Google File System

Imran Bohari, Prasad Chitnis
CSCI 652, Distributed Systems
May 14, 2016
Minseok Kwon.
email: iib8085@rit.edu, pgc5277@rit.edu

Abstract

We have implemented Google File System (GFS), a scalable distributed file system for large distributed data-intensive applications. It runs on commodity hardware and is fault tolerant. It is highly efficient as it can serve larger number of clients as compared to other distributed file systems [3].

In GFS component failure is the norm rather than an exception. Files are divided in chunks, then stored and replicated. The system architecture is made up of a single master server, multiple chunks servers and clients.

1. Introduction

We have implemented Google File System (GFS) to meet the needs of data-intensive applications. It runs on commodity hardware and is fault tolerant. It can serve larger number of clients as compared to traditional file systems and as a result provides better performance. Even though GFS shares many similar goals as traditional distributed file systems, its design has been driven on the features mentioned below.

First, as GFS runs on commodity hardware failures are bound to happen. Hence, exceptions are embraced and are considered a part of the system. Thus, constantly monitoring the file system, detecting failures and recovering from them is immensely important.

Secondly, GFS deals with large multi GB files. These, files are divided into smaller files known as chunks and then stored and replicated across the network. Dividing file in chunks enables GFS to serve more clients in turn making the system more efficient.

Lastly, most of the stored files are updated by appending data rather than overwriting existing data. Once, the files are written, they are mostly read and random writes within the files are practically non-existent.

The rest of the report is organized as follows; we discuss the challenges faced by GFS and how we overcome them section 2 and 3. Section 4 describes system architecture and system implementation. Section 5 describes future scope and finally we conclude the report in section 6.

2. Challenges

The challenges faced by GFS are as follows:

- Component failures as it runs on commodity hardware
- Single point failure as there is only a single master server
- Data consistency issue as chunk files need to be replicated across various chunk servers

3. Solution

In order to tackle the failure aspect of the system, data replication is done. We have replicated chunk files at two chunk servers. Hence, if a chunk server fails, all of the files maintained by it will be available at some other chunk server. Hence, data is guaranteed to be available at all times.

Secondly, we have a backup master server which is consistent with the original master server. When the original master server goes down, this backup master server takes its place. This solves the problem of single point failure.

Lastly, data append and write only take place at primary chunk server (which is nothing but a chunk server which behaves as master for other chunk server temporarily). Primary chunk server than ensures that data is updated at replica chunk server.

4. Implementation

Our implementation consists of a single master server, a backup master server, six chunk server and multiple clients. Figure 1 depicts system architecture of our implementation.

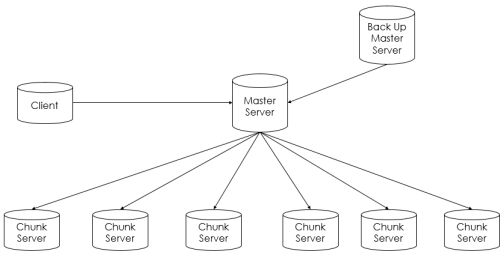

Figure 1. GFS System Architecture

When a client wants to create, read, append or write a file, it will send a request to master server. Depending upon the request, master will either forward it to specific chunk servers or will send a reply to the client with chunk server information. After this, the chunk servers do the needful processing and fulfill client's request. All of the requests are fulfilled by chunk servers not the master server, this prevents bottle neck at the master.

We have handled master server failure. The backup master server maintains same data as the original master server and it checks if the master is alive by sending heartbeats. If the master happens to go down at any point of time while serving a client, then the backup master server informs the client that the original master is down. At this point of time, all clients request will be served by the backup master and it only provides read functionality to the clients.

When a client wants to read, write or append a particular chunk file for the first time, it will send chunk file name to the master. The master replies with chunk server's information having that particular file. Client stores the chunk file name and chunk servers having this file.Thus, if client wants to perform any of the above mentioned operation on this same file, it will check its cache to see if it has chunk server information for that file which it wants to access, if yes than the client directly communicates with the chunk server. This reduces the processing load on master server and prevents bottleneck as well as improving the overall efficiency of the system.

