Date: 25th February 2019

CMPT 756: Systems for Big Data (Spring 2019) – Assignment 1

Identifying and Exploring Design Trade-Offs of a Spanner Architecture

Anurag Bejju – abejju@sfu.ca

Introduction

When *Spanner* was formally introduced at the *Google I/O '17* annual developer festival, *Dominic Preuss – Director of Product* Management shared some great insights about the evolution of data management systems at Google. In his keynote [1], he spoke about the major limitations Google faced with its manually sharded MySQL database cluster in 2005 built to support its advertising system, AdWords. The problem arose when a customer outgrows a shard and needs to be moved to another one, that transfer took many years to finish. Since it was a major constraint for the growth of the service, a solution had to be designed that can address various systems aspects like scalability, automatic sharding, fault tolerance, consistent replication, external consistency, and wide-area distribution [2]. Later in two years, Cloud spanner was put to production and replaced the MySQL database cluster to effectively support AdWords System.

Real World Applications

After its initial roll-out, many mission-critical services at Google were migrated onto spanner in the following years. It also collaborated with other businesses that requires a globally distributed transactional database which provides horizontal scalability, availability, and performance with a relational schema and SQL access. Companies like *Looker* [3] uses Cloud Spanner to enable users to query data in Cloud Spanner without the need to create read or slave replicas and impact production workloads. Also, cross-platform applications like *Evernote* [4] are migrating to a Cloud Spanner instance to efficiently manage more than 8 billion pieces of its customers' notes and replace over 750 MySQL instances.

Salient Design Features

From a key-value store that provided multirow transactions, external consistency, and transparent failover across datacenters, it has been transformed to a broader relational database system without changing the core features it initially offered. Some of the unique design aspects of spanner include a robust distributed query execution, better range extraction, effectively handle server-side failures and have a common SQL dialect widely used by software developers across the globe. All of these features are achieved due to its distinctive architecture built to behave like a *sharded, geo-replicated relational database*. That means even though the database behaves like a single one unit system, the data is sharded across multiple machines in a single data center as well as replicated in other data centers in different geolocations in order to balance load as well as effectively respond to system failures. This architecture helps google scale up to millions of machines across hundreds of datacenters and trillions of database rows

One of the most awarding design element of the spanner is its property to provide a *highly consistent distributed system* as well as allow multiple clients to simultaneously interact with it. In order to ensure the consistency of multiple concurrent transactions, Cloud Spanner uses a combination of shared locks and exclusive locks to control access to the data. When you perform a read as part of a transaction, Cloud Spanner acquires shared read locks, which allows other reads to still access the data until your transaction is ready to commit.

When your transaction is committing and writes are being applied, the transaction attempts to upgrade to an exclusive lock. It blocks new shared read locks on the data, waits for existing shared read locks to clear, then places an exclusive lock for exclusive access to the data. By making transactions use a replicated write-ahead redo log and *Paxus consensus algorithm* it does try to ensure availability of system despite of server, network and data center failures. It also ensures a high consistency of data by using a combination of pessimistic locking and timestamps. For blind write and read-modify-write transactions, strict two-phase locking ensures serializability within a given Paxos group, while two-phase commits (where different Paxos groups are participants) ensure serializability across the database.

It also tries to be consistent in terms of syntax or language used across several platforms. Systems like *F1, Dremel* and *BigQuery3* produced by Google share a common SQL dialect, called "*Standard SQL*". This ensures all applications share a common data model, type system, syntax, semantics, and function library in a uniform manner. It also helps developers or data analyst to better work with multiple systems offered by google without really worrying about the need to learn new syntax or language for each of them.

