Cloud Spanner
A Globally-Distributed Database by Google

Presented by Zoey Xinyi Ge
Section 1: Overview & Usage
Overview

• Relational Database with support for SQL semantics
  • ACID Transactions & External Consistency (explained later)

• High Availability
  • Globally distributed
  • Synchronous Replication (instant failover)

• Horizontal Scaling
  • **Horizontal scaling**: by adding more machines to your pool of resources (also described as “scaling out”)
  • Vertical scaling: by adding more power (e.g. CPU, RAM) to an existing machine (also described as “scaling up”).
## Overview

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cloud Spanner</th>
<th>Traditional Relational</th>
<th>Traditional Non-Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Consistency</td>
<td>Strong</td>
<td>Strong</td>
<td>Eventual</td>
</tr>
<tr>
<td>Availability</td>
<td>High</td>
<td>Failover</td>
<td>High</td>
</tr>
<tr>
<td>Scalability</td>
<td>Horizontal</td>
<td>Vertical</td>
<td>Horizontal</td>
</tr>
<tr>
<td>Replication</td>
<td>Automatic</td>
<td>Configurable</td>
<td>Configurable</td>
</tr>
</tbody>
</table>
Use Cases

Others usage:
• RDBMS + Scale
• User metadata
• Ads / Marketing

• Inventory management
• Financial transactions (OLTP)
• Game States
• ....
Usage

- Publicly available product
- Supports multiple programming languages
- Easy to use

```python
# Imports the Google Cloud Client Library.
from google.cloud import spanner

# Instantiate a client.
spanner_client = spanner.Client()

# Your Cloud Spanner instance ID.
instance_id = 'my-instance-id'

# Get a Cloud Spanner instance by ID.
instance = spanner_client.instance(instance_id)

# Your Cloud Spanner database ID.
database_id = 'my-database-id'

# Get a Cloud Spanner database by ID.
database = instance.database(database_id)

# Execute a simple SQL statement.
results = database.execute_sql('SELECT 1')

for row in results:
    print(row)
```
Usage: SQL

• Supports SQL queries (ANSI 2011 with extensions)

