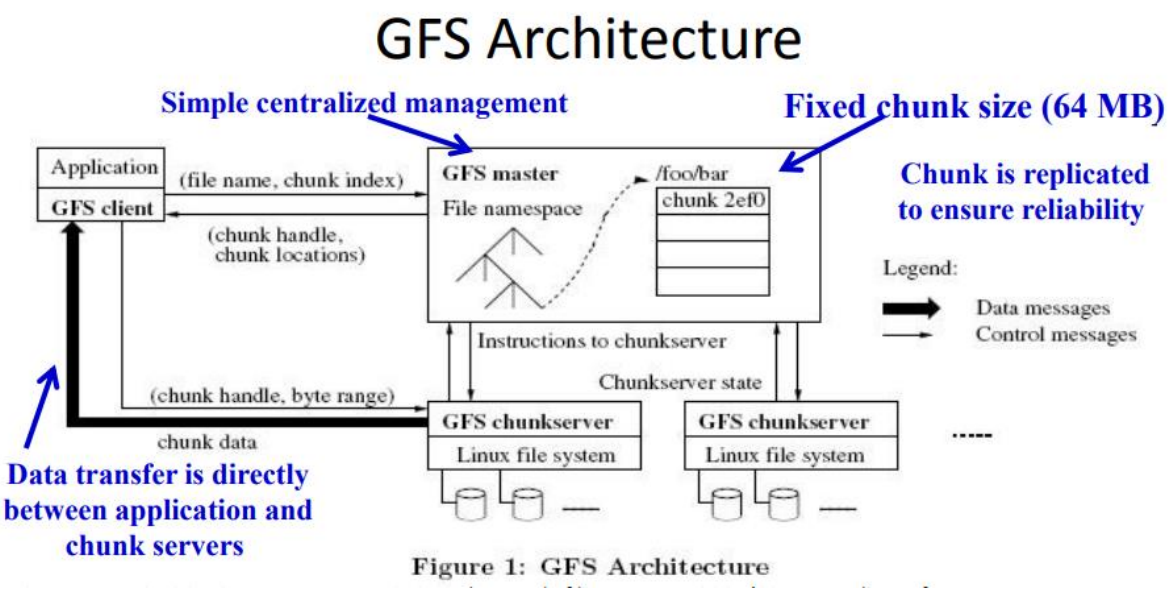
Body:

What aspects makes distributed storage hard?

- Need of high performance as data is supposed to be shared over many servers.
- Many servers give rise to constant faults.
- Fault tolerance introduces the need of replication.
- Replication leads to problems of inconsistencies.

Many google services needed a big, fast, and unified storage system and hence the inception of such a framework was important to google. Let's spend some time to see how Google File System delas with some of these challenges in this framework. I will take a different method to explain the components and workings of google file system , through the prism of each challenges described above.

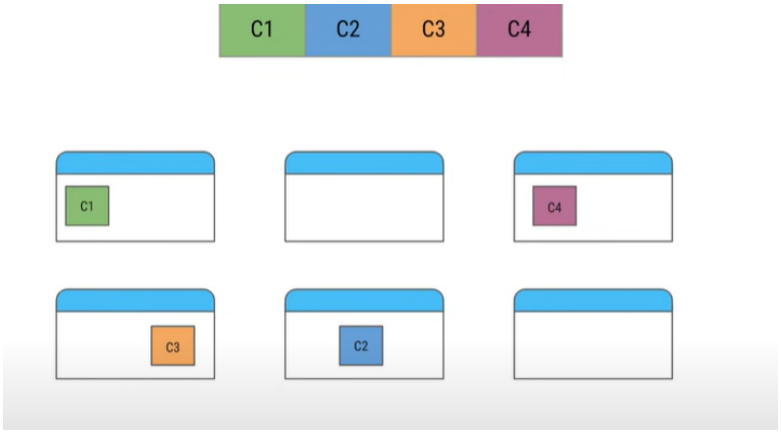
The GFS architecture picture:



