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

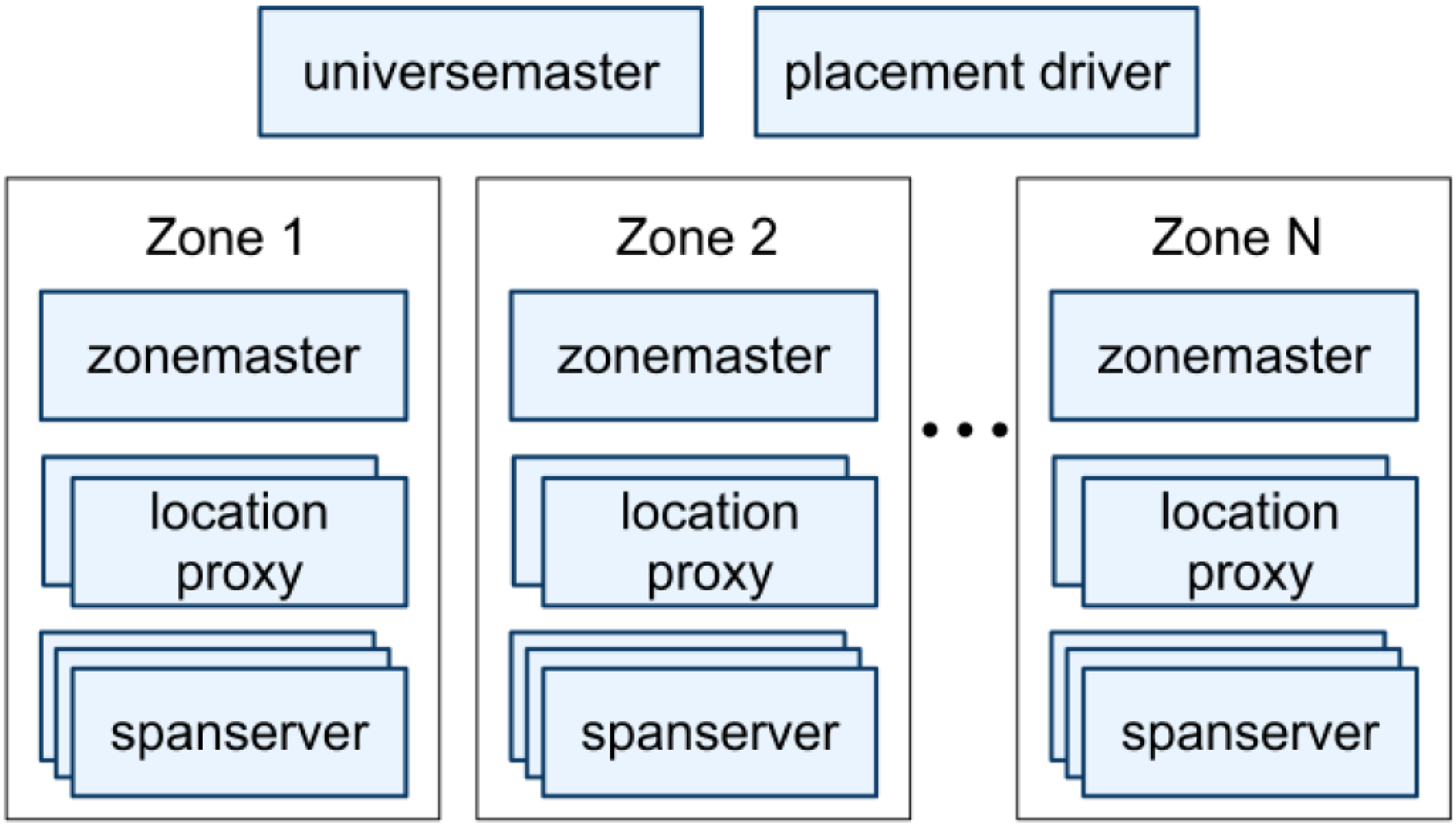
Solution: Spanner

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

Outline

- Motivation
- Spanner architecture
- TrueTime
- Transactional support
- Experiments

