Systems for Data Science - HW2

The Google File System - Paper Review

Shubham Shetty

February 13 2021

1 Summary

Google File System (GFS) is a distributed file system designed in the early 2000s to deal with Google's ever increasing data volume. GFS aimed to provide high performant and fault tolerant data processing capability, trading off on consistency. The successful implementation of GFS helped scale out Google and handle the increasing size of the web, and also formed the foundation for other distributed data processing systems such as MapReduce and Hadoop.

One of the main issues in designing distributed big data processing systems is that it is difficult to improve performance without losing out on availability or consistency. According to the CAP theorem, it is "impossible for a distributed data store to simultaneously guarantee more than two of the following - consistency, availability and partition tolerance". GFS aims to provide high availability and fault tolerance by design, at the expense of consistency. The main goals of GFS is to handle big data storage and processing in a fast and efficient manner, with a global API, data sharding, and automatic fault recovery, all using distributed architecture.

While designing GFS, a number of assumptions were made as guidelines -
- System is distributed over several inexpensive commodity components. These components are unreliable and may fail, hence the system would require some form of self-diagnosis to ensure fault tolerance and reduce need for manual intervention.
- The file system would mainly store files of large size. Small files would be supported but would not be optimized for.
- Reads would be of 2 types- large streaming reads or small random reads. Writes on the other hand would be large sequential append writes.
- The system would be built for Google's internal use and would require a well defined set of semantics for ease of use for multiple clients.
- Having high bandwidth would be preferred over having low latency.

The three main components of the GFS architecture are -
- *Master Server*: GFS consists of single master node. Master node contains all the file system metadata (for example - file name, chunk handle, logs, checkpoints, etc.). It also controls several other system wide functions and regularly communicates with chunk servers using a broadcast *heartbeat* message.
- *Clients*: Clients are the applications which are reading from or writing to GFS via API calls.
- *Chunk Servers*: GFS consists of multiple chunk servers. Files in GFS are divided into multiple chunks which are then stored across the chunk servers. Each chunk is identified by a globally unique *chunk handle*, whose details are stored in the master node.

As GFS is built on commodity hardware, node failure is a common issue. To ensure fault tolerance, GFS uses replication to ensure that no data is lost due to component failure. It ensures that at least three replicas of the file exist in different chunk servers, while the master node takes care of synchronisation and versioning issues. It also uses fast recovery to improve fault tolerance.

There are two main user operations supported by GFS - file reads and writes. For file reads, the client would first query the master server for the file name. The master server returns the chunk handle containing the requested data. Chunk handle would contain details such as list of chunk servers where the data is present (including replicas), version number, primary chunk server, lease expiration time, etc. Using the chunk handle, the client can directly communicate with the chunk server to read the required data.

File writes can be new files or sequential appends to existing files. To write a new file, client first pings the master server which returns locations of chunk servers which are relatively free. Client can then find the closest chunk server and write all data to that server. Following this the chunk server can then create replicas in other close-by servers. Once all the replicas are copied, the primary server requests commit after which the data is committed to memory from cache, after which the client is sent an 'ACK' message and corresponding chunk handle is updated in the master server.

2 Strengths

The following are the main advantages of using the Google File System -

- Provides a highly fault tolerant system despite running on inexpensive commodity hardware by constant monitoring, data redundancy, and automatic recovery.
- GFS is highly scalable and supports a large number of concurrent reads and writes from multiple clients by providing high aggregate throughput.
- Checksumming is used to detect data corruption at the disk level, improving the system's fault tolerance.
- GFS has demonstrated to be a practical implementation of a distributed system and has been deployed on a large scale producton environment.

3 Unclear Aspects

The following aspects remain unclear with respect to GFS -

- How to detect duplicate requests sent by client in case a server fails during a data read/write.
- How to make the secondary data more aware, i.e. allowing them to understand when there are system faults.

- How to ensure that all secondary data in the chunk servers are synchronized before being updated in the primary and master nodes.
- In case the primary copy of data crashes, how can the master node ensure that the new primary server is correctly synchronised.

4 Limitations

The following are the limitations and areas of improvement for GFS -

- The case remains that GFS does not have strong consistency. For use cases where strong data consistency is a must, GFS would not be an ideal match.
- GFS is highly dependent on the master server. Hence the master server can become a single point of failure.
- The master node has limited amount of memory. As the amount of data stored across chunk servers increases, this memory may run out.
- A single master node may not be able to handle high loads as the number of clients increases.