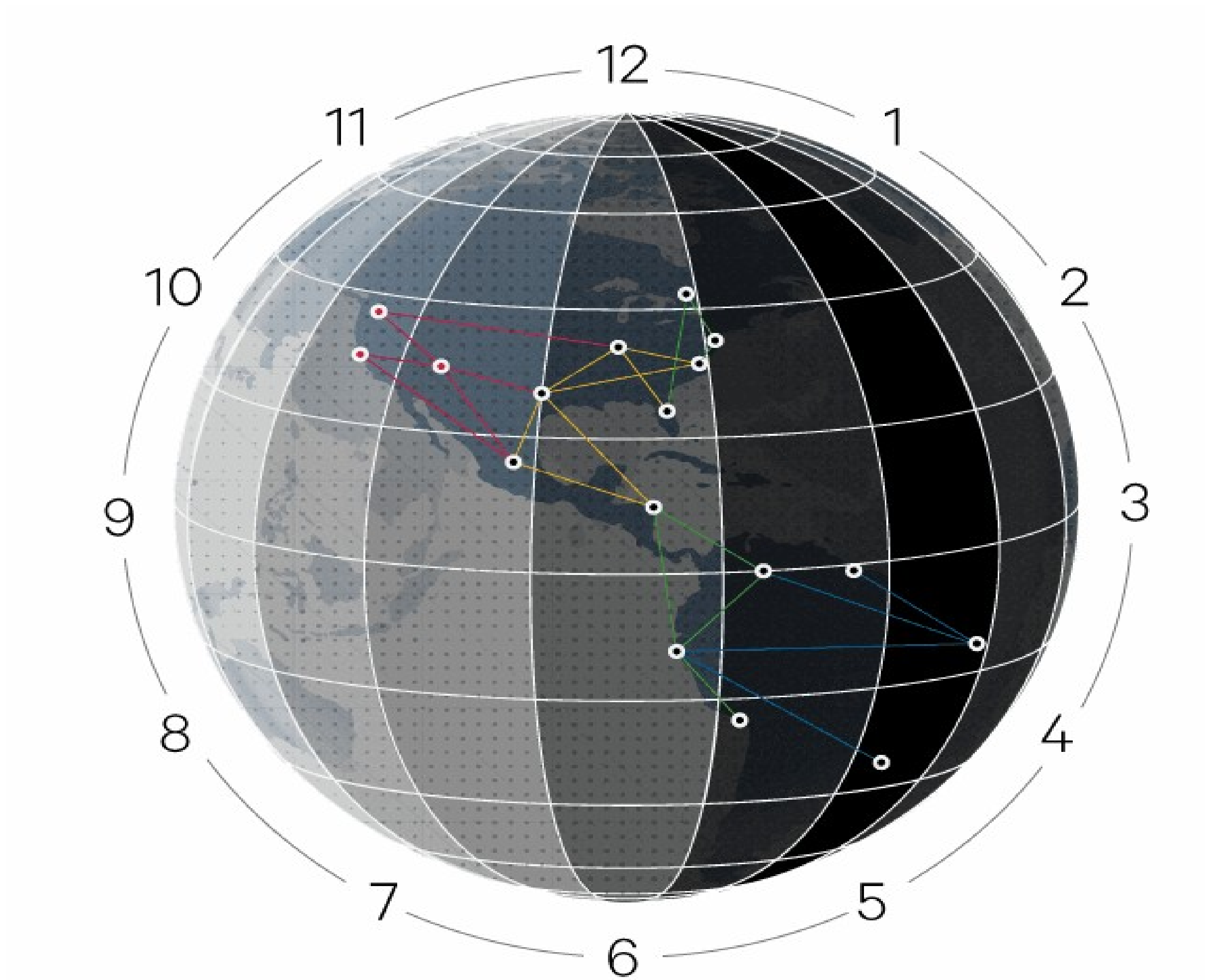
Spanner: Google's Globally Distributed Database



(presented by Philipp Moritz)

Why is this workload interesting?

