CS 657 - Paper Review 4

Spanner: Google's Globally-Distributed Database

Nick Alvarez

4 May 2021

1 Clarity

The paper is presented in a clear manner, despite leaning a bit heavy on the abstract side when describing TrueTime. The authors describe the issues with current databases, their improvements on the databases with Spanner, how it was implemented, and real-world applications. The benchmarks could have used some more practical applications as they were presented as just numbers and it was difficult to draw comparisons to useful operations.

Presenting a new database requires an examination of what is wrong with current databases, how it is fixed with the application, and examples of how it works. It is an excellent way to present a paper, however it seems the authors went a bit light on the examples section.

2 Problems and Existing Solutions

This paper recognizes the lack of viable solutions for managing cross-datacenter replicated data. Issues such as write throughout, consistency, and trouble with complex queries are present on other databases. Additionally, a database can tout consistency but may not necessarily guarantee it.

These problems are important, at least for globally distributed databases, because there may be any number of reads and/or writes going on at the same time. Factor in network delays, a potential down cluster, or other inconveniences, and there are multiple avenues for something to go wrong. Time is also a factor, and for complex queries on expansive datasets, users may be waiting an unacceptable amount of time for their operation to complete. All the while, depending on its implementation, the database may be locked to other reads or writes during the query.

Ensuring consistency and reliability in a globally distributed database is certainly a hard problem. Paramount to this guarantee is creating some way to track each transaction and when it occurred. Clocks can be uncertain, and are something that is often overlooked in computer hardware. The new database solution must take this into account when addressing the issue of consistency.

Before this paper was published, similar database systems in use were Bigtable and Megastore. The former was a key-value store, and users complained about its difficulty handling complex requests, as well as consistency issues when data was distributed nationally and/or globally (Corbett et al. 251). The latter was semi-relational instead of key-value, did support distribution and replication, but suffered from performance issues. With the existing solutions, users were forced to choose between performance or reliability.

3 Techniques and New Solutions

Spanner combines the best elements of Bigtable and Megastore to create a globally-distributed database. From Bigtable: Zones, the unit of physical isolation. Tablets, the data structures, are also similar. From Megastore: Semi-relational tables and synchronous replication. Corbett et al. implement additional features to Spanner as well, the most notable being timestamps. All data is versioned and stamped with each commit time. This allows users to read current as well as historical data based on a query including some bound for commit times. These timestamps are managed via the TrueTime API. A function to get the current time will return epoch time, in an interval, with bounded time uncertainty, such that the interval is guaranteed to contain the time in which the function was called. TrueTime runs on time master machines in datacenters, each with GPS or atomic clocks to track time. If the machine detects too much drift, it will remove itself from the pool of time masters. Timestamps are assigned in various ways to different types of transactions, but the key takeaway is that "these features enable, for example, the guarantee that a whole-database audit read at a timestamp t will see exactly the effects of every transaction that has committed as of t" (Corbett et al. 256).

The proposed solution is better than the state-of-the-art solutions. In fact, it combines their best features. Spanner brings significant performance gains over Megastore, despite being semi-relational as well. At the same time, Spanner uses data structures similar to Bigtable, but successfully brings wide area network consistency to the table.

The authors bring in benchmarks and a case study to demonstrate Spanner's merit. Most tables showed times for some operations and conditions, but it did not provide the best example of performance. The availability graph (Figure 5) did give a useful visualization of how the system will recover when a leader goes down. Their case study, using Google's ad backend, F1, was presented as a before and after comparison. Resharding had to be done manually, failure was risky, and most systems were not fit to be used in that environment where strong semantics are required. F1 began using Spanner due to its automatic resharding, automatic failover, and semantics. One of the most notable testimonials about Spanner remarked that when there were cluster failures, it was "nearly invisible" to the client and had very little impact on them (Corbett et al. 261). If a user does not know anything is wrong and things perform as usual, that is an excellent argument in favor of Spanner.

There are some limitations to Spanner. Reads cannot be done in parallel, however the authors are searching for a solution. TrueTime's ϵ, at higher values, can affect performance. Complex SQL queries also impact performance as they were designed for key-value accesses, but Spanner is not using key-value stores. Data load balancing is also not implemented.

4 Suggestions

Section 4 went very heavy on how timestamps worked, which is useful from a very low-level perspective, but the paper is meant to detail Spanner as a whole. The authors seemed to be very proud of the TrueTime implementation, which may explain why they went into so much detail, but it felt out of place for a general overview of the database.