Spanner

Google’s Globally-Distributed Database

A presentation by:

Aneesh Raman, Athanasios Filippidis

For:

CS591 A1 Data Systems Architectures @ Boston University

Spanner authors

Spanner authors

James C. Corbett - UMASS PhD
Lead of Google Storage Testing
Spanner, Megastore

Jeffrey Dean - UW PhD
Lead of Google AI
Spanner, Google Translate, BigTable, MapReduce, LevelDB, TensorFlow, Google Brain

Michael Epstein - Harvard BSc
Lead of Google Cloud Platform
Spanner, BigTable

Agenda

- Spanner: A Distributed Systems and a Databases perspective
- Spanner’s software stack and hierarchy
- TrueTime: an innovation in timestamping
- Spanner’s operations
- Benchmarks
- F1
- DynamoDB
- Conclusion

Spanner

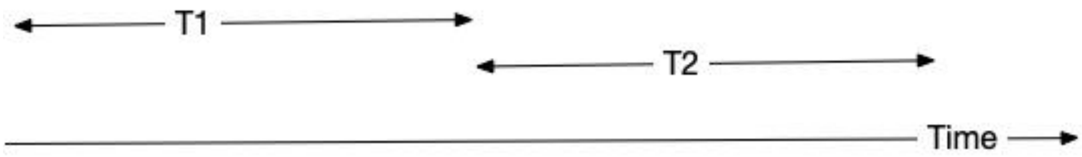
What is Spanner?

Spanner is a database with some extremely useful features from the **distributed systems** domain:

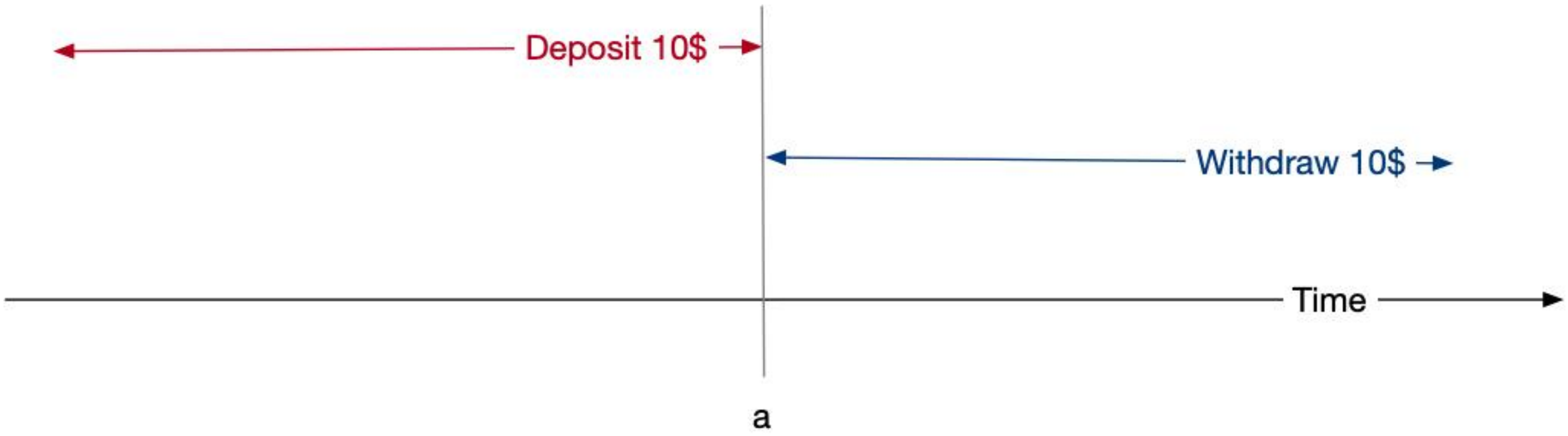
- Scalable
- Multi-versioned
- Globally distributed
- Synchronously replicated
- Supporting external consistent read writes
- Globally consistent reads
- And more

Scalable and globally distributed

Synchronously replicated, externally consistent



Synchronously replicated, externally consistent



TrueTime

Why TrueTime?

Clocks are never in absolute sync

Timestamps are the main way to order based on time

TrueTime

What is TrueTime?

Novel API

Exposes clock uncertainty

Guarantees Spanner's timestamps are bound

Spanner

What is Spanner?

