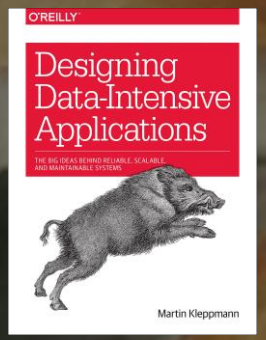
MSFT Sys Meetup



http://

Freely accessible resources

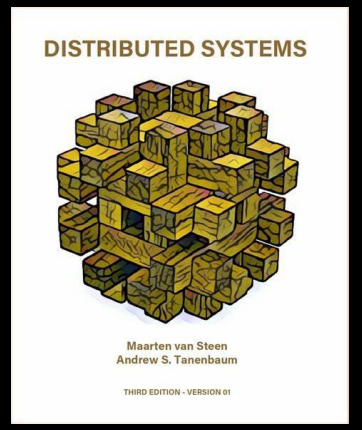
[Code]

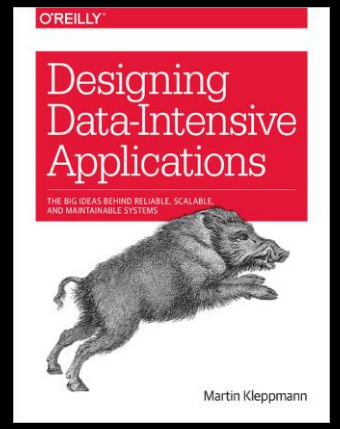
[Zoom]

[Course]

[DDIA (O'Reilly)]

[Distributed System 3rd edition]





Calendar: https://docs.google.com/spreadsheets/d/1RsbGpq1cwNSmYn5hcmT8Hv5O4qssl2HXsTcG82RHVQk/edit?usp=sharing

(Internal) Teams: g078pwd

(Public) Discord

(Public) WeChat: add mossaka or Lin1991Wen

Notion: https://www.notion.so/invite/cd6df70a94e7f67f6d21f4c509783d3c9cfd0e69

YouTube: https://www.youtube.com/playlist?list=PL1voNxn5MODMJxAZVvgFHZ0jZ-fuSut68

Company Privacy

Microsoft

Spanner

- Spanner
  - Google's scalable, multi-version, globally-distributed, and synchronously-replicated DB.
  - 1st system to distribute data at **global** scale and support **externally-consistent** distributed transactions.
- Motivation – F1 Advertising DB
  - Geographic replication
  - Strict serializability
  - Flexibility, scalability, availability…
  - Workload: most r/o

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.

Server Organization

- Zones
- Spanservers
- Tablets (Bag of <key, timestamp>=>value maps)
- Paxos state machine

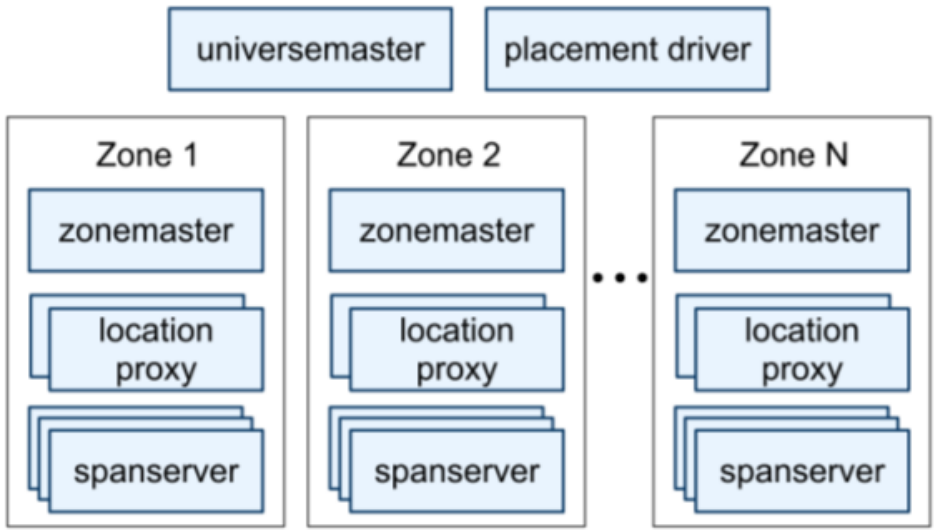


Figure 1: Spanner server organization.

