Homework-1: Summary of The Google File System

Name: Kunal Baweja
UNI: kb2896
COMS E6998 Cloud Computing & Big Data (Fall 2016) September 26, 2016

Introduction

This article summarizes the various aspects, design principles and goals of *Google File System*[1] presented by Sanjay Ghemawat, Howard obioff and Shun-Tak Leung, as researchers at Google.

The *Google File System(GFS)*[1] is a scalable distributed file system designed for large, distributed and data intensive applications. It provides efficient performance on large number of clients and storage machines and also offers fault tolerance by constant monitoring, replication of crucial data, and fast and automatic recovery. In addition, *GFS*[1] is capable of running on inexpensive commodity hardware. It mininmizes the metadata storage for file system and separates data flow from control flow to maximize network bandwidth usage. The relaxed consistency model of *GFS*[1] simplifies file system design greatly by excluding factors such as symlinks etc in comparison to traditional file systems and also provides crucial guarantees in terms of data availability to applications using *GFS*[1].

Design: The data intensive distributed applications require guarantees in terms of data availability, integrity, reduced network latency, file system consistency and efficient metadata management. *GFS*[1] tries to address these concerns by making following design considerations:

1. Component failures are considered as a norm rather than an exception to tackle failure of hardware components.
2. The average size of files stoered on *GFS*[1] is assumed to be of multiple gigabytes.
3. Most files are modified by appending new data, rather than overwriting existing data and once written, all files are read-only in a sequential manner.
4. API designs for *GFS*[1] and applications intended to run over it have been optimised to support high read and write throughputs along with multiple clients concurrently appending to a file. This is supported with *atomicity of writes* guarantee provided by relaxed consistency model for *GFS*[1].
5. High sustained bandwidth is more important than low latency as it's assumed that most applications prioritize processing large amounts of data than quick response times.

File System Interface: Apart from the standard file system operations, *create, delete, open, close, read and write*, *Google File System*[1] provides two new operations:

- The *snapshot* operation facilitates users by creating a copy of a file or directory tree at low cost, sometimes even receursively if required by user.
- The *record append* operation allows multiple clients to append data concurrently to a file while guaranteeing atomicity of operations.

Architecture: A *GFS*[1] cluster consists of single *master* and multiple *chunkservers* which are accessed concurrently by multiple clients. Files are divided into fixed-size *chunks*. Each *chunk* is identified by an immutable and globally unique 64 bit *chunk handle* assigned by the master at the time of chunk creation. Chunkservers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers and the metadata of the whole file system including information about chunks is maintained by the master node. Clients interact with the master node for metadata related operations but all data-bearing communication is directly handled by chunkservers, as delegated by the master.

The single master node, specifically handles all filesystem metadata operations and records operation logs.It has minimal or no involvment in reading and writing data, thus preventing it from becoming a bottleneck in I/O operations. The client contacts the master node for chunkserver information for read/write operations. The master replies with the corresponding chunk handle and locations of the replicas, client caches this response using file name and chunk index as the key and then directly contacts relevant chunkservers in the future for data operations.

Fault Tolerance and Diagnosis: *GFS*[1] applies an array of measures to incorporate fault tolerance against possible failure of system components:

1. **High Availability** - *GFS*[1] employs *fast recovery* wherein it doesn't distinguishes between normal and abnormal termination of master or chunkserver, hence enabling them to restore their state and start within a few seconds irrespective of cause of shutting down. It also implements *chunk replication* to deal with situations like disk failures, network failure or server failure etc, so in case one or more chunkservers become unavailable, there are replicas available to take up the requests for data on their behalf. *GFS*[1] also replicates metadata stored in *master* node to safeguard against failure of master node.
2. **Data Integrity**: Each chunkserver uses checksumming to detect corruption of stored data. To confirm integrity of data across multiple replicas, the chunkservers obtain checksum(e.g., MD5) which is compared against the checksum values of othe chunkservers containing the replicas of the same chunk of data to detect anamolies. This mechanism prevents against data integrity failures due to component failures or read-write failures.
3. **Diagnostic Tools**: *GFS*[1] servers generate diagnostic logs that record significant events and prove helpful in problem isolation, debugging and performance analysis. The performance effect of this logging is minimal whereas benefits are huge, as controlling and observing a cluster of thousands of nodes is not an easy task without proper logging mechanisms.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, (New York, NY, USA), pp. 29–43, ACM, 2003.