Spanner is a **database** with some extremely useful features from the distributed systems domain:

- Semi-relational tables
- Query language

- Spanner: Becoming a SQL System *David F. Bacon et al.*

Spanner Implementation

- Universe
- Zones
- Spanservers

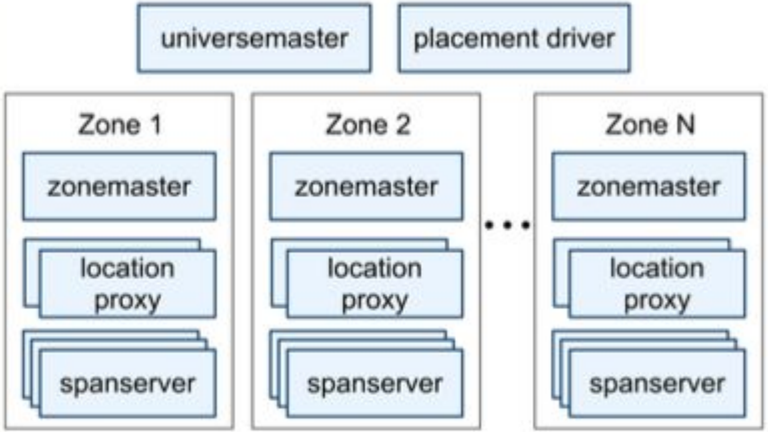


Figure 1: Spanner server organization.

Storage Data Model

(key:string, timestamp:int64) → string

100-1000 instances

Colossus

State machines store metadata

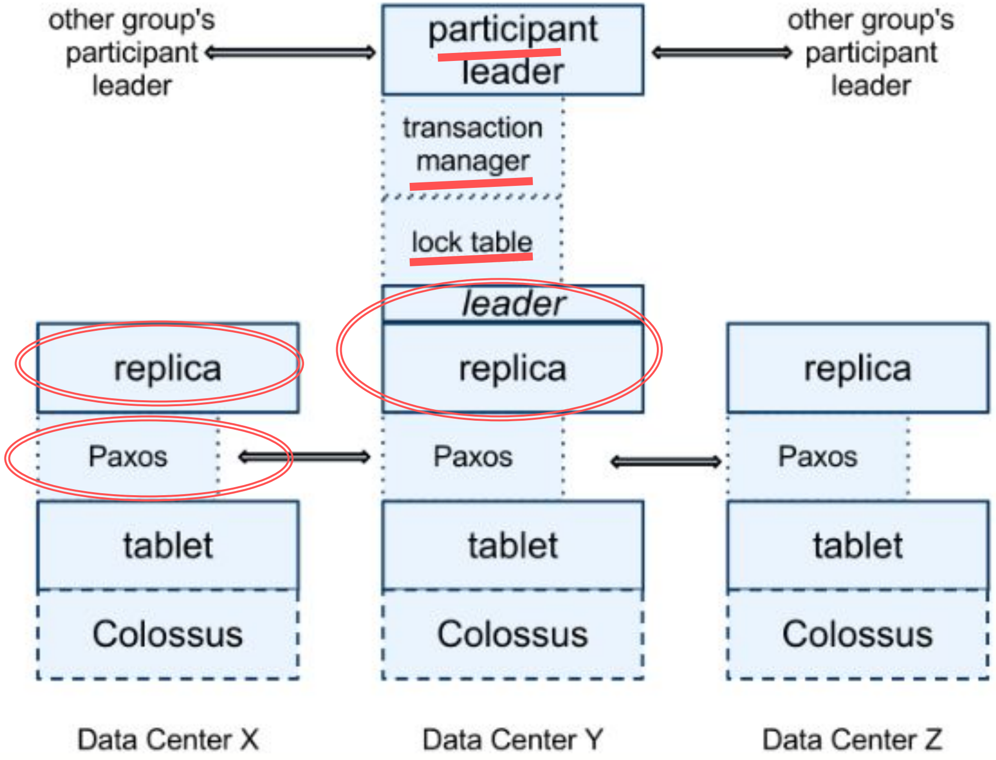
Why Paxos?