Spanner architecture



Spanservers

- Spanserver maintains data and serves client requests
- Data are key-value pairs
  `(key:string, timestamp:int64) -> 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

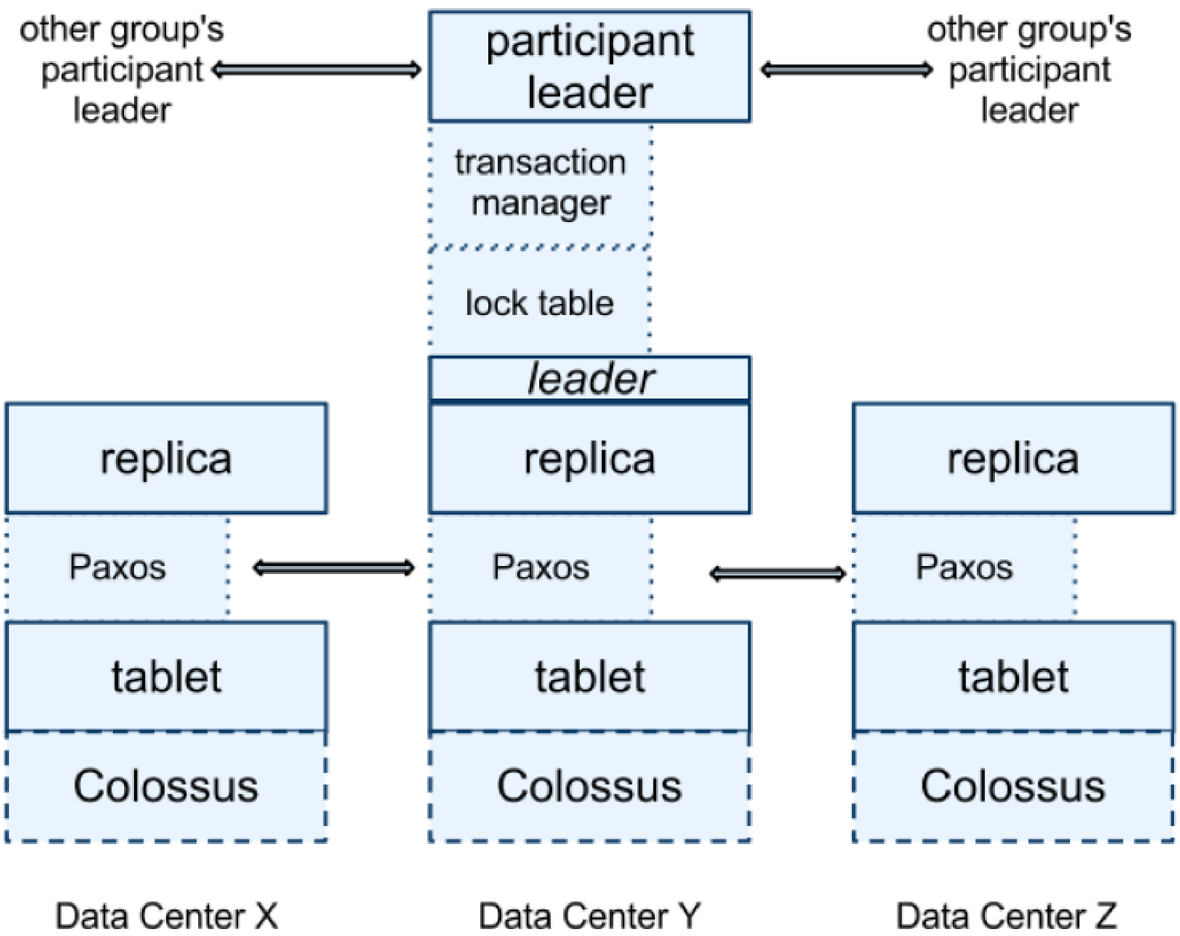
- Each participant can send message to each other to exchange some *state*
- Participant states must be consistent for each one to have a consistent view of *system state*

