Spanner

Aman Sharma, Cyrus Nikolaidis
Spanner interface

• Distributed database that supports transactions

<table>
<thead>
<tr>
<th>UMID</th>
<th>name</th>
<th>age</th>
<th>standing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Spread across data centers (in contrast to BigTable)
Properties

• External Consistency
• Globally distributed
• Previously considered impossible

If a transaction T1 commits before another transaction T2 starts, then T1’s commit timestamp is smaller than T2’s.
TOO MUCH LATENCY
TOO MUCH LATENCY
How Spanner Tackles this

• Every transaction has a commit timestamp
• When a row is modified, a new version of it is created
How Spanner Tackles this

• Every transaction has a commit timestamp
• When a row is modified, a new version of it is created
How Spanner Tackles this

• Every transaction has a commit timestamp
• When a row is modified, a new version of it is created
How Spanner Tackles this

• Every transaction has a commit timestamp
• When a row is modified, a new version of it is created
• Read-only transaction reads latest version of data committed before its timestamp.
• The commit timestamps should monotonically increase for us to be able to ensure linearizability
Synchronizing clocks

• There are some timemasters in each datacenter, and each server has a timeslave daemon that synchronizes with each machine every 30 seconds.
Uncertainty

Assume worst case drift of 200 μs/s
TrueTime API

- TT.now() returns a range [earliest, latest]
- The actual current time is somewhere in this range
TrueTime API

• s is the commit timestamp for a transaction
• We assign s so that s is before the absolute time at which the commit occurs.
Linearizability

T1 Start

Commit Wait

Wait till s1 < TT.now().earliest

Pick s1 = TT.now().latest

T1 Commit

T2 Start

Pick s2 = TT.now().latest
We can serve a read if its timestamp is less than the most recent applied write.
End Result

• Each transaction (read-only or read/write) is assigned a timestamp

• Timestamps are externally consistent and imply a strict serial order between transactions that touch the same shard

• A read will have the same result at all replicas, without a lock
Writing to a shard

- Each shard is a paxos group that has a leader.
- Leader responsible for keeping lock table.
- Important for efficiency of managing lock table leader is long lived.

- **Solution: Leader Leases!**
Leader Leases

- Purpose: Create long-lived leaders
- Uses timed leases (10 seconds by default)
- Potential leader sends requests for time lease votes
- Upon receiving a quorum of lease votes, the leader knows it has a lease
- For each Paxos group, each leader’s lease interval is disjoint from every other leader’s
Schema change transactions

• Very large number of participants
• Infeasible to use standard transactions
• Explicitly assigned a timestamp in the future
## Evaluation - Latency and Throughput

<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±6.6</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>14.4±1.0</td>
<td>1.4±1.0</td>
</tr>
<tr>
<td>3</td>
<td>13.9±6.6</td>
<td>1.3±1.0</td>
</tr>
<tr>
<td>5</td>
<td>14.4±4.0</td>
<td>1.4±0.5</td>
</tr>
</tbody>
</table>

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

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

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

Figure 5: Effect of killing servers on throughput.
Figure 6: Distribution of TrueTime $\epsilon$ values, sampled right after timeslave daemon polls the time masters. 90th, 99th, and 99.9th percentiles are graphed.