Spanner: Becoming a SQL System

A short summary by Platon Karageorgis

The following summary focuses on the development of a Google system called Spanner. Spanner was implemented at first as a key-value store, but the need for a schema and the urge to use a strong language like SQL to avoid complicated queries, motivated Google to make a shift and transform it into a complete OLTP relational database system. Spanner is broadly used at Google managing data and is a part of big applications, like Google Play that handle huge amounts of data.

For a start, the authors complete a background check on Spanner's architecture. Spanner is a geo-replicated RDBMS that puts into effect horizontal sharding. The Paxos algorithm is an important aspect of Spanner, since it is the algorithm that makes replication happen. Paxos operates in groups that contain one or more shard and is responsible for gathering the votes of the replicas. The vote is about deciding the content of the log entries. To face concurrency issues, Paxos uses 2-phase locking while timestamps are assigned to transactions. In addition, the coprocessor framework is a useful tool for Spanner that makes sure to conceal the complexity of finding where data is stored, whilst Spanner uses LSM trees for storing. When it comes to query compiling, the function is similar to a normal RDBMS, but a significant part that needs to be mentioned is the usage of Cross Apply and Outer Apply. These are correlated logical join operators and the algebraic tree relies heavily on them.

Query distribution is a challenging segment of Spanner. As the Spanner works in many different computers and parallelization is present, the manner that it distributes queries is crucial. The method that it mainly follows, is the common one for relational databases. The usage of operators is interesting, one of them being the Distributed Union which is placed on top of every Spanner table, it first sends the parts of the query to different shards and is responsible for merging their outcomes. Group By, Top and other similar operations are performed beforehand, as they have input from many different shards. Furthermore, Distributed Union takes advantage of the aspects of coprocessor mentioned above, and achieves latency reduction when it comes to the routing of a subquery action. It does that by dispatching them to the shard groups in parallel. On the other hand, for distributed joins the Distributed Apply operator is the protagonist, by using the previous apply methods along with Distributed Union, to achieve parallelization and to reduce the summation of the machine calls for key-based joins. Additionally, Spanner has two different API models, the single consumer and the parallel consumer.

Continuing the analyses, the authors move on to query range extraction that is about the examination of queries, to conclude about what percentage of each table is used when they are running. There are many methods of range extraction with some of them being distribution range extraction, seek range extraction and lock range extraction. These happen mostly on the WHERE clause and operate at runtime, whereas the rewriting happens during compilation time. In addition, the filter tree is a data structure and an extra add on that performs bottom-up calculations and runs solely at runtime. In general, range extraction is aiming to the generation of key ranges that are small even if the query is complicated.

Query restarting is another significant topic of the Spanner, as it conceals almost completely the network errors that might occur during a query execution, resulting usually to a decrease of latency. This means that for example, if a snapshot transaction fails, a message will not be generated and sent to the client. In addition, Spanner doesn't initiate retry loops and uses paging to get chunks of the query results that will fit into memory. It also faces long running queries and implements recurring rolling upgrades, which contribute significantly when it comes to the dealing of bugs. On the other hand though, query restarting has to overcome some issues that should be noted. For a start, it has to face dynamic resharding meaning that if a server crashes but a part of a query on a shard has started to be computed, a way must be found so that the progress is not lost. Moreover, non-determinism is an issue because some results are returned in a non-repeatable form, making the restarts a lot trickier as it is very difficult to catch up after a restart without tracking large results.

One of the main reasons that Spanner became what it currently is, is the urge for a strong SQL-like language, but the language that it uses is not exactly SQL. Spanner uses a similar language called Standard SQL that attempts to cover SQL gaps. The most important trait it has, is that it ensures consistency via three shared-components, compiler front-end, library of scalar functions and shared testing framework.

Last but not least, it needs to be outlined that the SSTable, a data format used for NoSQL databases, was initially obtained by Spanner, but it was transformed to a new type called Ressi data layout. This layout saves databases as LSM trees.

Which questions aren't answered by this text?

I feel that the paper has many challenging ideas that are not really explained, especially in the query distribution part when it comes to the differences between compilation and execution. In addition, the examples that the authors attempt to make and the pictures that illustrated them are really confusing and not explained clearly.

What has changed since this was written?

The paper was written in 2017 so I consider it is up to date.