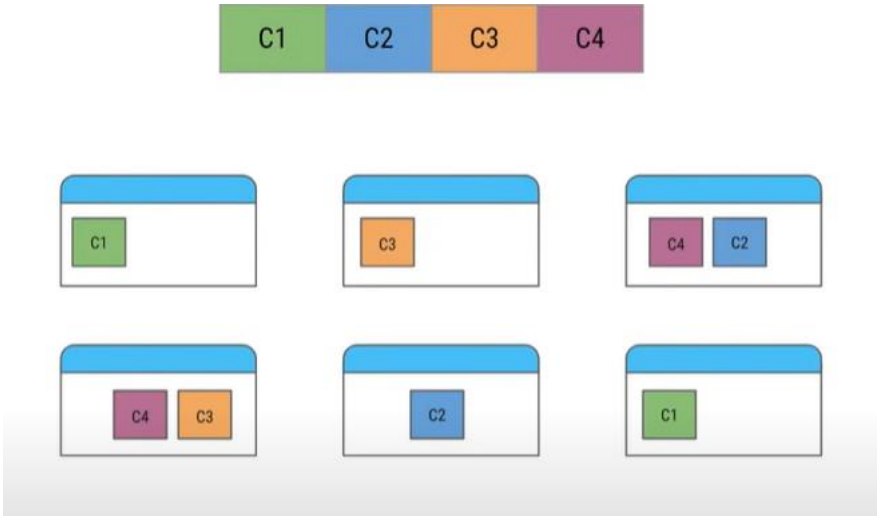
The three abstract components of GFS are:

GFS Client:
The Client is an entity which is used to make request in the GFS. The request can be a read or write to the files in the system. The client entity is either being a computer outside the system or might be just another application program in the same system. We can safely assume that in GFS system a client acts like a customer.

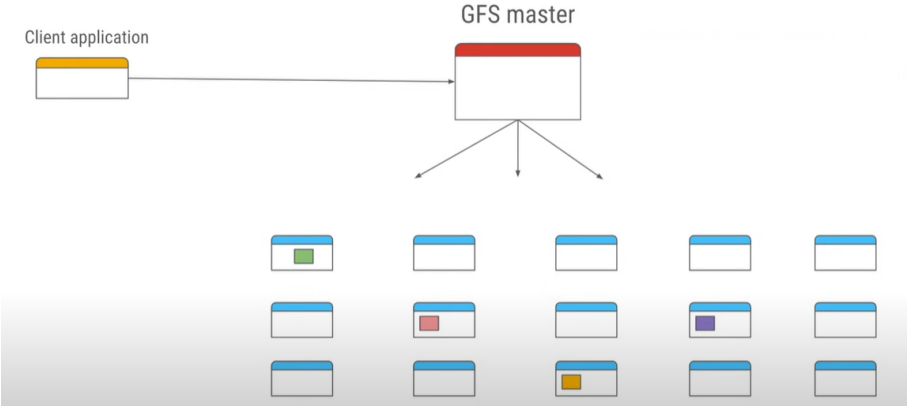
GFS Chunk Server:
A typical file size ranges from 100 MB to few GBs. The idea is that a single file is not stored on a single server but actually subdivided into multiple chunks and each chunk is of 64 MB. These chunks are spread across multiple machines and these machines are part of one cluster. All these machines within this one cluster are called chunk servers. These chunk servers are not storing an entire file but chunks of a particular file and these chunks are identified by globally unique 64 bit ID. The picture below shows file has four chunks and each of those four chunks reside on four different servers.



Many Servers giving rise to constant faults, hence, need of Replication - The servers can have fault or can go down any time, so if the chunk of files have only one copy, there are high chances that you may lose the files forever. Thus, Google file system ensures that each chunk of your file has at least three replicas across three different servers so that even if one server goes down you still have other two replicas to work with and these replicas count by default is three but is configurable. This is how GFS ensure durability of data if chunk servers go down.



