Literature Survey on Distributed File Systems

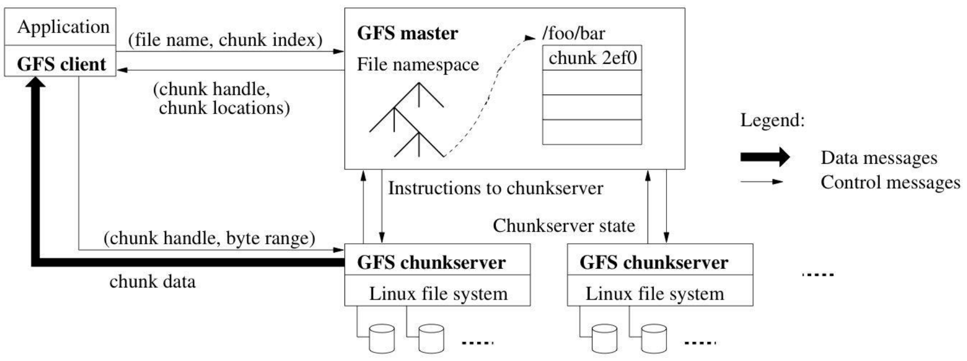
–HUANG Jiaying, TIAN Jingqi

Introduction

As we step into the cloud native era, it is essential to understand the mechanism of distributed file systems, especially GFS (Google File System), in that distributed file systems in general provide data access transparency, location independence, network efficiency, and permanent storage based on a massive number of inexpensive commodity parts. This essay mainly focuses on the research in GFS (*The Google File System, December 2003, Sanjay Ghawat, Howard Gobioff, and Shun-Tak Leung*) to gain some insights on future development.

GFS

1. Design Overview

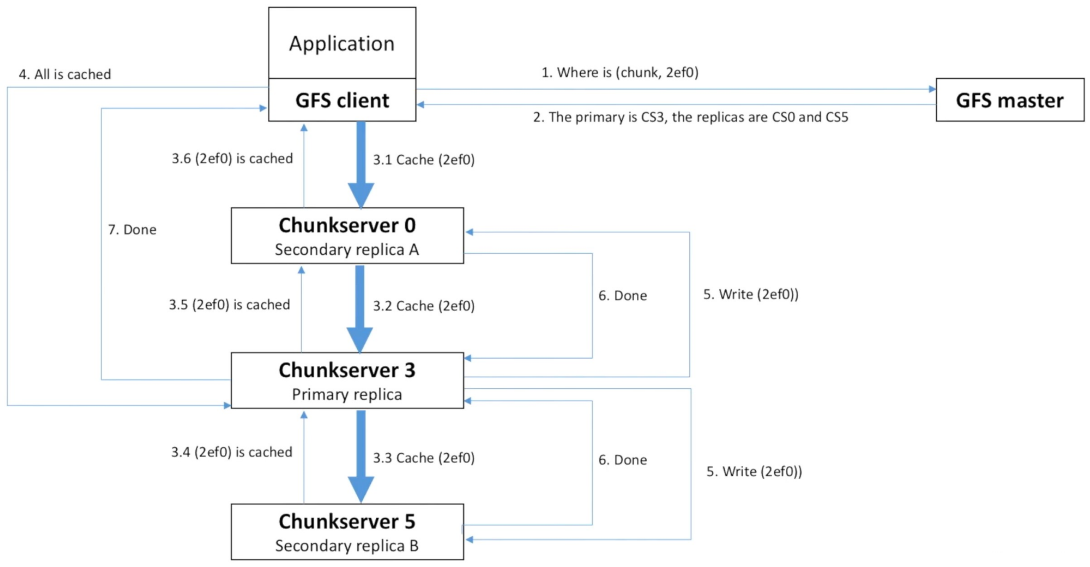


GFS continues to use the master-slave architecture, with each cluster consisting of a single master and multiple chunkservers. Metadata information, including the namespace, access control information, mapping from files to chunks, and current chunk locations, is maintained by the master. Files are divided as fixed-size chunks, each is replicated on multiple chunkservers to prevent data corruption, and stored on local disks as Linux files by the chunkservers.

2. Reduced Client and Master Interaction
   Read and write operations requested by the clients are issued by chunkservers rather than the master to minimize its involvement. The client simply asks the master for free chunk location information, and the rest of the data flow is between the client and the replicas under the supervision of chunkservers.

3. Operation Log
   Operation log functions as the only persistent record of metadata information which is crucial to GFS in that it can identify the order of the creation/version of files and chunks. Assisted by operation log, the master is able to handle file namespace mutations easily.