The Consensus Problem

- Collection of computers
- We want them to agree on something
- Consensus means agreement
- Reasons we might want consensus: mutual exclusion, elections, state machine replication
- Most frequently for replication
- Replication is useful for fault tolerance and scalability

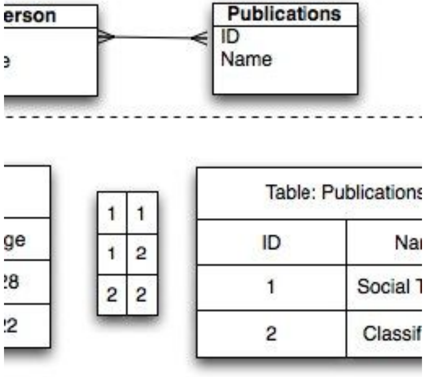
What is Paxos?

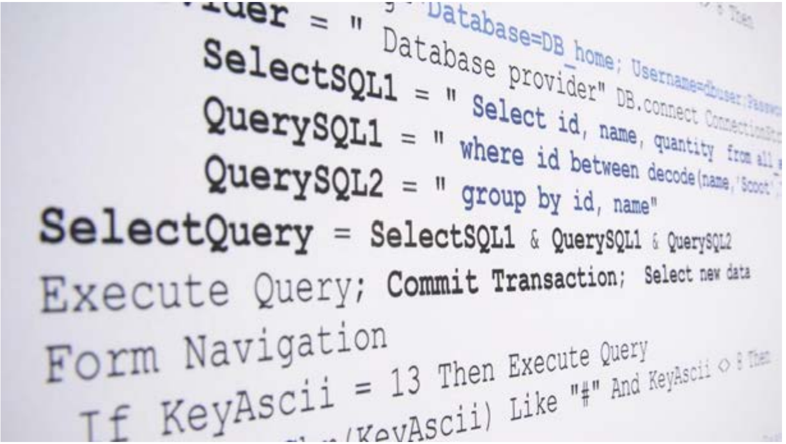
- Algorithm to achieve consensus
- Developed by Leslie Lamport (LaTeX, Byzantine fault tolerance, Lamport timestamps, Turing award winner)
- Set of computers that either are unreliable or their connection is unreliable
- Widely used (Google in Spanner, Chubby and Megastore, Yahoo in ZooKeeper)
- First consensus algorithm to be formally proven to be correct