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 architecuture



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
- $TT.now.earliest \leq t_{abs}(now) \leq TT.now.latest$

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

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 ε is derived from local clock drift, time-master uncertainty and communication delays
  - ε 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

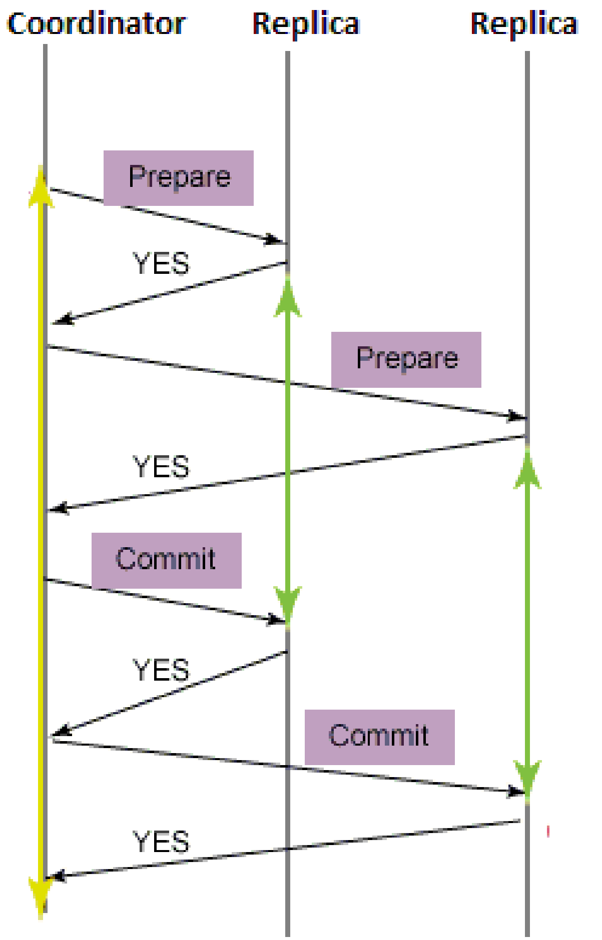
Transactions in Spanner

Operation	Concurrency control
Read-write transaction	Two-phase
Read-only transaction	Lock-free
Snapshot read	Lock-free

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 commited 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