4. Atomic Record Appends
   Unlike traditional overwrites, GFS applies record append when it comes to write operations, which does not require the client to specify the location, avoiding the situation when multiple users access the same chunk location concurrently. In the meantime, appending is much more efficient and effective to deal with application failures by periodical checkpoints and allowing users to restart incrementally.
5. Namespace Management and Locking
   GFS keeps its namespace as a table, with each record representing full path names to metadata corresponding to a read-write lock. The file creation only needs to read lock on directory name to protect the parent directory from being deleted, renamed, or snapshotted, which enables concurrent mutations in the same directory.
6. High Availability
   This nice property is guaranteed by fast recovery and replication in GFS. Both chunks and masters are replicated for reliability purpose. For chunk replication, different replication levels can be specified by the users. For masters, GFS has shadow masters to provide read-only operations when the primary master goes offline.
7. Write Mechanism
   GFS has a special mechanism for the write operation. The client asks for locations to write and get replica locations from the Master. Then the client will first write data to the closest chunkserver but not all the chunkservers, because the bandwidth between chunkservers will be much higher. Then the chunkservers can use Linux's buffer cache to cache the new information, but not directly write them. This strategy can efficiently avoid written failure. When all chunkservers finish caching, the write operation will start. The primary replica will inform the client after the whole write operation.



Comparison between GFS And HDFS

HDFS (Hadoop Distributed File System) is also developed and implemented to handle huge amounts of data and provide high throughputs. It is inspired by GFS. Differences and similarities between them can provide a general view of the popular file system.

	GFS	HDFS
Objective	To handle large files. Designed more for batch processing	To handle large files. Designed more for batch processing
Operating system	Linux kernel	Cross-platform
Develop language	C, C++ environment	Java environment
Scalability	Cluster-based architecture. Each cluster may consist of thousands of nodes with huge data size storage.	Cluster-based architecture. Each cluster may consist of thousands of nodes with huge data size storage.
Types of Nodes used	Master and Chunkserver	NameNode and DataNode
File Serving	Files are divided into fix size units called chunks. The default chunk size is 64MB and can be changed by the user. Chunks can be stored on different nodes in cluster for load balancing and storage management.	Files are divided into units called blocks. The default block size is 128MB and can be changed by the user. Blocks can be stored on different nodes in cluster for balancing storage resources and demand.

File Management	GFS supports a hierarchical directories data structure and access by path names.	HDFS only supports a traditional hierarchical directories data structure.
Hardware used	Commodity Hardware or Server	Commodity Hardware or Server
Database Files	Bigtable	HBase
Snapshots	Each Directory and file can be snapshotted.	Allowed up to 65536 snapshots for each directory in HDFS 2.
Cache Management	Clients do cache metadata. Chunks are stored as local files in a Linux system. Linux buffer cache helps keep frequently accessed data in memory	HDFS uses distributed cache facility using MapReduce framework
Cache Consistency	Append-once-read-many model	Write-once-read-many model
MetaData	MetaData information managed by Master	MetaData information Managed by NameNode

Conclusion

Pros:

1)Heartbeat strategies enable each chunkserver to be constantly monitored. If a chunkserver fails, it can be detected and recovered.

2) Constant monitoring and replication, which includes chunk and master replication and recovery, provide high availability and fault tolerance.

3)GFS uses an efficient centralized approach to simplify the design, meanwhile, it can deliver good performance for its main design goals-- support large files.

4)Earlier GFS couldn't handle a heavy workload because of the imperfection of the design structure in master. Now GFS has improved the performance by upgrading the structures to allow binary searches.

6)Automatic garbage collection of orphaned chunks.

7)The bandwidth is increased by operation log, garbage collection and other batch operations.

Cons:

1)GFS is not optimized for small-sized files less than 100MB. Small files will have few chunks, sometimes even one. In case of many client requests, these chunkservers that store small files will become hot spots. And these small files will be involved in master processing, causing a potential bottleneck.

2)Though the data structures can do efficient binary searches, the master still can be a bottleneck sometimes.

3)Because of the appending strategies, random file writing or efficient file modification can't be handled by GFS.

4)GFS only optimizes the data processing rate but does not optimize the time performance for a single read or write.

In conclusion, the review introduces the design overview and some special mechanisms of the Google File System, makes a comparison between GFS and another popular file system, HDFS, and summarizes the pros and cons of GFS. From the above way, GFS can successfully support large-scale data processing workloads and manage network maintenance and failures. Many modern DFSs are inspired by GFS.

References

[1] Ghemawat S, Gobioff H, Leung ST. The Google file system. InProceedings of the nineteenth ACM symposium on Operating systems principles 2003 Oct 19 (pp. 29-43).

[2] Verma C, Pandey R. Comparative analysis of GFS and HDFS: technology and architectural landscape. In2018 10th International Conference on Computational Intelligence and Communication Networks (CICN) 2018 Aug 17 (pp. 54-58). IEEE.