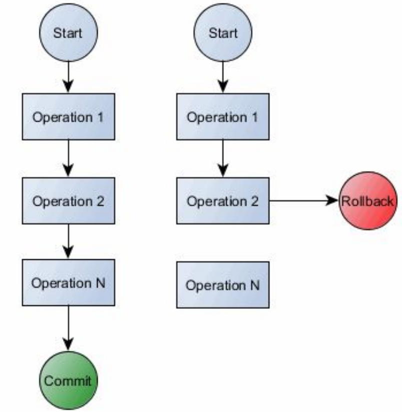


Spanner Software Stack

Data Model

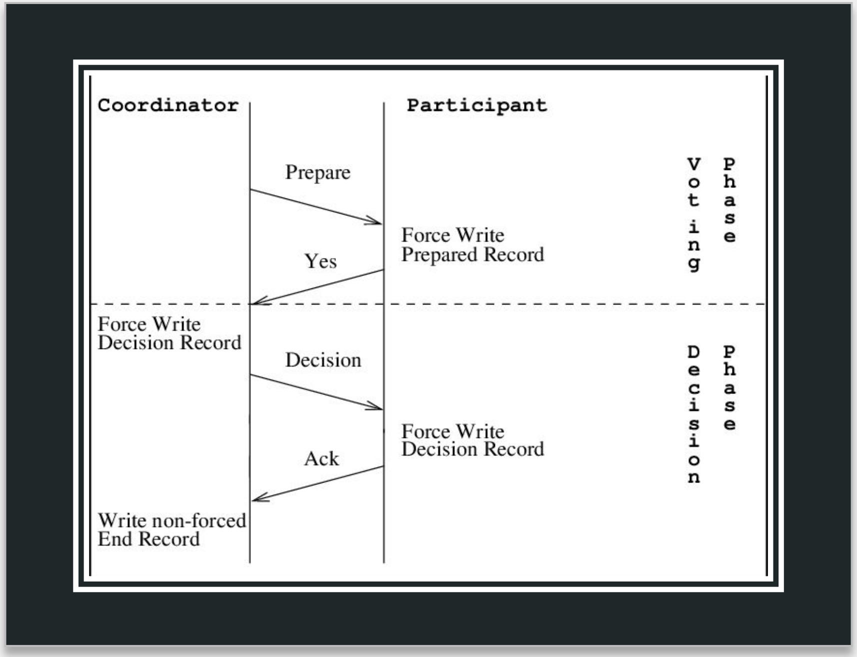

Publications
ID
Name
Table: Publications
ID
1
2


SelectSQL1 = " Select id, name, quantity
QuerySQL1 = " where id between
QuerySQL2 = " group by id, name"
SelectQuery = SelectSQL1 & QuerySQL1 & QuerySQL2
Execute Query; Commit Transaction; Select new data
Form Navigation
If KeyAscii = 13 Then Execute Query


Start
Operation 1
Operation 2
Operation N
Commit
Start
Operation 1
Operation 2
Rollback
Operation N

Two Phase Commit

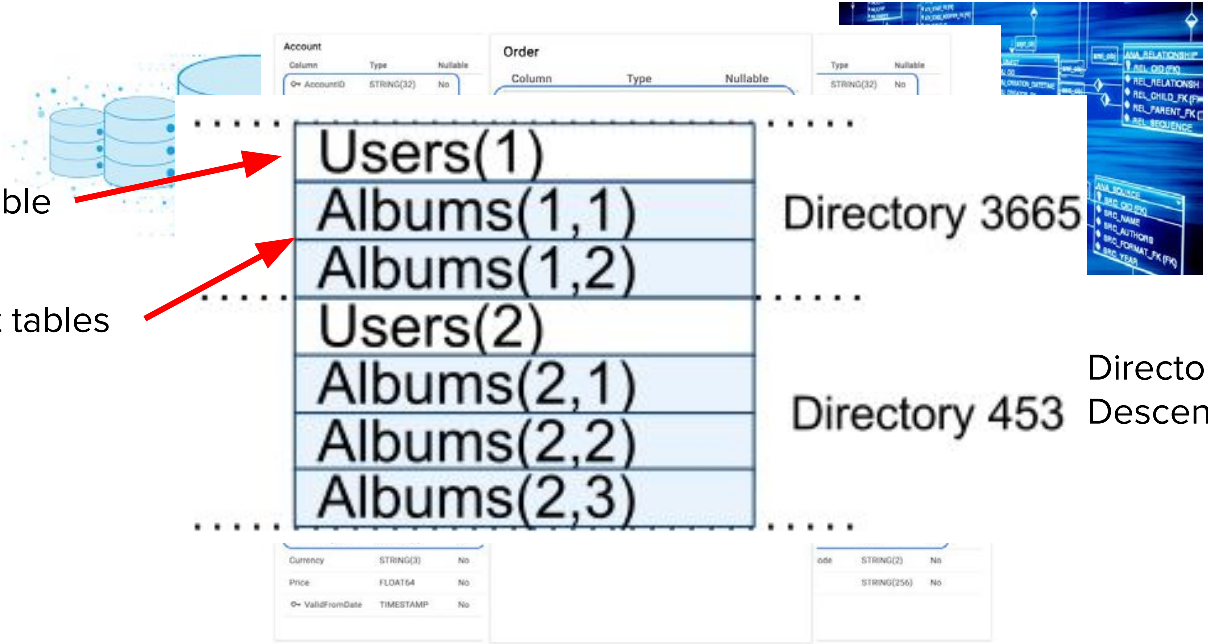
- ACID Databases - Atomicity, Consistency, Isolation, Durability
- 2PC - Two Phase Commit
- Role of Paxos



Data Model

Directory Table

Descendant tables



Directory table ***K*** +
Descendents = Directory

An Example

```
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;
```

Why Interleave?

TrueTime

TTinterval instead of e.g. seconds

Varying between 2-14ms

Spanner works reliably with clock uncertainty

ε is the instantaneous error and half of interval’s width

TrueTime API

Method	Returns
TT.now()	*TTinterval*: [*earliest*, *latest*]
TT.after(t)	true if t has definitely passed
TT.before(t)	true if t has definitely not arrived

TrueTime API. The argument t is of type TTstamp

Tt = TT.now()

Tt.earliest <= a <= Tt.latest

TrueTime time references

But how TrueTime stays within those strict bounds?

