15-712:
Advanced Operating Systems & Distributed Systems

Spanner

Prof. Phillip Gibbons

Spring 2020, Lecture 22
“Spanner: Google’s Globally-Distributed Database”


• James Corbett (Google)

• Jeff Dean (Head, Google Brain) – NAE, ACM Fellow, AAAS Fellow, Mark Weiser Award

• Dale Woodford (Google)
Database vs. Key-value Store

Bigtable [OSDI’06 best paper] was Google’s big thing for scalability

key-value store
eventual consistency

(continuing a long line of papers disparaging databases)

But...

“We consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications.”

“Why we built a database as opposed to a key-value store: We want to make it easy for programmers at Google to build their applications.” [from OSDI’12 talk]
Spanner

- Worked on Spanner for 4½ years at time of OSDI’12

- Scalable, multi-version, globally-distributed, synchronously-replicated database
  - Hundreds of datacenters, millions of machines, trillions of rows

- Transaction Properties
  - Transactions are externally-consistent (a.k.a. Linearizable)
  - Read-only transactions are lock-free & globally-consistent
  - Read-write transactions use 2-phase-locking

- Flexible replication configuration
  - Which datacenters contain which data, how far data are from their users, how far replicas from each other, how many replicas
Spanner: Google’s Globally-Distributed Database

Wilson Hsieh
representing a host of authors
OSDI 2012
What is Spanner?

- Distributed multiversion database
  - General-purpose transactions (ACID)
  - SQL query language
  - Schematized tables
  - Semi-relational data model

- Running in production
  - Storage for Google’s ad data
  - Replaced a sharded MySQL database
Example: Social Network

US
- San Francisco
- Seattle
- Arizona

x1000

Brazil
- Sao Paulo
- Santiago
- Buenos Aires

x1000

Spain
- London
- Paris
- Berlin
- Madrid
- Lisbon

x1000

Russia
- Moscow
- Berlin
- Krakow

x1000
Overview

• Feature: Lock-free distributed read transactions
• Property: External consistency of distributed transactions
  – First system at global scale
• Implementation: Integration of concurrency control, replication, and 2PC
  – Correctness and performance
• Enabling technology: TrueTime
  – Interval-based global time
Read Transactions

- Generate a page of friends’ recent posts
  - Consistent view of friend list and their posts

Why consistency matters
1. Remove untrustworthy person X as friend
2. Post P: “My government is repressive...”
Single Machine

User posts
Friend lists

Block writes

Friend1 post
Friend2 post
...
Friend999 post
Friend1000 post

User posts
Friend lists

Generate my page

OSDI 2012
Multiple Machines

Block writes

Friend1 post
Friend2 post
...
Friend999 post
Friend1000 post

User posts
Friend lists

User posts
Friend lists

Generate my page

OSDI 2012
Version Management

• Transactions that write use strict 2PL
  – Each transaction $T$ is assigned a timestamp $s$
  – Data written by $T$ is timestamped with $s$

<table>
<thead>
<tr>
<th>Time</th>
<th>&lt;8</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>My friends</td>
<td>[X]</td>
<td>[]</td>
<td></td>
</tr>
<tr>
<td>My posts</td>
<td></td>
<td></td>
<td>[P]</td>
</tr>
<tr>
<td>X’s friends</td>
<td>[me]</td>
<td>[]</td>
<td></td>
</tr>
</tbody>
</table>
Synchronizing Snapshots

Global wall-clock time

==

External Consistency:
Commit order respects global wall-time order

==

Timestamp order respects global wall-time order given
timestamp order == commit order
 Timestamps, Global Clock

- Strict two-phase locking for write transactions
- Assign timestamp while locks are held

\[
\text{Pick } s = \text{now()}
\]

Acquired locks  Release locks
Timestamp Invariants

- Timestamp order == commit order

- Timestamp order respects global wall-time order
TrueTime

• “Global wall-clock time” with bounded uncertainty

TT.now() 2*\varepsilon

earliest latest
time

2*\varepsilon
Timestamps and TrueTime

- Acquired locks
  - Pick $s = \text{TT.now().latest}$
- Release locks
  - Wait until $\text{TT.now().earliest} > s$