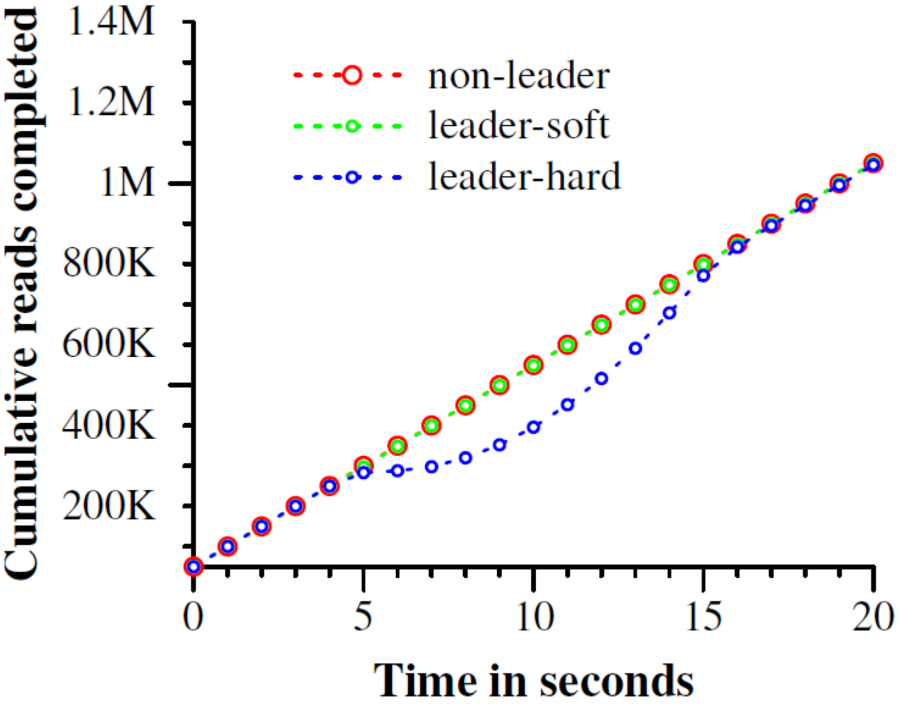
Microbenchmarks

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

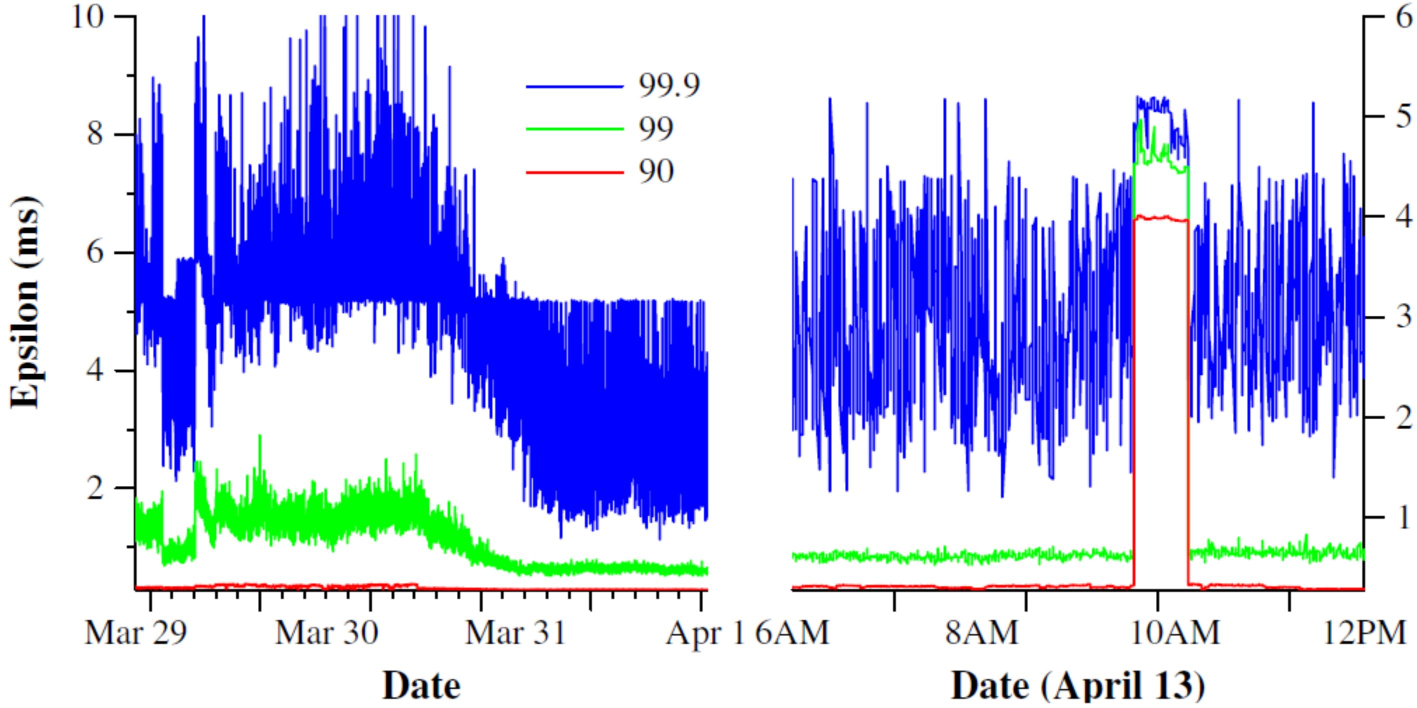
replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
				4.0±.3		
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

Availability

- Replicas manually killed to measure effect on read throughput



Distribution of ε values



Thank You!