Paper Review –I

The Google File System

Title: The Google File System
Authors: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung Google*

Summary

The paper 'The Google File System' provides the design and implementation details of Google File System, A scalable distributed file system for large distributed data-intensive applications. By making assumptions about large data intensive applications usage and performance, the Authors have developed a Scalable File System that has successfully met the storage needs within Google to be deployed as the Storage platform for the generation and processing of large data sets.

Good Aspects

The GFS has taken good consideration into both Applications requirements as well as the Storage requirements and provides an implementation by balancing tradeoffs in a well and optimized manner.

The GFS has done away with traditional approach of File System implementation by implementing certain features such as Replication, Garbage Collection and Fine Tuning Record Appends and still keeping the API to the applications same with certain minor differences.

Review

Following details will cover all aspect of the paper providing crucial and also interesting aspects of the paper that appealed the reader.

GFS is a new file system to meet growing demands of Google's needs. Its design has been driven by key observations of their Application workloads hence it has a marked departure from traditional approach.

Key Areas Focused

The Authors have explored radically different points in the design space. Following are the key points

1. Constant monitoring, error detection, fault tolerance and Automatic Recovery.
2. File System parameters such as I/O operation and block sizes must be revisited since data is in order of MB and GB
3. Fine tune for append operation which is a pattern for huge files hence focus on performance optimization and Atomicity.
4. When designing, work on both applications and file system API so it helps build consistency model and simplify implementations.

Assumptions

- System is built using inexpensive hardware that has a high failure rate.
- Multi GB files are common and small files are sparse.
- Two kinds of reads – Large streaming reads and Small random reads.
- File System must manage concurrent access to data with Atomicity and efficient Synchronization.
- Focus on bandwidth and less response time than low latency.

Technical Details

- **Interface and Features**
  - Traditional create, delete, open, close, read, write
  - **Atomic Record Append** – Efficiently Appends records Atomically .
  - **Snapshot** – Copy of file or a directory tree at low cost.
  - **Garbage collection** – Free up unused/expired chunks.
  - Chunk lease – A process where master assigns a primary chunk server and multiple secondary chunk servers to replicate record/data across multiple locations using a Mutation order.

- **Technical Terms**
  - Master – A machine that helps in initiation of data access.
  - Chunk – A part of file that is addressable by master.
  - Chunk servers – Servers that are controlled by Master.
  - Cluster – Combination of Master and Chunk servers.
  - Chunk handle – 64bit integer representing the chunk
  - Metadata – A dynamic mapping that describes and provides useful information about operating Chunk servers
  - Mutations – Modifications to existing data residing on master or chunk servers

- **Architecture**
  - Each cluster consists of Single master and multiple chunk servers.
  - A Single Master Simplifies design and enables it to make use of sophisticated chunk placement techniques to achieve balance of utilization, performance and Efficiency.

  - Master stores information (metadata) in memory about all GFS chunk servers that are available. It also stores information of chunk locations by providing namespace mapping to the chunk handle.

  - Master stores this information on memory in the form of a modified BTree with Prefix Compression thus saving memory. Extra Information is store on Operation Logs.

  - Operation log stores important important onto disk thus helping in Restoring in case of failures.

  - In order to achieve maximum performance, master and Chunk communication happens asynchronously for Data messages and Control messages.

  - Master need not poll individual Chunk servers but instead using Heart Beat message (I'm alive) from individual servers to determine if they are still alive or not. Some useful parameters are piggy backed on control messages to save round trip time and leverage existing TCP connection.

  - Application interacts with Master only once and this is using GFS Client code that offloads all communication work from the application.

  - Chunk size of 64MB is a default chosen block size considering access pattern.

- **Consistency Model**

  - A relaxed consistency model is central to the file system implementation and API implementation
  - File namespace mutations are atomic.
  - State of a file region after a data mutation depends on type of mutation.
  - Consistency is achieved if all clients see the same data regardless of which replica.
  - How GFS achieves consistency?
    - Applying mutations to a chunk in the same order on all its replicas.
    - Using chunk numbers to detect any replica that has become stale because it missed mutation when the chunk server was down.
    - Stale replicas are Garbage Collected.
  - Achieved in a timely manner to ensure fair usage of Network bandwidth.
  - Beyond Control? – A chunk is lost irreversibly only if all its replicas are lost before GFS can react.

- **Replication**

  Replication is central to the GFS since its highly distributed and communication can be across many number of switches

  Chunk replica placement is to ensure maximum data reliability and availability and maximize network bandwidth utilization.

  Why create Chunk Replicas?

  a. Place replicas on chunk servers where disk space is under utilized.
  b. Limit the number of recent creations on each chunk server.
  c. Spread replicas of a chunk across racks.

- **Fault Tolerance and Diagnosis**

  One of the greatest challenges in GFS is to work on low quality hardware that has high failure rate and this is guaranteed by tools that diagnose when these events occur

  1. High Availability is achieved by Fast recovery, Chunk Replication, Master Replication.
  2. Data Integrity is measured by calculating checksum to detect corruption in the stored data. Each chunk server can calculate the checksum and reports errors by piggy backing messages via the control signal to the master server.

- **Performance**
  - Reads - The aggregate read rate reaches 94 MB/s about 75% of the 125 MB/s link limit for 16 readers so efficiency drops as the number of readers increases and so does the probability that multiple readers read simultaneously.
  - Writes – The write rate is 6.3 MB/s which is half of the limit owing to the network stack and delay in propagation. However, it does not significantly affect the aggregate write bandwidth.
  - Record Appends – One again the performance is limited by the network bandwidth of individual chunk servers that store the last chunk of a file. However, its not a significant issue since client is decoupled when chunk servers are writing files since its stored in LRU queue.

Conclusion and Summary

The following paper gives an excellent overview of storage needs and challenges in a network paradigm. The paper seems to have captured all technical aspects but provided lucid explanation to the reader. The paper how ever assumes reader to have knowledge of at least what a file system does and what a file system API comprises of.

The implementation of Chunk replication, Garbage Collection strategy of Unused/Stale Chunks, Master – Client decoupling, LRU queue for each Chunk server indicates the modernization of File System that is otherwise not yet available.

The Authors have justified every loss in read/write/append by mentioning the reasoning and also giving an aggregate analysis of performance which is almost close to the actual perceived performance.

I believe the paper and implementation could have also covered about working/Fine tuning the Network stack in order to improvise efficiency and bandwidth. How ever this might cause incompatibility but for google this can help improve their storage needs since it’s a proprietary file system.