- SQL → NoSQL → NewSQL
- Large scale transactional databases
- Eventual consistency is not good enough (?):
  - Managing global money/warehouses/resources
  - Auctions, especially Google's advertisement platform
  - Social networks, Twitter
  - MapReduce over a globally changing dataset
- We need external consistency:

  $T(e1(commit)) < T(e2(start)) \rightarrow s1 < s2$

Concepts

- Main idea:
  - Get externally consistent view of globally distributed database
  - Spanner = BigTable with timestamps + Paxos + TrueTime
- Details:
  - Globally distributed for locality and fault-tolerance
  - Automatic load balancing between datacenters
  - Semirelational + SQL like query language (cf. Dremel)
  - Versioning
  - Full control over
    - How far data is from user (read latency)
    - How far replicas are from each other (write latency)
    - How many replicas (durability, availability, throughput)

Paxos in a Nutshell

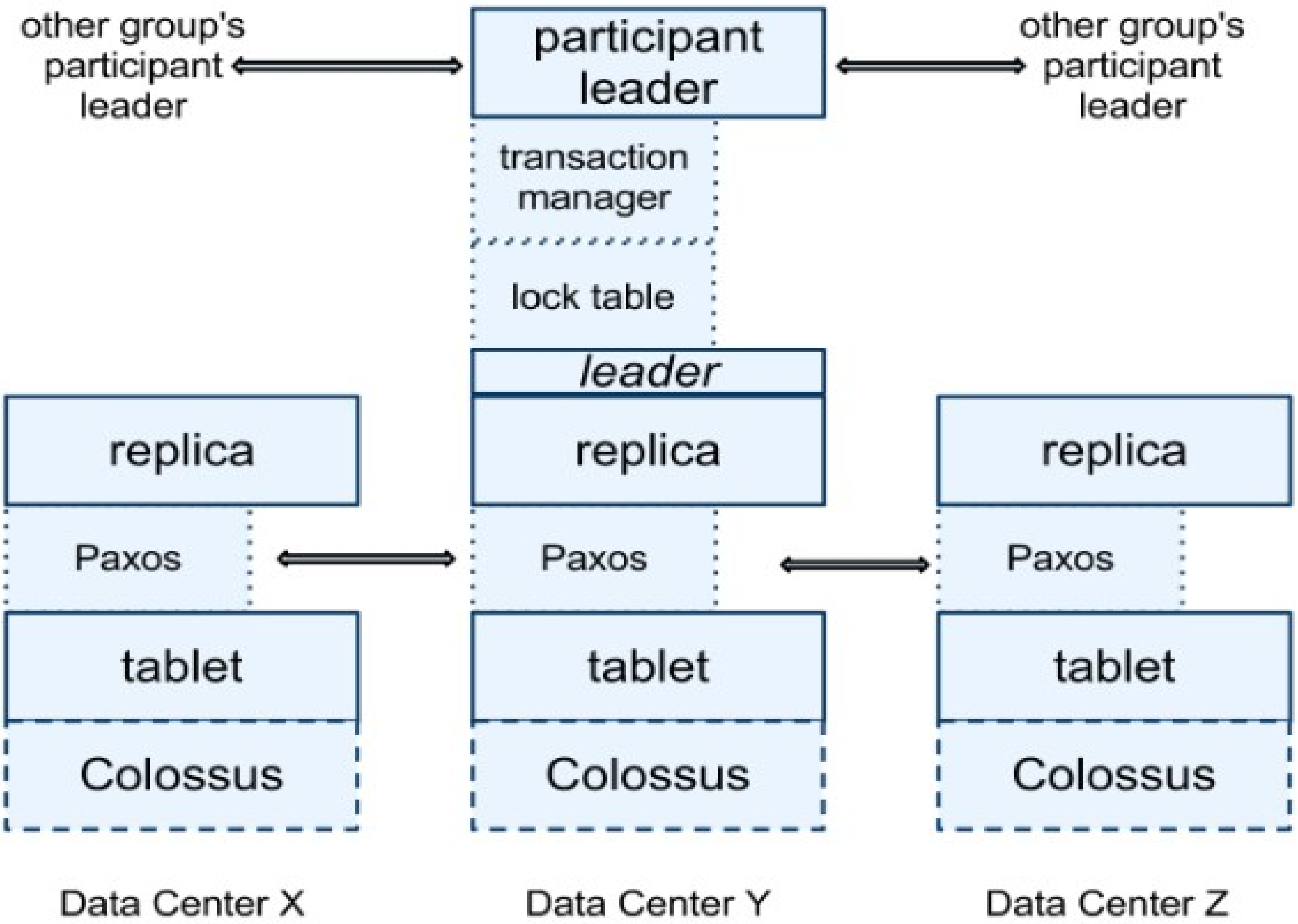
- Algorithm for finding consensus in a distributed system

```
Client   Proposer      Acceptor     Learner
   |         |          |  |  |       |  |
   X-------->|          |  |  |       |  |  Request
   |         X--------->|->|->|       |  |  Prepare(1)
   |         |<---------X--X--X       |  |  Promise(1,{Va,Vb,Vc})
   |         X--------->|->|->|       |  |  Accept!(1,Vn)
   |         |<---------X--X--X------>|->|  Accepted(1,Vn)
   |<---------------------------------X--X  Response
   |         |          |  |  |       |  |
```