4.1. Create Operation

When a client wants to create a file, he sends a create request to the master server which consists of file name and data to be written on the file. Upon receiving this request, master checks if the data is less than 50 kb, if it is than it appends random data and makes the file size at least 50 kb. It then divides the file into chunks of 10 kb and pushes these chunks on any 2 chunk servers. The chunk server will write these chunk files and notify master, which in turn notifies client. Figure 2 depicts create file operation.

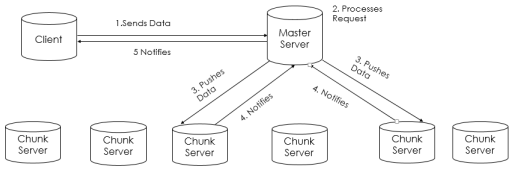

Figure 2. Create Operation

4.2. Write Operation

To write a chunk file, client sends file name to the master server. The master checks its log and provides client with chunk servers having the requested chunk file. The client pushes data on chunk servers provided by the master. It then sends a write request to any one of the chunk server, this chunk server then becomes primary chunk server. The primary chunk server checks if the new data is of size 10 kb, if not then it appends some data to make its size at least 10 kb and then it overwrites the file with this new data and forwards write request to the other chunk server which follows the same procedure. Primary chunk server is notified upon successful write operation by the other chunk server. The primary chunk server then notifies the client. Figure 3 depicts write operation.

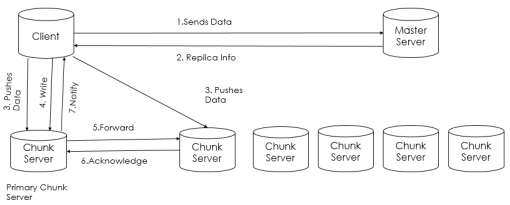

Figure 3. Write Operation

4.3. Read Operation

To read a chunk file, client sends file name to the master server. Master checks which chunk servers have this file and sends their name to client. If the file is not present then the client is notified about the same. The client communicates with any one of these chunk servers and requests file data. Chunk server reads file data into a string and sends it to the client. While sending, chunk server only sends the original data written by the client (i.e. it won't send extra appended data which was done to make the file size of 10 kb). If the chosen chunk server is down, the client will request other chunk server. Figure 4 depicts read operation.

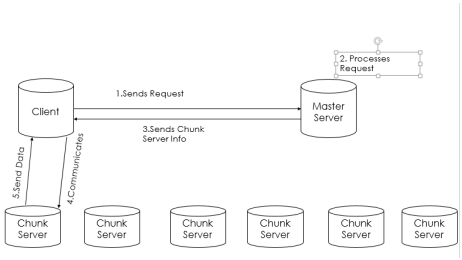

Figure 4. Read Operation

4.4. Append Operation

Client sends name of the chunk file where data is to be appended to the master. Master replies back giving chunk server information where these chunk files are stored.

Client pushes data on the chunk servers received from master and performs write on any one of the chunk server which in turn becomes primary chunk server. Primary chunk server appends data to the chunk file and ensures that the data has been appended on the replicated file stored on the other chunk server and replies back to the client. Figure 5 depicts append operation.

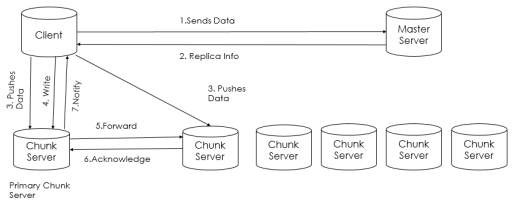

Figure 5. Append Operation

5. Future Scope

Currently, only a single client can append or write data. In the future implementations multi-client append and write functionality can be provided.

We have handled master server failure only. The same can be done for chunk servers by having some backup chunk servers. When any chunk server goes down one of the backup chunk server takes it place. It communicates with the master and all of the files present at the failed chunk server can be stored at the new chunk server.

6. Conclusions

We have successfully implemented Google File System which consists of a master server, a backup master server and 6 chunk servers. We have provided create, read, write and append operation functionality. Multiple client request can be served by GFS.

We have handled master failure there by extinguishing single point failure in the system. By maintaining an additional copy of each chunk file, we handle the scenario of commodity hardware failure.

References

[1] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

[2] Saritha, S. "Google file system." (2010).

[3] Raghunath Nandyala . "Google File System Report."

[4] Wikipedia contributors , "Google File System", Google File System `page version id=719531436`, date of last revision: May 10, 2016 (07:20 UTC), accessed May 11, 2016.