GFS Master Server:



The Google file system Master Server is the critical component which contains all the metadata about the cluster. Let's look at the these metadata – It contains a table which has name of all the files in the cluster, how many chunks each file is divided into and their 64 bit ID, count of replicas of each chunk and their respective location.

Read/Write Requests: When a client requests for a particular file, the Master Server returns the ID of the respective chunk, along with the IP address of the chunk server using a chunk handle. If the request is for a write operation the Master creates the file and returns the address of the chunk on which the file is to

be written. Once the client has that information, it directly accesses the chunk server and establishes a connection. It is important to note here that GFS Master only reads the metadata about the request, the actual exchange or connection is established directly between the Client and the Chunk Server. This helps to reduce the load on the Master Server. Caching further helps to reduce the load on Master Server, as the Client chunk handle store some of the frequently accessed information. The Master Server has the intelligence to pass on those servers for write operations which has relatively more space than other chunk servers. When the Client receives these chunk addresses (3 of them, to include the replication), the Client pass on the data to the nearest servers and then it is the responsibility of that servers to pass the information to other two servers. So, the Client's responsibility is to pass the information to one server and that's how the bandwidth is utilized.

Fault Tolerance Mechanism: Since there are thousands of servers, it is possible that any server can go down. The GFS has a mechanism of heartbeat messages. The chunk servers constantly sends a heartbeat message to the Master Server, so that Master knows that the chunk server is alive. If the heartbeat message is stopped from a chunk server, the master immediately creates a replica of it. While a particular chunk server was down, the other two replicas of that chunk was able to take care of the requests. At any point in time the Master always ensures to have 3 replicas of each chunk server.

Operation log: It is interesting to note that there is only one Master Server in the cluster. So, what would happen if Master Server were to go down? Well, there is an operation log stored in a remote machine which stores operations that occur on the files in an append-only manner. Each file operation with the corresponding timestamp and the user details who performed that operation is stored in this operations log and this log is very important and thus is directly written into an input disk as well as replicated to a remote machine and only then the acknowledgement is sent to the client. If the master crashes, it can come back up to its current state by reading the operation log. It is possible that operations log becomes too long and that is why behind the scenes there is a thread which keeps on compressing the log after every hundred lines/transaction. This would keep the log small and if the master were to go down, it does not have to read through all the lines but the compressed format to come back to its current state.

This is the basic overview of Google file system and how it is used to store huge amount of data which is difficult to achieve with vertical scaling and because now these files are distributed across machines it also helps in doing batch processing much faster than a traditional server and that batch processing system is called MapReduce which is out of scope for this article.

Conclusion

There are few thoughts about what few challenges or bottleneck around the Google file system can be. Some of them I understand are

- GFS is an application-oriented file system. How will it work with another application, is not known.
- Implications of having only one Master Server.
- Small file stored only one chunk server; it may become hot spot due to multiple accesses.
- I believe 90% of the request is read only to the GFS, just the nature of the application it supports. How will GFS scale when the requests become write heavy.

In brief, we will see another Distributed File System called Hadoop File System. Yahoo! made an open-source version of Google file system named as Hadoop File System - HDFS. Unlike GFS, HDFS don't provide the appending function. As both GFS and HDFS depends on a single master node which at times proves to be a failure point. Different variants of HDFS were introduced namely RFS (Ring File System) and EDFS (Efficient Distributed File System). Just like GFS the dataset is divided into blocks with TCP used as a Protocol for both message passing and data transferring. As in GFS replicas are created at the time of creation. HDFS is almost like GFS apart from the appending function.

References

[1] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung (2003). The Google File System. http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf

[2] GFS: Evolution on Fast-forward (2009) https://queue.acm.org/detail.cfm?id=1594206

[3] https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system

[4] https://levelup.gitconnected.com/the-google-file-system-the-distributed-file-system-that-google-built-68463670ac57