Review, The Google File System

Robert Hoff

November 8, 2011

1 Summary

The paper [1] presents the Google File System (GFS), a distributed file system designed to offer global reach and to operate across thousands of storage nodes. It was designed with Google's special requirements in mind (mainly of size and scope), and is today used as one of the building blocks to service most of the company's key applications. Just a few examples are it's principle search service, Google Earth, Google Analytics and Orkut (social networking). GFS was introduces to replace the existing storage solution at Google called BigFiles [3]

2 The Problem

Most of Google's services require large distributed files systems to manage requirements such as availability, volume and persistency. The data in question must always be accessible, grow to virtually unbounded sizes and have a guarantee not to get lost or corrupted. None of the existing distributed systems at the time matched Google's exact requirements. Most system's were designed for the end user in mind, for example NFS drives (the most popoular) are only limited to one per server (although they can be chained together). Another system, ITC [2] had many of the same design goals as GFS and did work across many servers, but in ITC data integrity was a medium priority, and large files or databases a low priority. In contrast to GFS, support for large files is a key requirement.

3 Approach

The GFS was designed to principally store large files (in chunk sizes of 64MB). This choice reduces the number of control messages required to fetch large volumes of data, and reduces the number of references required on the master. A reference to a chunkserver has a maximum size of 64 bytes, so several hundred million references to large files can be served in RAM by a single machine. Thereby one of the design choices was to designate only a single machine as the master.

Replications of all data is managed across at least 3 chunkservers, and the master polls the chunkservers continuously to check if data is still available and valid. If a server fails, or data is seen to be corrupted, control messages are exchanged to propagate data so that it exists in the minimum number of locations. Each file has a checksum on each of the chunkservers, to provide validation on file-transfers. One of Google's choices is to use commodity hardware for the server's, because no matter how expensive the hardware, the system would still necessitate an assumption of a high failure rate.

GFS is mostly POSIX compliant, but additional functions were added to support specialised tasks. The most important of these is the atomic record appends that adds data to the end of existing files. This makes the service more efficient in it's support of databases (BigTable).

Write locks are implemented by a system called leases, which is a type of permission that is given to a chunkserver as it's data is being changed by a potential client. If a second client requires modification of the same data, those requests will be appended to a write log that is attached to the lease.

To avoid the single point of failure problem that arises with a single master, there exists several shadow masters that keep copies of the master's meta-data. These copies may lag behind slightly, but should the master fail, one of the shadow master's would quickly be instated and eventually refresh it's completed meta-data by polling the chunkservers. Another issue with the master is the possibility of a client failing in the middle of a write operation to the master, this problem is solved by logging all command chains with START and END tokens before processing.

4 Evaluation

Tests show that even when whole chunkservers fail the data is replicated at a high priority within matters of minutes, or possibly hours. The requirement for not losing data is that the two other chunkservers holding the backups do not fail simultaneously during this time.

It is shown that the master can server upto 500 client request a second, which is sufficient for Google's applications. With the designation of 64MB chunks, a single master does not have any practical limitations on storage references. Even the petabytes of data generated by Google's web crawler can be references by only a couple of GB.

Reading or writing a file may take several minutes or more, so issuing leases is a common and important operation.

5 Opinion and Questions

1. Why is there a high proportion of dead files in the examples shown? Does this indicate that the server is to busy to manage the garbage collection, is this somehow intentional?
2. There wasn't much emphasis on security, or permissions. How do the chunkservers know that read and write requests are from approved clients?
3. How widespread is GFS used outside of Google (if at all)?
4. If a set of replicated chunkservers fail at the same, then the master would contain references that would be impossible to resolve. What happens in this case?

References

[1] Ghemeway A. and et al. The google file system. *ACM SIGOPS*, 2003.

[2] Satyanarayanan M. The itc distributed file system: principles and design. *ACM symposium on Operating systems principles*, 1985.

[3] Lawrence P. Sergey B. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 1998.