Commit wait

- Average $\varepsilon$

OSDI 2012
Commit Wait and Replication

Acquired locks

Start consensus

Achieve consensus

Notify slaves

Release locks

Pick s

Commit wait done
Commit Wait and 2-Phase Commit

- Start logging
- Acquired locks
- Release locks
- Committed
- Notify participants of $s$
- Release locks
- Prepared
- Send $s$
- Commit wait done
- Compute overall $s$
- For each
- Compute $s$
- Acquired locks
- Release locks
- Acquired locks

OSDI 2012
Example

Remove X from my friend list
Remove myself from X's friend list

Time | <8 | 8 | 15
--- | --- | --- | ---
My friends | [X] | [] | 
My posts | | | [P]
X's friends | [me] | []
Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.
What Have We Covered?

- Lock-free read transactions across datacenters
- External consistency
- Timestamp assignment
- TrueTime
  - Uncertainty in time can be waited out
What Haven’t We Covered?

• How to read at the present time
• Atomic schema changes
  – Mostly non-blocking
  – Commit in the future
• Non-blocking reads in the past
  – At any sufficiently up-to-date replica
TrueTime Architecture

Datacenter 1

Datacenter 2

... 

Datacenter n

GPS timemaster

GPS timemaster

Atomic-clock timemaster

GPS timemaster

GPS timemaster

GPS timemaster

Client

Compute reference [earliest, latest] = now ± ε
TrueTime implementation

\[ \text{now} = \text{reference now} + \text{local-clock offset} \]

\[ \varepsilon = \text{reference } \varepsilon + \text{worst-case local-clock drift} \]

OSDI 2012
What If a Clock Goes Rogue?

- Timestamp assignment would violate external consistency
- Empirically unlikely based on 1 year of data
  - Bad CPUs 6 times more likely than bad clocks
Network-Induced Uncertainty

Epsilon (ms)

Date
Mar 29 Mar 30 Mar 31 Apr 1
Date (April 13)
1 2 3 4 5 6

99.9
99
90

Mar 29 Mar 30 Mar 31 Apr 1 6AM 8AM 10AM 12PM
What’s in the Literature

• External consistency/linearizability
• Distributed databases
• Concurrency control
• Replication
• Time (NTP, Marzullo)
Future Work

• Improving TrueTime
  – Lower $\varepsilon < 1$ ms

• Building out database features
  – Finish implementing basic features
  – Efficiently support rich query patterns
Conclusions

• Reify clock uncertainty in time APIs
  – Known unknowns are better than unknown unknowns
  – Rethink algorithms to make use of uncertainty

• Stronger semantics are achievable
  – Greater scale != weaker semantics

[End of slides from OSDI’12 talk]
• **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.
Semi-relational Data Model

- Rows must have names
  - Every table must have one or more primary-key columns

- Applications control data locality thru their choice of keys

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

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

• Must declare “read-only” upfront

• Must have “scope” expression
  – Summarize keys that will be read by the entire transaction

Executes at a system-chosen timestamp w/o blocking writes
Availability

Figure 5: Effect of killing servers on throughput.
## F1 Advertising Backend

<table>
<thead>
<tr>
<th>operation</th>
<th>latency (ms)</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>std dev</td>
</tr>
<tr>
<td>all reads</td>
<td>8.7</td>
<td>376.4</td>
</tr>
<tr>
<td>single-site commit</td>
<td>72.3</td>
<td>112.8</td>
</tr>
<tr>
<td>multi-site commit</td>
<td>103.0</td>
<td>52.2</td>
</tr>
</tbody>
</table>

Table 6: F1-perceived operation latencies measured over the course of 24 hours.

---

**Lock Conflicts**
**Only one DC had SSDs**
Friday: Interim Project Reports due

Monday’s Class

“TensorFlow: A System for Large-Scale Machine Learning”

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng 2016

Further Reading:

“MapReduce: Simplified Data Processing on Large Clusters”

Jeffrey Dean, Sanjay Ghemawat 2004