Spanner

Background

Spanner is a database created by Google that falls under the category of NewSQL - it uses new designs over the same familiar relational data model of SQL in order to greatly increase its performance.

Spanner mainly relies on using synchronized clocks via their service known as TrueTime! While there are other similar databases to Spanner, such as CockroachDB (which basically is the same), its lack of ability to ensure that all servers have closely synchronized clocks makes the solution a bit less viable.

Spanner Guarantees

- ACID transactions
  - Atomicity
    - Achieved by two phase commit across shards
  - Serializability
    - Achieved by two phase locking on a single node (Spanner optimizes on this!)
- Replicas are kept consistent via a consensus algorithm
- Linearizable reads and writes

All while maintaining great performance!

The Issue With Two Phase Locking

Recall: In two phase locking, a transaction may hold the lock on an object in either shared or exclusive mode.

- If T1 holds lock in exclusive mode, T2 cannot read it until T1 lets go
- If T2 holds lock in shared mode, T1 cannot write it until T2 lets go

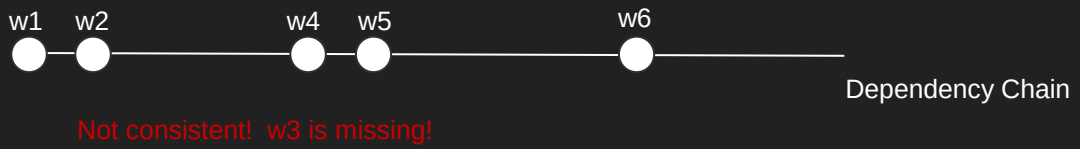
Spanner makes it so that reads do not require grabbing locks, huge performance gains! Previously, reads returning lots of data such as full database scans would require grabbing tons of locks, basically freezes writing to the database.

Consistent Snapshots

A consistent snapshot is consistent with causality: if a snapshot contains a write, all writes that the writer had read in order to make its own write must be present.

Consistent Snapshots

A consistent snapshot is consistent with causality: if a snapshot contains a write, all writes that the writer had read in order to make its own write must be present.



Consistent Snapshots

In order to track causality, Spanner uses actual timestamps! Since they believe they can rely on their server's clocks, any read only transaction will only read writes with a lower timestamp than it.

A read only transaction that starts at 2:34 and finishes at 2:36 will not read w2:35 because that has a greater timestamp!

Why not Lamport Timestamps?

Recall: Lamport Timestamps are a way of creating a total ordering of writes in a distributed system consistent with causality. But do they work here?

Lamport Timestamps will leave out many writes that are “concurrent”, but still did not happen at the same time.

It may be the case that we have two writes, W1 and W2, where the W1 < W2 (because they are concurrent and use an arbitrary node ordering to decide order) and W2 happened an hour earlier than W1 - hence Lamport Timestamps are not linearizable.

Additionally, end users cannot communicate with one another externally!

E.g. Jordan sees a cooking video on his friend’s TikTok for you page, immediately goes to follow the account. There is no way for TikTok to know analytically that Jordan had first been influenced by the for you page.

Why not Centralized Timestamps?

Could use a single server or cluster of servers to give each write to the system an ordering.

However, this is a huge bottleneck, as the ordering server could fail, and adds a ton of network latency for users not located geographically close to the servers.

TrueTime

Does not return a single timestamp, but instead an uncertainty interval [t_min, t_max], in which it takes into account multiple sources of error (such as network round trip time, quartz drift, GPS receiver accuracy. This range ensures a very high probability that the actual time is within it.

One write is considered to precede another if its maximum time is lower than the minimum time of the other.

TrueTime

Recall: Clocks in distributed system are not perfectly in sync, even if they are frequently updated using NTP!

Spanner Writes

- Every single write to spanner has to wait a time period equal to the size of its uncertainty interval
  - So imagine we have W1 occurs at [8, 10] - W1 must wait two seconds before committing
  - This ensures that no matter where W1 actually was in the interval, it commits after time 10
  - If any transaction R1 reads W1, this must happen after time 10
  - So R1 must have interval value [x, y such that y is greater than 10]
  - Using the latest timestamps, R1 comes after W1
- If we have to wait the uncertainty interval every write, we want to minimize the uncertainty interval
  - Google does this by putting atomic clocks in every data center and synchronizing with them frequently!

Conclusion

Spanner is an extremely interesting example of using non-commodity hardware in order to achieve very high throughput using a relational data model and achieving linearizability. Although it probably doesn’t make sense for most businesses, simply due to the lack of ability to run it on generic hardware, it is an interesting study on the lack of synchronized clocks in distributed systems and what could be accomplished if we could synchronize them.