The Google File System

Joseph Archer

November 25th 2013

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

Main Idea of the paper

The Google File System is a scalable distributed file system that has successfully met the storage needs of Google. The file system is used for large distributed data intensive applications while running on inexpensive hardware. It provides high availability as well as data integrity.

How that idea is implemented

Interface	Architecture	Single Master	Chunk Size	Metadata
- Provides a familiar file system interface - Files are stored hierarchically in directories and identified by pathnames - Supports the usual operations such as Create, Delete , Open , Close , Read , Write - GFS has a snapshot and record append operations	- A cluster consists of one master and multiple chunkservers - Files are divided into fixed-size chunks - The master maintains all file system metadata. - For reliability, each chunk is replicated on multiple chunkservers	- Simplifies the design - Allows the master to make intelligent chunk placement and replication decisions - Clients never read and write file data through the master - Clients typically ask for multiple chunks in the same request, and the maser can also include the information for chunks immediately	- Uses a 64MB chunk size which is larger than a typical file system - The large chunk size reduces the clients need to interact with the maser - Each chunk replica is stored as a plain Linux file on a chunkserver - Lazy Space allocation is used to avoid wasting space due to internal fragmentation	- All the metadata is kept is the master's memory - The Master does not store chunk location information persistently - The master scans through its entire state periodically to implement chunk garbage collection - An operation log is kept that contains a historical log of critical metadata changes and the masters uses it to recover its state

Analysis of the idea and implementation

- Design

The design of the file system is efficient , well organized , and reliable.

In my opinion the strong design assumptions played a large role in the success of the file system.

- Interactions

The decision to push the flow of data linearly is a major factor that greatly increases the efficiency of the file system.

- Fault Tolerance

The file system provides an ample amount of fault tolerance using fast recovery as well as replication.

The levels of replication and speed of system recovery allow the system to be reliable and efficient.

Advantages and Disadvantages

Advantages	Disadvantages
- **Lazy garbage collection** After a file is deleted , the physical storage is not immediately reclaimed making the system more reliable - **Data Flow** Data is pushed linearly along a chain of chunkservers machines full outward bandwidth is used to transfer data as fast as possible rather than divided among multiple recipients **- High Availability** The file system uses fast recovery allowing both the master and the chunkservers to restore their state in seconds The master and chunkservers are both replicated for reliability	- **System is not optimized for small files** A small file consists of a small number of chunks, perhaps just one. The chunk servers storing those chunks may become hot spots if many clients are accessing the same file **- Component failures are the norm rather than the exception** The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines

Real world use cases

Two clusters are in use within Google

Cluster A

- Used regularly for research and development by over a hundred engineers
- A typical task is initiated by a human user and runs up to several hours
- It reads through a few MBs to a few TBs of data , transforms or analyzes the data , and writes the results back to the cluster

Cluster B

- Used for production data processing
- The tasks last longer than cluster a and continuously generate and process multi-TB data sets with only occasional human intervention

In both cases , a single “task” consists of many processes on many machines reading and writing many files simultaneously