TrueTime

- Goal: Provide globally synchronized time with sharp error bounds
- Do not trust synchronization via NTP
- With GPS and “commodity” atomic clocks, Google created their own time standard
- TrueTime API:
  - TT.now(): Interval [earliest, latest]
  - TT.after(t): true if t has definitely passed
  - TT.before(t): true if t has definitely not arrived
- Spanner implements algorithms to make sure these guarantees are respected by the machines (non-conformists are evicted)
- Time accuracy on the order of 10ms

Spanservers



Interplay of Paxos and TrueTime

- Guarantee externally consistent transactions

$$s_1 < t_{abs}(e_1^{commit}) \quad \text{(commit wait)}$$

$$t_{abs}(e_1^{commit}) < t_{abs}(e_2^{start}) \quad \text{(assumption)}$$

$$t_{abs}(e_2^{start}) \leq t_{abs}(e_2^{server}) \quad \text{(causality)}$$

$$t_{abs}(e_2^{server}) \leq s_2 \quad \text{(start)}$$

$$s_1 < s_2 \quad \text{(transitivity)}$$

Evaluation

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.

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

Table 4: Two-phase commit scalability. Mean and stan deviations over 10 runs.

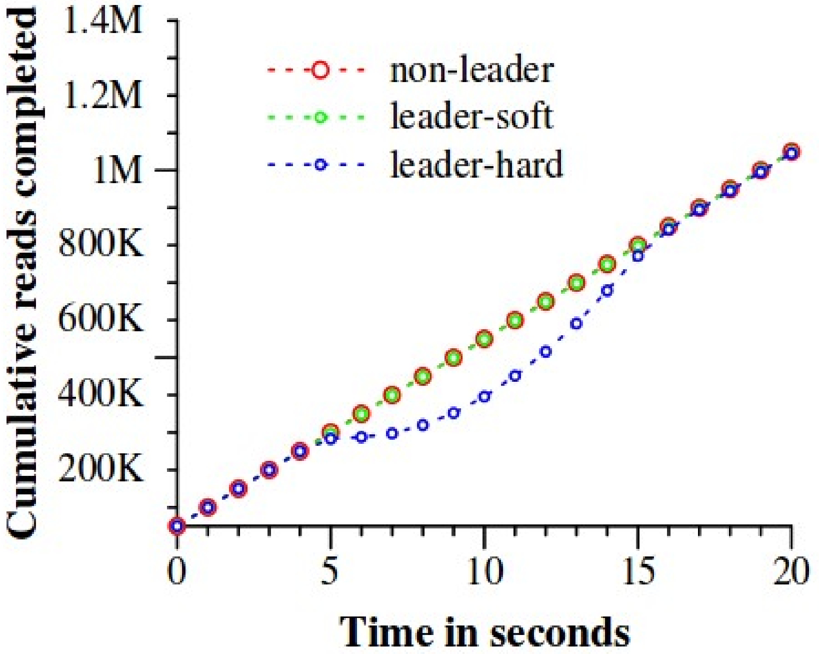


Figure 5: Effect of killing servers on throughput.

Discussion

- Tradeoff: Complexity of the System vs. Importance of Guarantees
- Is eventual consistency good enough if the operations we care about are fast enough?
- If not: Can we isolate a small subset of data for which we care about consistency and store it on a single server?
- Open Source implementation of similar ideas: https://github.com/cockroachdb/cockroach