GOOGLE FILE SYSTEM

Team : Learners
Nainy Sharma(20172008), M Chitra(20172009),
Swati Bhandari(20172013), Richa Kushwaha(20172056)

ABSTRACT

We have designed and implemented Google File System, a scalable distributed file system. It serves large number of clients. It shares many of the same goals as previous distributed file system like network transparency, location transparency, location independence, user mobility, scalabity, etc.

In this report, we present file system interface to support distributed application, discuss our design aspects and real world use.

1. INTRODUCTION

We designed and implemented Google File System to meet the rapidly increasing demand of data processing need. Its design is based on the observations of our application workload. Google File System (GFS or GoogleFS) is a proprietary distributed file system developed by Google to provide efficient, reliable access to data using large clusters of commodity hardware. A new version of Google File System code named Colossus was released in 2010.

2. DESIGN OVERVIEW

2.1 Assumptions

While designing this file system, we considered the following assumptions to help us design it with better functionalities.

- The system is formed by many componenets which can fail. It must monitor itself to detect, tolerate and recover from such failures.

- The system stores large files efficiently and also supports storage of small files, but no need to optimize for them.

- The workload consists of large reads, small reads and large sequential writes. In large reads, individual operations read hundreds of KBs or, even an MB or more maybe from contiguous region of file. Write operations are similar to read operations where data is appended to the file. However, files once written are often modified again.

2.2 Interface

GFS provides a familiar file system interface, though does not implement a standard interface. Files are oraganized hierarchially in directories and accessed through pathnames. The supported operations are put, get and delete files in/from the file system.

2.3 Architecture

GFS consists of a single masterserver and mutiple chunkservers, that are accessed by mutiple clients. We can also run both chunkserver and client on the same machine, as long as machine resources permits.

Files are divided into chunks of fixed size. Each chunk has a unique id which is assigned at the time of chunk creation by the masterserver. Chunkservers store these chunks as Linux files on the local disk. By default we store two copies which we can change depending on our need.

The masterserver maintains all file metadata such as unique chunk id and the chunkserver address. The masterserver periodically checks the state of the chunkserver through Heartbeat messages. The GFS client communicates with the masterserver and the chunkserver to perform read, write or delete operations on behalf of the application.

The masterserver only contains the metadata, and the actual data is stored at the chunkservers so the client communicates with the masterserver to get the metadata and use it to perform data transaction with the chunkserver.

2.4 Single Masterserver

Having a single masterserver simplifies our design and enables the masterserver to make sophisticated chunk placement and replication decisions. Clients never read and write file data through the masterserver. Instead, a client asks the masterserver which chunkserver it should contact. First, using the fixed chunk size the client translates the filename and byte offset specified by the application into a chunk index within the file. Then it sends the masterserver a request containing the filename. The masterserver replies with corresponding IP Address and Port of the chunkserver.

The client then sends request to the chunkserver. Further read of the same chunk require no more client-masterserver interaction. When new chunkservers connect with the masterserver, the masterserver dynamically maintains the table ”activechunk” to maintain their information. This table is automatically deleted when the masterserver goes down. The chunk size maintained is 1MB and the chunk ID is of 128 bits.

2.5 In-Memory Data Structure

Since metadata is stored in memory, masterserver operations are fast. In this project two in-memory data structures are used both by the masterserver – “file_table” which maps the chunk ID created for a particular file to the chunkservers and “activechunk” which stores the currently connected chunkserver’s IP Address and Port.

3. DATA FLOW

The masterserver is started using the command “python server.py” and similarly for chunkserver “python chunkserver.py”. The client can run the commands :
1) python client.py put <file_name> <alias>
2) python client.py get <alias>
3) python client.py delete <alias>

The put command takes the file from the client and divides it according to the chunk size and sends it to randomly selected chunkservers. By default the first selected chunkserver is the primary chunkserver that forwards the chunks to other chunkservers.
The get command retrieves the chunks from chunkservers and assembles it to form the original file and prompts file corrupt if the whole file is not retrieved.

The delete command removes the chunks of the file from the respective chunkservers and also deletes the file entry from the in-memory “file_table” data structure.

Methods used in the code:
In server.py:
In class MasterService: exposed_read, exposed_write, exposed_delete_file_table_entry, exposed_get_file_table_entry, delete_from_chunkserver, exposed_get_block_size, exposed_get_minions, get_chunkservers, calc_num_blocks, exists, alloc_blocks.
In chunkserver.py:
In MinionService: exposed_put, exposed_get, exposed_delete, forward.
In client.py:
send_to_minion, read_from_minion, get, put, delete.

4. ACKNOWLEDGEMENT

We wish to thank the following people for their contributions to the project or the paper: Our respected faculties Prof. Avinash Sharma (IIIT-Hyderabad), Prof. Manish Shrivastava (IIIT-Hyderabad). We also like to thank our project guide Nagaraj Poti (M.Tech IIIT-Hyderabad). We also wish to thanks our classmates for their help in testing the code.

5. REFERENCES

[1] http://www.cs.cornell.edu/courses/cs614/2004sp/papers/gfs.pdf
[2] https://en.wikipedia.org/wiki/Google_File_System
[3] https://research.google.com/archive/gfs.html

Language used: Python 2.7.12