- Global Positioning System (GPS)
- Atomic clocks

Why those two?

- Completely different failure modes
- GPS fails mainly due to antenna and receiver failures
- Atomic clocks fail mainly due to frequency error
- There is no correlation between them

TrueTime hierarchy

- Time Master
- Timeslave Daemon

- Majority of masters have GPS
- Remaining have atomic clocks

- Masters and slaves communicate with Marzullo’s algorithm

TrueTime synchronizations and bounds calculation

Instantaneous error bound or ε

ε is derived by:

- Worst case local clock drift
- Masters’ uncertainty
- Communication overhead

TrueTime synchronizations and bounds values

Slave polling frequency is every 30 seconds

Worst expected clock drift is 200µs every 1s

200µs * 30s = 6ms

Communication overhead accounts for 1s

Total worst case maximum ε of 7ms

TrueTime issues

But when TrueTime fails?

Time - master unavailability

Network overload

Machines overload

Concurrency Control

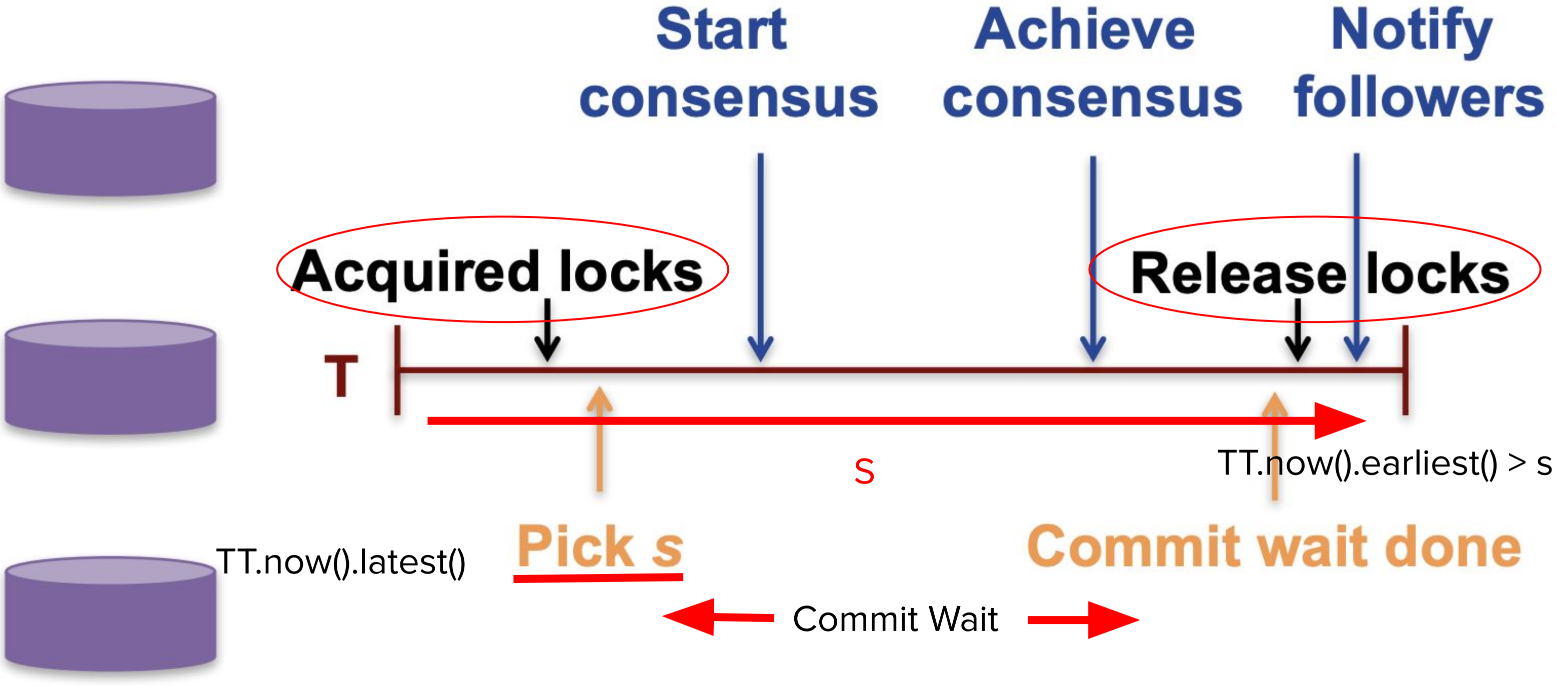
Timestamp Management

Read-write transactions

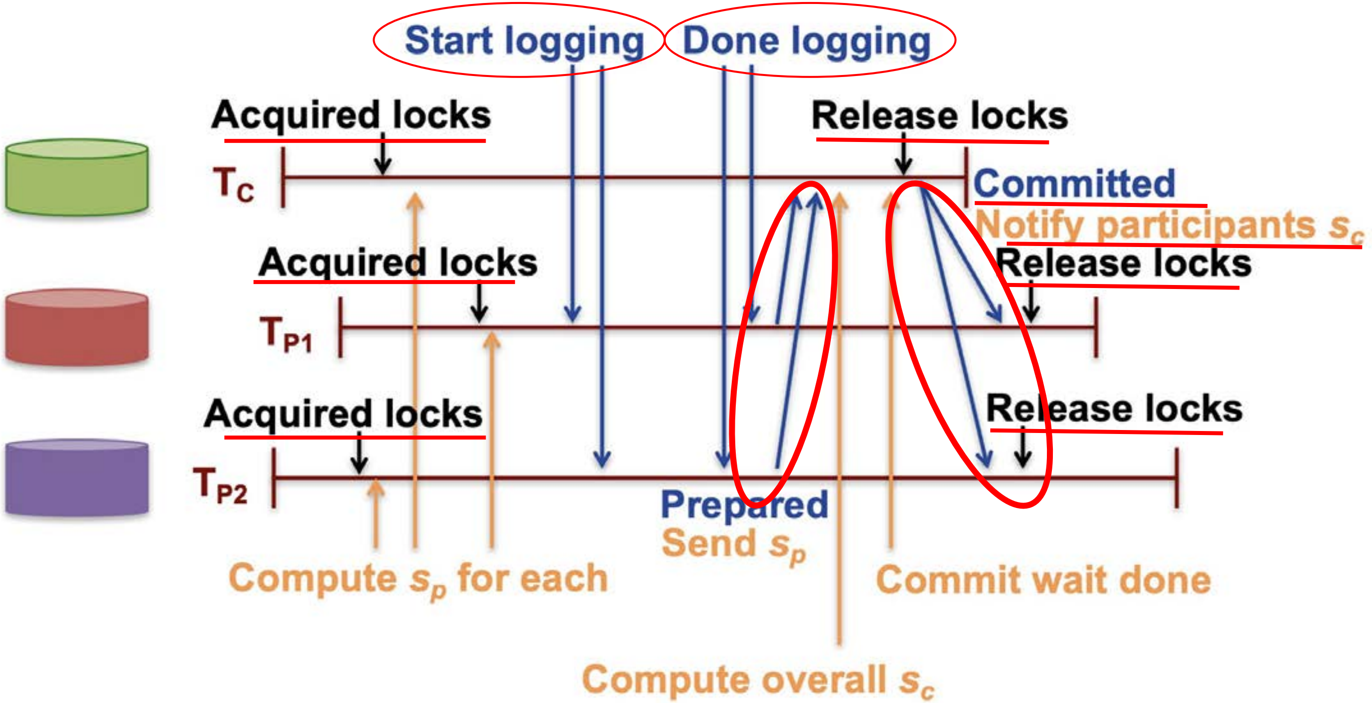
Read-only transactions

Snapshot reads

Commit Wait and Replication



RW Transactions - Commit Wait and 2PC



Example

Time	<8	8	15
My friends	[X]	[]	
My posts			[P]
X's friends	[me]	[]	

Read-Only Transactions

- 2 Phases:
  - Assign a timestamp -> S_read
  - Execute reads as snapshot reads
- Snapshot reads can execute at any replica that is up to date with respect to S_read

Schema-Change Transactions

- TrueTime enables atomic schema changes

Evaluation

Microbenchmarks

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
1D	9.4±.6	—	—	4.0±.3	—	—
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

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

Scheduling units of 4GB RAM and 4 cores (AMD Barcelona 2200MHz)

50 Paxos groups with 2500 directories. Operations were standalone reads and writes of 4KB,

Clients were run on separate machines.

