Spanner: Google’s Globally-Distributed Database

James C. Corbett, Jeffrey Dean, et. al.
Google, Inc.
OSDI 2012 and TOCS 2013

Presented by: M. Safdar Iqbal
Problem statement

• Build a globally-distributed database that supports consistent distributed transactions
Outline

• Motivation
• Spanner architecture
• TrueTime
• Transactional support
• Experiments
Challenges

- DBMS ensure consistency
  - Any read sees all effects of all writes before it

- Scalability is a challenge
  - Traditional DBMS solution is pay up or go home
  - Disclaimer: This is changing rapidly in recent years
Motivation: Google’s earlier solutions

- BigTable – Google’s distributed key-value store
  - Eventually consistent

- Megastore – SQL-like joins on top of BigTable
  - Slow write throughput
Solution: Spanner

• Spanner is
  • Externally consistent
  • Globally-distributed
  • Provides SQL-like data motel
Outline

• Motivation
• Spanner architecture
• TrueTime
• Transactional support
• Experiments
Spanner architecture

- universe master
- placement driver

Zone 1
- zonemaster
- location proxy
- spanserver

Zone 2
- zonemaster
- location proxy
- spanserver

Zone N
- zonemaster
- location proxy
- spanserver
Spanservers

- Spanserver maintains data and serves client requests
- Data are key-value pairs
  \((\text{key: string, timestamp: int64}) \rightarrow \text{string}\)
- Data is replicated across spanservers (could be in different datacenters) in the unit of tablets
- SQL-like data model is also supported
Consistent replication via Paxos

- Spanner uses Paxos To maintain consistency between tablet replicas
- Spanner maintains a Paxos state machine per tablet per spanserver
- Paxos group: the set of all replicas of a tablet
Paxos

• Paxos is a consensus protocol

Consider a system of $n$ participants

• Each participant can send message to each other to exchange some state

• Participant states must be consistent for each one to have a consistent view of system state
Paxos

- Participants elect a *leader*
- Leader is responsible for achieving the consensus
- A *majority* of participants have to agree on a state for it be “chosen” as consistent

- Paxos maintains *consistency* while maintaining availability
Spanserver architecture
Transaction manager

- Transaction manager (TM) runs on every Paxos leader
- Paxos leader becomes a participant leader
- All the replicas become participants in the transactions
- Transactions involving just one Paxos group are not handled by the TM
Outline

• Motivation
• Spanner architecture
• TrueTime
• Transactional support
• Experiments
TrueTime

- API for syncing timestamps and time intervals across global data centers
- Exposes clock uncertainty to application
- $TT.now.earliest \leq t_{abs}(now) \leq TT.now.latest$

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

• Time references
  • GPS
    • Antenna/receiver faults
    • Radio interference
    • System outages
  • Atomic clocks
    • Clock drift

• Atomic clock failures uncorrelated to GPS failures and vice versa
TrueTime: implementation

- **Time masters** in each data center
  - Equipped with GPS or atomic clocks (*Armageddon* masters)
  - Sync time with each other
  - Advertise an *uncertainty* during syncs - based on worst-case clock drift

- **Timeslave daemon** on every machine
  - Polls the time masters in nearby *and* farther datacenters
  - Time uncertainty $\varepsilon$ is derived from local clock drift, time-master uncertainty and communication delays
  - $\varepsilon$ is a sawtooth; 1 to 7 ms (6 ms from drift, 1 ms from delays)
Outline

• Motivation
• Spanner architecture
• TrueTime
• Transactional support
• Experiments
Concurrency control

- Enabled by TrueTime – leads to global consistency

- Supports the following features
  - Externally consistent transactions
  - Lock-free read-only transactions
  - Non-blocking snapshot reads

- Snapshot reads are “reads from the past”
  - Client can provide timestamp
  - Client can provide a bound on staleness
## Transactions in Spanner

<table>
<thead>
<tr>
<th>Operation</th>
<th>Concurrency control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-write transaction</td>
<td>Two-phase</td>
</tr>
<tr>
<td>Read-only transaction</td>
<td>Lock-free</td>
</tr>
<tr>
<td>Snapshot read</td>
<td>Lock-free</td>
</tr>
</tbody>
</table>
Two-phase commit

- Transaction coordinator sends ‘Prepare’ messages to all replicas
- Commit cannot take place unless all replicas reply ‘YES’ to the prepare message
Paxos leader leases

- Spanserver sends request for *timed* lease votes
- Leadership is granted when it receives acknowledgements from a *quorum*
- Lease is extended on successful writes
- Disjoint leases are invariant within the same Paxos group
Read-write transactions

• Each transaction must assigned a timestamp

• Time-stamp invariants

  1. Timestamps must be assigned in monotonically increasing order.
     • Leader must only assign timestamps within the interval of its leader lease.

  2. If transaction $T_1$ commits before $T_2$ starts, $T_2$'s timestamp must be greater than $T_1$'s
     • External consistency
Read-write transactions

- Two-phase commit (cross-group transactions)
- Participant leaders choose prepare timestamps and send prepare messages through Paxos to the coordinator
- Coordinator assigns a commit timestamp $s_i$ no less than all prepare timestamps and $TT.now().latest$ (computed when receiving the request)
- Coordinator ensures that clients cannot see any data committed by $T_i$ until $TT.after(s_i)$ is true (this is done by waiting until absolute time $> s_i$ to commit)
Snapshot read transaction

- Safe time: a timestamp at which the replica is up-to-date
- Replicas are not up-to-date if they in the prepare phase or in-between prepare and commit phases
- Each replica tracks a safe time $t_{safe}^{Paxos}$
- Each participant leader has a safe time $t_{safe}^{TM}$
- To read snapshot at $t$, $t \leq \min(t_{safe}^{Paxos}, t_{safe}^{TM})$
Read-only transactions

• Leader assigns a timestamp to the read operation (derived from $TT.now.latest$)

• Then it does a snapshot read on any replica

• External consistency requires the read to see all transactions committed before the read starts - timestamp of the read must be no less than that of any committed writes
Outline

• Motivation
• Spanner architecture
• TrueTime
• Transactional support
• Experiments
Microbenchmarks

- Measure latency and throughput read-write, read-only and snapshot transactions (4 KB) individually

<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>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>
Availability

- Replicas manually killed to measure effect on read throughput
Distribution of $\varepsilon$ values
Thank You!