• Clients use **INTERLEAVE IN** to declare hierarchy and group relevant data for better locality and performance

```sql
CREATE TABLE Users {
    uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
    aid INT64 NOT NULL, aid2 INT64 NOT NULL,
    name STRING
} PRIMARY KEY (uid, aid),
INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

![Diagram showing example Spanner schema for photo metadata, and the interleaving implied by INTERLEAVE IN.](image)

Figure 4: Example Spanner schema for photo metadata, and the interleaving implied by **INTERLEAVE IN**.
Section 2: Abstractions
Abstractions : Directory

- A directory contains contiguous keys that share a common prefix
- Smallest unit of data placement and movement
- Smallest unit of geo-replication configuration by applications
Abstractions : Directory

Data placement & movement example:
Abstractions: Tablet

- Directories in the same tablet are replicated together.
- Spanner assigns timestamps to data.
- Data are multi-versioned.
Abstractions: Spanserver

- Each spanserver is responsible for between 100 and 1000 tablets
Abstractions: Zone
Section 3: Replication
Replication

Colossus:
Distributed file system
Derived from GFS.
Stores the tablet’s state and the write-ahead log.
Replication

Paxos Group:
Several paxos state machines that work on the same tablet across different replicas.
Replication

**Lock table:**
Each spanserver has one lock table. It maps ranges of keys to locking states.
(Eg. Lock a range of keys when they are undergoing a write operation)
Replication

Transaction manager & Participant leader:
When a transaction involves multiple paxos groups, the participant leaders do **2-phase commit** to maintain consistency.
2-Phase Commit across paxos groups
Summary for Replication

Three main techniques are used to keep consistency:

1. Paxos across different locations for each tablet

2. Lock tables for concurrent access of same keys

3. Two-phase commit for transactions that involve multiple paxos-groups (i.e. different tablets)
Section 4: Correctness & More details
Building Block: TrueTime API

- `TTinterval` is guaranteed to contain the absolute time during which `TT.now()` is invoked

- Calculated from instantaneous error bound (in production, typically 1ms – 7ms)

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TT.now()</code></td>
<td><code>TTinterval: [earliest, latest]</code></td>
</tr>
<tr>
<td><code>TT.after(t)</code></td>
<td>true if <code>t</code> has definitely passed</td>
</tr>
<tr>
<td><code>TT.before(t)</code></td>
<td>true if <code>t</code> has definitely not arrived</td>
</tr>
</tbody>
</table>
TrueTime API

Time Master Machines:
- GPS time calibration
- *plus* Atomic Clock
- *plus* Check with local machine clock

Daemons (non-masters):
- Polls Time Masters from different data centers
Requirements of the system

1. Monotonicity

2. External Consistency
Realization of Monotonicity

Within each Paxos group, Spanner assigns timestamps to Paxos writes in monotonically increasing order, even across leaders.

- With one single leader replica: trivially increase timestamps
- With changed leaders
  - Each leader is designed to have long-lived lease (10 sec)
  - Leader lease intervals are disjoint (implemented with TrueTime; details skipped)
  - A leader must only assign timestamps within the interval of its leader lease
Realization of External Consistency

If the start of a transaction T2 occurs after the commit of a transaction T1, then the commit timestamp of T2 must be greater than the commit timestamp of T1. (Across tablets)

• Rule 1: “Start”
  • The coordinator leader (in 2PC) assigns a commit timestamp $s_i$ less than the value of TT.now().latest after receiving the commit request T.

• Rule 2: “Commit Wait”
  • The coordinator leader (in 2PC) ensures that clients cannot see any data committed by T until TT.after($s_i$) is true.
Realization of External Consistency

• Prove external consistency, that is:  \( t_{abs}(e_{1}^{commit}) < t_{abs}(e_{2}^{start}) \Rightarrow s_1 < s_2 \)

1. Timestamp 1 is prior to T1’s actual commit
2. T1’s commit is prior to T2’s start
3. T2’s start is prior to T2’s arrival at server
4. T2’s arrival at server is prior to timestamp 2
5. Timestamp 1 is prior to Timestamp 2
Types of Transactions Supported

• Read-write transactions

• Read-only transactions
  • Must be predeclared as not having any writes
  • Executed at a system-chosen timestamp
  • No locking, so incoming writes are not blocked

• Snapshot reads
  • Client specifies a timestamp, or specify an upper bound on staleness
  • No Locking, no blocking

On Any replica that is sufficiently up-to-date
Read-Write Transactions
Commit Wait and 2-Phase Commit

- Acquired locks
- Start logging
- Done logging
- Release locks
- Committed
- Notify participants of s
- Released locks
- Computed s for each
- Prepare s
- Send s
- Commit wait done
- Compute overall s
Read-Only Transactions / Snapshots

0. Define the scope (ranges of keys involved)

Phase 1: assign a timestamp $s_{\text{read}}$

- If only one paxos group involved:
  - timestamp of the last committed write at a Paxos group
- If multiple paxos groups involved:
  - $S_{\text{read}} = \text{TT.now().latest}$ (might need to wait for ongoing paxos writes and 2PC to complete)

Phase 2: serve the read

- Lock-free and non-blocking read at one server
- Can serve as long as the timestamp $s_{\text{read}} \leq t_{\text{safe}}$
When is it safe to serve a read?

\[ t_{safe} = \min(t_{safe}^{Paxos}, t_{safe}^{TM}) \]

- \( t_{safe}^{Paxos} \) last Paxos write
- \( t_{safe}^{TM} \) \( \infty \) if no transaction going through 2PC; 
  else \( \min_i(s_{i,g}^{prepare}) - 1 \)
Atomic schema change is supported in Spanner.

- **Challenge:** Could involve millions of paxos groups
- **Solution:**
  1. Predetermine a future timestamp $t$ that the change will be executed. Register it in the prepare phase.
  2. Reads and writes only proceeds if their timestamps precede $t$
Section 5: Evaluation
Latency & Throughput (Microbenchmarks)

<table>
<thead>
<tr>
<th>replicas</th>
<th>latency (ms)</th>
<th>throughput (Kops/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>write</td>
<td>read-only transaction</td>
</tr>
<tr>
<td>1D</td>
<td>9.4±.6</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>14.4±1.0</td>
<td>1.4±.1</td>
</tr>
<tr>
<td>3</td>
<td>13.9±.6</td>
<td>1.3±.1</td>
</tr>
<tr>
<td>5</td>
<td>14.4±.4</td>
<td>1.4±.05</td>
</tr>
</tbody>
</table>

Table 3: Operation microbenchmarks. Mean and standard deviation over 10 runs. 1D means one replica with commit wait disabled.
Scaling

- Scaling the number of participants (tablets involved in a transaction)

<table>
<thead>
<tr>
<th>participants</th>
<th>latency (ms)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>99th percentile</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>17.0 ±1.4</td>
<td>75.0 ±34.9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>24.5 ±2.5</td>
<td>87.6 ±35.9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>31.5 ±6.2</td>
<td>104.5 ±52.2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30.0 ±3.7</td>
<td>95.6 ±25.4</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>35.5 ±5.6</td>
<td>100.4 ±42.7</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>42.7 ±4.1</td>
<td>93.7 ±22.9</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>71.4 ±7.6</td>
<td>131.2 ±17.6</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>150.5 ±11.0</td>
<td>320.3 ±35.1</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Two-phase commit scalability. Mean and standard deviations over 10 runs.
Availability

- Servers in 5 zones, 25 spanservers each
- All leaders placed in Zone 1
- Non-leader: Kill all servers in Zone 2
- Leader-hard: Kill all servers in Zone 1
- Leader-soft: Kill all servers in Zone 1 but with prior notifications to all servers they should handoff leadership first

Figure 5: Effect of killing servers on throughput.
TrueTime Error Bound

Figure 6: Distribution of TrueTime $\epsilon$ values, sampled right after timeslave daemon polls the time masters. 90th, 99th, and 99.9th percentiles are graphed.
Q & A
References


• [https://www.youtube.com/watch?v=C75kpQszAjs](https://www.youtube.com/watch?v=C75kpQszAjs)

• [https://www.youtube.com/watch?v=lfsTINNCooY&t=566s](https://www.youtube.com/watch?v=lfsTINNCooY&t=566s)