**Paper Review [Spanner Google’s Globally-Distributed Database](https://xduan7.com/2016/02/19/paper-review-spanner-googles-globally-distributed-database/)**

**Summary:** Spanner is Google's scalable, multi-version, synchronously-replicated and globally-distributed database. Spanner uses non-blocking reads in the past, along with lock-free read-only transactions plus atomic schema changes. It uses a novel TrueTime API to support distributed transactions at global scale.

**Introduction:** Systems like Bigtable has been reported difficult to use for some types of applications that have complex and evolving schemas and those that require strong consistency in presence of wide-area replications. Spanner is a database that fragments and spreads data across many sets of Paxos state machines in data centers spread across the world. It automatically redoes the fragmenting and spreading of data across machines as the amount of data or the number of servers changes.

**Literature Review:** The motivation for Spanner came from Megastore that has been used by many Google applications like Gmail, Calendar, Picasa etc. Spanner inherits the use of semi-rational tables and synchronous replication from Megastore that already have the implementations despite of it’s bad write throughput. Spanner evolved from a Bigtable-like versioned key-value store into a temporal multi-version database. Data is stored in semi-relational tables and Spanner provides a SQL-based query language and supports general-purpose long-lived transactions. The authors believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions.

**Implementation:** Spanner provides externally consistent reads and writes, and globally-consistent reads across the databases at a timestamp which is enabled by having a globally meaningful commit timestamps to distributed transactions. This key enabler of this timestamps is a new TrueTime API and its implementation. The API directly exposes clock uncertainty and the guarantees on Spanner's timestamps depend on the bounds that the implementation provides. If the uncertainty is large, Spanner slows down to wait out the uncertainty. Uncertainty is typically less than 10ms by using GPS and atomic clocks as references.

A spanner deployment is called a universe. Spanner is organized as a set of zones which is the unit of administrative deployment. Each zone has a zone master and a hundred to several thousand span servers. Zone masters assign data to span servers where span servers serve data to clients. A universe also has a universe master and a placement driver, where the latter moves data across zones.

Data is organized as directories or more accurately buckets. It's a contiguous key that share a common prefix. A directory is a unit of data placement. Spanner supports SQL like queries plus some extension to support protocol-buffer-valued fields. Every table is required to have an ordered set of one or more primary-key columns.

**Results:** Spanner combines and extends ideas from the database community, a familiar, easy-to-use, semi-relational interface, transactions, and an SQL-based query language; from the systems community, scalability, automatic sharding, fault tolerance, consistent replication, external consistency, and wide-area distribution.

**Discussion:** One thing that seems missing from this paper is whether Spanner can perform more advanced relational database operations such as aggregations, subqueries or joins. Usually performing these operations in a distributed system requires some extra component to store the intermediate values, which was not mentioned in the paper.

**Conclusion:** The authors successfully developed Spanner to address some limitations with other solutions developed in the past: Big Table, which lacks schema and strong consistency; Megastore, which has poor write performance; and a sharded MySQL, which required the application to know about the sharding schema and resharding being a risky and lengthy process. Moreover, authors made an intelligent use of TruTime API ensuring time guarantees.