Spanner achieve high levels of concurrency and consistency by using a *robust distributed query processor* that executes code in parallel on many machines that host data in order to better serve both online and long running queries. A Spanner SQL query compiler that builds a relational algebra operator tree and optimize it by using equivalent rewrites. Since query distribution uses explicit operators while building the query algebra tree, distribution operator Distributed Union is used to migrate each subquery to a different shard of data, and concatenate the results later. This distribution operator performs a key operation in Spanner since the sharding of a table might change while the execution of a query or after a query restart. During this, the data that was once available locally might be moved to a different location making static partitioning for query planning completely unreliable.

Query range extraction is one more design feature that spanner uses to analyze a query and determining what portions of tables are referenced by the query. These referenced row ranges are indicated as intervals of primary key values and provides various methods to perform it. Although Range extraction mostly gives minimal key ranges for complex expressions, but it is still an approximation that may not always give the best results. In a worst case scenario, the subquery might be sent to an irrelevant shard which will return no rows (bad for availability). This could be a tradeoff between *seeks* vs. *scans* and involves better optimization of query range extraction methods.

Biggest Design Trade-Off: Consistency for Availability

Despite being a distributed system, this paper does portray spanner to be a solution that is both: *Highly Available* and *very consistent* in nature. That said, it does not really dwell too much to back its claims of being highly available as compared to it being consistent. In fact, it briefly talks about it in just one paragraph where its states – "*Spanner uses a form of Multi-Paxos and two-phase commits, in which a single long-lived leader is elected and can commit multiple log entries in parallel, to achieve high throughput. Because Paxos can make progress as long as a majority of replicas are up, we achieve high availability despite server, network and data center failures*". The key statement is "*as long as a majority of replicas are up, we achieve high availability*".

On further analyzing how Paxos algrorithm works and the drawbacks of having a two-phase commits system, I have come down to the following conclusions:

- When a distributed data system like spanner uses Paxos groups to get consensus and the leader of the pack cannot maintain a majority due to a partition, it stalls any ongoing updates which will result in the system being not available. Even though, eventually a new leader will be chosen, that operation also requires a majority of systems to be available.
- Also when we use a two-phase commit to perform cross-group transactions, the members of the paxos group can prevent commits during a partition.

That said, spanner does provide industry-leading *99.999% (five 9s) availability* helping customers manage effects of system failures, data resharding, binary rollouts using a combination of query distribution operators, modified rules, internal iterator interfaces extension, etc. Even though these attributes do help spanner maintain consistency across a shareded distributed system, it has some shortcomings when it comes to guaranteeing 100% availability of the system.

Spanner as a Bad Design Choice: Example Use Case

As part of my Big Data II project, I have been studying about the design and architecture of *block chain* and *cryptocurrencies* technologies. Blockchain networks (like Bitcoin) use sharded distributed systems that can scale immensely. Since millions of transactions happen every minute inside a block chain, any connectivity problem, or any failing node/component should not interrupt the live transactions of a cryptocurrency. That means there is a tradeoff between availability over consistency. These networks do use other methods like mining & blocks to become eventually consistent over time. If spanner is used in this scenario, it ensures the data is consistent but the client has to wait till the majority block chain has come to a consensus on it. This would lead to undesired latency and availability problems. It also have the risks of becoming unavailable if there's a network sharding that could prevent a consensus to happen.

Conclusion

Tradeoffs involved in constructing a distributed database system is often very complex and require extensive design and architectural considerations to be made. Since these tradeoffs are unique to each customer and use case, there can never be one fit for all solution. Spanner is an exceptional solution that provides the best of relational database structure and non-relational database scale and performance with external strong consistency across rows, regions, and continents. That said, its design does tradeoff for consistency over high availability (i.e 100%) which may not be a right fit in some use cases. Overall, spanner is a great enterprise-grade, globally-distributed, and strongly consistent database service with very few design tradeoffs.

References

[1] https://www.youtube.com/watch?v=cOANqaMnkNE
[2] https://ai.google/research/pubs/pub46103
[3] https://www.sdxcentral.com/articles/news/googles-cloud-spanner-global-database-now-actually-global/2017/11/
[4] https://cloud.google.com/spanner/partners/