Megastore

Daniel Hand / 16405352 / Group J

Synopsis:

In this paper the authors discuss the large-scale distributed storage system, Megastore, used in Google's cloud infrastructure that is designed to adhere to 5 key principles, many usually in conflict. As the growth achieved in modern services is often exponential, this system had to be *highly scalable* and support *rapid development* while retaining rich features such as *highly available*, *consistent data* with *low latency*, not offered in many large-scale storage systems such as Google's BigTable. In this sense it combines the scalability of NoSQL and the power of RDBMS with data partitioning of the storage ensuring atomic, consistent, isolated, and durable (ACID) transactions giving performance close to ext3 [1], popular with Linux distributions.

The authors discuss the methods employed to build a large, reliable storage platform that alleviates the problems associated with storing large amounts of data such as ensuring availability through the use of Paxos, a consensus protocol that is fault-tolerant. Further improvements to availability came from the use of numerous replicator logs, each of which controlled a partition of their data and stored *transactions* (a set of read and write operations with a commit request) with only a single transaction being capable of altering the log, guaranteeing atomicity and consistency.

This leads on to the partitioning approaches used in which *entity groups* (each a mini-database) are apportioned to each datastore in a datacentre allowing for completely independent, synchronous replication with serializable ACID semantics preserved within an entity group irrespective of what replica a client is initiated from. The advantage of this design was using Megastore's asynchronous messaging service, a queue, for operations across entity groups that were logically different but not physically distant leaving the network free for synchronous, consistent replicated transactions. Cross-group atomic updates are possible by *two-phase commit* but this usually incurs higher latency [2] Care had to be taken when selecting boundaries for the entity groups so as not to overlap which would affect throughput while also not using boundaries with too fine a grain so group-to-group operations were minimised.

Megastore needed to perform at high capacity meaning an expressive querying language, if users' operations were executed at query time performance would be severely affected. The authors instead opted for a predictive language which they developed to require few joins by constructing hierarchical table layouts and declarative denormalization via *inline indexing*. This hierarchical structure where tables are either *entity group root* tables or *child* tables (where a child table references a root table via a foreign key and each root can have multiple child tables) lends itself extremely well to the local, global and repeated indexing Google uses for tags in its Google Photos service [3].

Reads in this system are tracked by a *coordinator* giving better utilisation and decreasing latency. Unlike other paradigms, writes are controlled by *leaders* instead of masters where the first writer gets precedence, forcing the other replicas to accept that write with a policy designed to use the closest replica. *Witness replicas* also exist with the sole role of preventing stalemates. Failures are mitigated by using Google's Chubby lock service that takes multiple backups a day, comparing replicas to each other every few hours [4].

Along with this failure detection, the authors have limited write rates an entity can perform each second to attempt to decrease cross-group latency. For clients that must exceed this limit, advisory locks are available to serialize pieces of transactions that contain conflicts. If a connection is unreliable or lost to a exceptionally full replica, some fairly crude procedures must be employed, such as completely disabling the replica but luckily this is rarely used with methods such as nominating a healthier leader for the next write or rerouting traffic to other replicas more often used. Overall performance is impressive with average read latencies of milliseconds and write latencies of between 100 and 400 milliseconds. As the system was designed with testing in mind, bugs are quickly found via the *pseudo-random test framework*. With over 100 applications tackling internal and external users Megastore has shown how the 5 principles mentioned can be consolidated into a single distributed storage system.
========== THIS SHOULD NOT GO ONTO PAGE 2 ==========

References

[1] C. P. Wright, R. Spillane, G. Sivathanu, and E. Zadok, “Extending {ACID} Semantics to the File System,” *ACM Trans. Storage*, vol. 3, no. 2, pp. 1–42, Jun. 2007.

[2] X. R. Guerin and M. Shicong, “Passive two-phase commit system for high-performance distributed transaction execution.” Google Patents, 2015.

[3] M. Frigon, “Users tagging users in photos online.” Google Patents, 2015.

[4] M. Burrows, “The Chubby lock service for loosely-coupled distributed systems,” in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 335–350.