Critique of The Google File System

Xin Wang

1 Summary

The paper "The Google File System" [2] introduces a reliable and scalable distributed file system called Google File System(GFS) for large distributed data processing. Compared with old file systems, GFS have dealt with problems such as component failures, rapidly growing file size, appending data to existing files, and co-designing file system API. To provide architecture support, in each GFS cluster, there is a master and multiple chunkservers, which are accessed by clients. The master decides complicated chunk replacement and replica storage, yet involves a little in operations to the file data. Huge files are cut into small chunks in fixed size and stored on local disks of chunkservers. Commonly there are three replicas of each chunk in different chunkservers for the sake of reliability. The master send instructions to chunkservers periodically and collects state information of the chunkservers.The metadata stored on master includes the file and chunk namespace, mapping from files to chunks and the locations of chunk's replicas. The consistency model of GFS is guaranteed by namespace locking and its atomicity.

GFS implements data mutations, atomic record append and snapshot by interaction among the master, chunkservers and clients. In this way, the master focuses mostly on communications, management and decisions, yet involves little in operations such as create, delete, read, write etc. First, leases are used to maintain a consistent mutation order across replicas and the management overhead on the master is minimized. Second, the data flow is decoupled from the control flows and the data is pushed to a chain of chunkservers linearly so as to make full use of newtwork bandwidth. The nearest machines in the network are selected to receive data flows. Third, GFS provides record append for atomicity. Concurrent writes to the same place are specified as data by the clients. GFS appends the data atomically. Forth, the snapshot makes a copy of the file system tree as backup so that users of GFS can checkpoint the current state and then make changes and commit easily.

The master manages chunk replicas and executes namespace operations. Since multiple operations are allowed to be active at the same time, locks are used to lock namespace for serialization. All master operations require locks before operation, yet the locks are acquired in a proper order so that deadlock is avoided. The master also manages and replaces chunk replicas to optimize reliability of GFS. The master creates a chunk and chooses positions of its replicas, re-replicates a chunk when the number of replicas is under a standard, and rebalances replicas periodically for better disk space. When a file is deleted, the master logs the deletion and removes the marked deleted files in its regualr system namespace scan.

2 Positive reactions

One advantage of GFS is its rapid access to data. The master commonly manages three replicas of a data and often put one replica one the same machine as or the closest one from writer of the files. In this way, when data are read, the reads are enabled from the nearest available replica, so that fast access to data is ensured. Another advantage of GFS is that though there are not complicated backup techniques, the system is fault-tolerant but friendly to administers. When some chunkservers in network crashes, the master takes the nodes out of the network and adds in new ones, and uses replicas to recover data. GFS provides good platform for other techniques, for example, with the support of GFS, Map-Reduce shows high performance such that it can process a large number of file chunks in parallel, while developers only see data as keys and values instead of raw bits and bytes and without file descriptors. In addition, GFS provides replicas of disk

chunks so that Map-Reduce works by sharing disk storage yet the Map and Reduce tasks share an integrated GFS that makes thousands of disks behave exactly like the same one. [5]

3 Extension proposal

GFS has relatively simple caching method. To improve read performance, the data can be stored hierarchically. For example, the BigTable [1] uses two levels of caching, with the higher-level caches key-value pairs and lower-level caches blocks read from GFS. Moreover, in GFS, if the commit log is stored for each tablet in a separate log file, many files should be written at the same time in GFS. Also, having separate log files for different tablets will reduce the effectiveness of group commit performance. A good improvement choice is to append mutations to a single commit log for each tablet chunk server, as BigTable does. In GFS, one of the potential optimization is to speed up data replication and distribution by efficient graph algorithm. One of the successful improvement is an edge-disjoint complete graph in server-centric network architecture proposed by Guo et al. [3] The replicas of chunks are chosen to locate at different locations for reliability. By this algorithm, the source and selected chunk servers form a pipeline to reduce the replication time because the selected servers are located at different levels in the graph. Similarly, algorithms that can utilize multiple links at a server to accelerate file replication and recovery is also helpful to deal with the bandwidth limit in GFS network. DCell [4]is one of this success example.

4 Conclusions

To sum up, the fast and automatic recovery techique, crucial chunk replication scheme, data integrity, constant monitoring and the diagnostic tools of GFS have made it a robust and reliable one supporting large-scale data processing on cheap commodity hardware. The experimental results and many algorithms based on this platform have proved the effectiveness of the system.

References

[1] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[3] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[4] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 75–86, New York, NY, USA, 2008. ACM.

[5] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 1029–1040, New York, NY, USA, 2007. ACM.