Each zone contained one spanserver

Availability

Test universe is divided into 5 zones each with 25 spanner servers. All leaders were placed in Z1

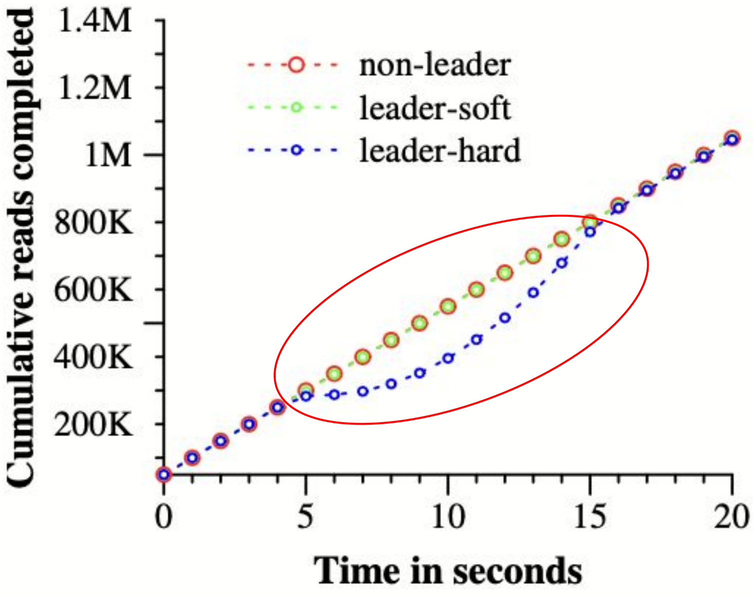


Figure 5: Effect of killing servers on throughput.

TrueTime

Fig represents truetime data at several thousand spanserver machines upto 2200 km apart.

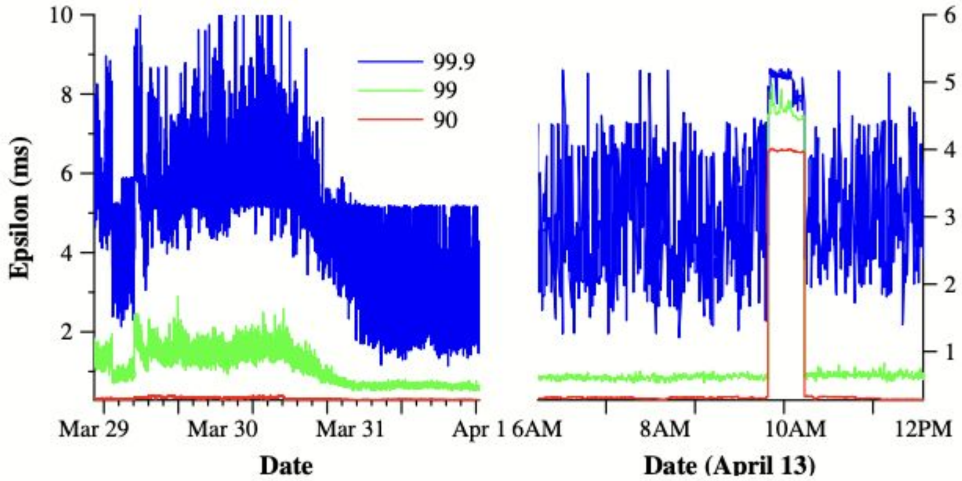


Figure 6: Distribution of TrueTime ϵ values, sampled right after timeslave daemon polls the time masters. 90th, 99th, and 99.9th percentiles are graphed.

F1

# fragments	# directories
1	>100M
2–4	341
5–9	5336
10–14	232
15–99	34
100–500	7

Table 5: Distribution of directory-fragment counts in F1.

operation	latency (ms)		count
	mean	std dev	
all reads	8.7	376.4	21.5B
single-site commit	72.3	112.8	31.2M
multi-site commit	103.0	52.2	32.1M

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

Related work

Megastore

DynamoDB

Conclusion

From the databases community perspective:

An easy-to-use, semi-relational interface that serves transactions utilizing an SQL-based query language

Conclusion

From the distributed systems community perspective:

Exceptional scalability, automatic sharding, fault tolerance, consistent replication, external consistency and wide area distribution

Conclusion

The linchpin of Spanner's feature set is TrueTime

By accepting and exploiting bounded clock uncertainty we can build distributed systems with much stronger time semantics