Spanner: Google’s Globally-Distributed Database

Tuochao Chen, Tongyan Wang
What is Spanner

Spanner is Google’s scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support *externally-consistent* (provides clients with the strictest concurrency-control guarantees for transactions) distributed transactions.
Why Spanner?

Pros & Cons of Bigtable & Megastore:

**Bigtable:**
Pros: It supports high read and write throughput at low latency
Cons: asynchronous when performing cross-data center replication, thus only achieve eventual consistency; more like a key-value storage thus can be difficult to use.

**Megastore:**
Pros: support schemas and provides a SQL-based query language.
Cons: suffers from relatively poor write throughput.
Advantages of Spanner

1. Supports general-purpose transactions, and provides a SQL-based query language; Data is stored in schematized semi-relational tables.
2. Each transaction is automatically timestamped with its commit time. Provides externally consistent reads and writes, and globally-consistent reads across the database at a time-stamp.
Data Model of Spanner

“Each database can contain an unlimited number of schematized tables...”
“Spanner’s data model is not purely relational. More precisely, every table is required to have an ordered set of one or more primary-key Columns.”

<table>
<thead>
<tr>
<th>key</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>goog</td>
<td>Google</td>
<td>20</td>
</tr>
<tr>
<td>fb</td>
<td>Facebook</td>
<td>14</td>
</tr>
</tbody>
</table>

Google Spanner

```
Goolge Spanner
```

Google Bigtable

```
"goog": {name: "Google" },
"fb" : {age: 14, location: "Menlo Park, CA"},
"amzn": {url: "www.amazon.com"}
```

Bigtable (schemaless) vs Spanner
Organization

- A Spanner deployment is called a *universe*. Each Spanner is separate for different *universe*.

Figure 1: Spanner server organization
Organization

- Spanner is organized as a set of *zones*, which are the unit of administrative deployment. The set of zones is also the set of locations across which data can be replicated.

Figure 1: Spanner server organization
Organization

- A zone has one *zonemaster* and between one hundred and several thousand *spanservers*.

*Zonemaster* assigns data to spanservers while *spanserver* serve data to clients.

Figure 1: Spanner server organization
Organization

- The per-zone *location proxies* are used by clients to locate the spanservers assigned to serve their data.

Figure 1: Spanner server organization
Organization

- The *universe master* is primarily a console that displays status information about all the zones for interactive debugging.
- The placement driver handles automated movement of data across zones on the timescale of minutes.

![Figure 1: Spanner server organization](image)
Spanserver Software Stack

The main components of distributed storage system are familiar:

- 2 Phase Commit
- Paxos and Replica

Figure 2: Spanserver software stack
Spanserver Software Stack – Bottom Part

- Each Replica is responsible for one tablet and Colossus, which is used for data storage.
- Leader: efficiently execute Paxos protocol.
- The set of parts that are responsible for the same dataset is called a paxos group.

Figure 2: Spanserver software stack
Spanserver Software Stack – Top Part

- Lock table:
  Control read-write access

- Transition manager:
  Responsible for the data exchange between groups

- Participant Leader:
  In charge of 2 phase commit and ensure consistency of transaction.

Figure 2: Spanserver software stack
Directories and Placement

A directory is the unit of data placement. All data in a directory has the same replication configuration. When data is moved between Paxos groups, it is moved directory by directory.

Pros and cons of directory movement:

1. Balance load between different paxos group
2. Lower latency of data read & write by moving directory to a paxos group that are closer to the client.

Figure 3: Directories are the unit of data movement between Paxos groups.
TrueTime API

TT.now() = [earliest, latest]
TrueTime API

Achieved by GPS and atomic clock

(uncertainty < 10ms)
Assign Timestamp

Given transaction, Spanner assigns it the timestamp that Paxos assigns to the Paxos write that represents the transaction commit.
External Consistency Requirement

If a transaction $e_1$ commits before another transaction $e_2$ starts, then $e_1$’s commit timestamp is smaller than $e_2$’s:

$s_1 \rightarrow e_1$

$s_2 \rightarrow e_2$

$t_{\text{abs}}(e_1_{\text{commit}}) < t_{\text{abs}}(e_2_{\text{start}}) \rightarrow s_1 < s_2$
External Consistency Rules

\[ t_{abs}(e1_{commit}) < t_{abs}(e2_{start}) \implies s1 < s2 \]

**Start:** When coordinate leader recv commit request (ei_server), assign si to ei with si \( > \) TT.now().latest

**Commit Wait:** After si is assigned, commit the transaction (ei_commit) when TT.after(si) comes true.
External Consistency Achievement

\[ t_{\text{abs}}(e_1_{\text{commit}}) < t_{\text{abs}}(e_2_{\text{start}}) \rightarrow s_1 < s_2 \]

Coordinate leader:

- \( e_1_{\text{commit}} \)
- \( e_2_{\text{start}} \)
- \( e_2_{\text{server}} \)
- \( e_2_{\text{commit}} \)

Commit Wait

Start

Commit Wait

World time
Monotonicity Invariant

A leader must only assign timestamps within the interval of its leader lease.

Abdicating Leader_i

World time

s_max

TT.after(s_max)
Concurrency control

if a transaction $T_1$ commits before another transaction $T_2$ starts, then $T_1$’s commit timestamp is smaller than $T_2$’s.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Timestamp Discussion</th>
<th>Concurrency Control</th>
<th>Replica Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Write Transaction</td>
<td>§ 4.1.2</td>
<td>pessimistic</td>
<td>leader</td>
</tr>
<tr>
<td>Snapshot Transaction</td>
<td>§ 4.1.4</td>
<td>lock-free</td>
<td>leader for timestamp; any for read, subject to § 4.1.3</td>
</tr>
<tr>
<td>Snapshot Read, client-chosen timestamp</td>
<td>—</td>
<td>lock-free</td>
<td>any, subject to § 4.1.3</td>
</tr>
<tr>
<td>Snapshot Read, client-chosen bound</td>
<td>§ 4.1.3</td>
<td>lock-free</td>
<td>any, subject to § 4.1.3</td>
</tr>
</tbody>
</table>
Read-Write Transaction:

Client

Participant
leader
Paxos Group

Participant
leader
Paxos Group
Read-Write Transaction:

- Client
  - Reader Request
    - Participant
      - Leader
        - Paxos Group
    - Participant
      - Leader
        - Paxos Group
    - Participant
      - Leader
        - Paxos Group
    …
Read-Write Transaction:

Client

Participant leader
Paxos Group

Participant leader
Paxos Group

......

Participant leader
Paxos Group

Recent data
Read-Write Transaction:

- Client
- Participant
- Leader

Paxos Group

**2PC begins**

- Write commit Query
- Coordinating leader

- Participant
- Leader
- Paxos Group

- Participant
- Leader
- Paxos Group
Read-Write Transaction:

2PC begins

Client

Coordinare leader

Prepare reply (t_prepare)

Participant

leader

Participant

leader

Paxos Group

Paxos Group
Read-Write Transaction:

Assign timestamp and commit

\[ s > \max \left( \max_g \left( t^g_{\text{prepare}} \right), \text{TT.now().latest} \right) \]
Read-Write Transaction:

Client

Coordinare leader

Write commit s

Participant

leader

Paxos Group

Write commit s

Participant

leader

Paxos Group
Thank you