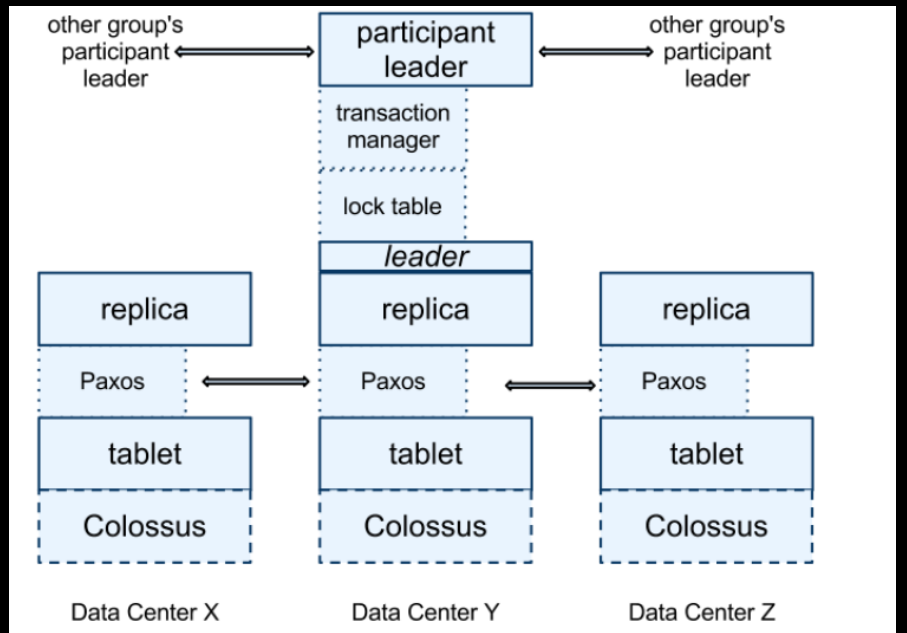


Figure 2: Spanserver software stack.

Data Organization

- <key, timestamp> -> val
- Keys partitioned “directories”
- Scalability, flexibility, fault tolerance, read perf

```
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;
```

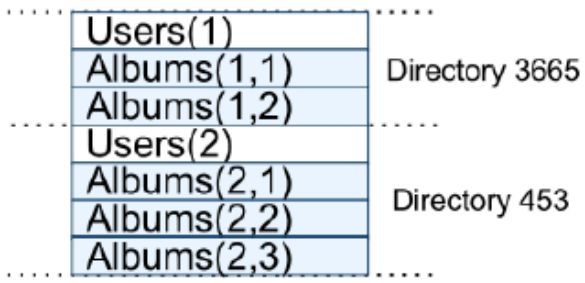


Figure 4: Example Spanner schema for photo metadata, and the interleaving implied by `INTERLEAVE IN`.

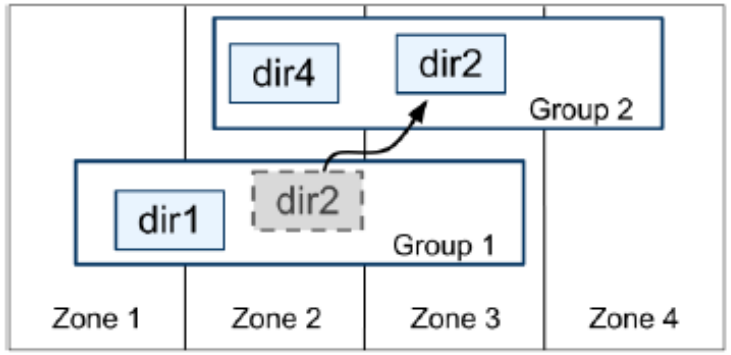


Figure 3: Directories are the unit of data movement between Paxos groups.

Spanner API

- SQL like query language
- Linearizable r/w transactions
- Linearizable lock-free r/o transactions
  - Can run at server-chosen recent time
  - Or client specified time/range (snapshot read)
  - r/o transactions don’t abort (because of no locking), unless old data has been garbage collected.

Main Idea

- 2PC over Paxos
  - Mitigate availability issue
- Use real time to order all transactions globally
  - 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

Table 1: TrueTime API. The argument t is of type $TTstamp$.

R/W Transactions

- 2PL
- Timestamp can be within [all locks acquired, any locks release) -> commit time
- External consistency
  - If the start of a transaction T2 occurs after the commit of a transaction T1, then the TS of T2 must be greater than the TS of T1.
  - **Start**: The coordinator leader for a write T_i assigns a commit timestamp s_i no less than the value of *TT.now().latest*, computed after the leader receives commit request.
  - **Commit Wait**: The coordinator leader ensures clients cannot see any data committed by T_i until *TT.after(s_i)* is true.

$$
\begin{aligned}
s_1 &< t_{abs}(e_1^{commit}) && \text{(commit wait)} \\
t_{abs}(e_1^{commit}) &< t_{abs}(e_2^{start}) && \text{(assumption)} \\
t_{abs}(e_2^{start}) &\leq t_{abs}(e_2^{server}) && \text{(causality)} \\
t_{abs}(e_2^{server}) &\leq s_2 && \text{(start)} \\
s_1 &< s_2 && \text{(transitivity)}
\end{aligned}
$$

R/W Transactions

- First, acquire all read locks cross groups to get read values, and client buffers writes
  - Reads won't reflect writes in the same transaction.
- Client initiates 2PC and selects coordinator group, and broadcasts commit request to all participant groups' leader.
- Non-Coordinator-Participant Leader acquires all write locks, and sends a "prepare timestamp" to coordinator leader.
- Coordinator leader also acquire all write locks, and assigns timestamp s:
  - S > TTnow().latest when commit request was received
  - S >= max prepare timestamp
  - S lies within lease terms of all participant leaders.
- Coordinator commit waits and then sends commit TS to client and all participant leaders. Writes can then be applied and locks are released.

R/O Transactions

- Client specifies all Paxos groups it will read
- Pick timestamp s for the transaction
  - For single-Paxos-group r/o transactions:
    - Safe: use TTnow().latest at the time read request received
    - Better: if no running transactions, use lastTS() (last committed transaction's TS)
  - For multi-group r/o transactions:
    - TTnow().latest
- Use snapshot isolation at s on reads
- A spanserver can safely respond to the read at s when s < min(t_Paxos, t_TM)
  - t_Paxos: last stamp in Paxos log
  - t_TM: infinity if no pending transactions, or lowest prepare time for running transactions.

Questions?

Freely accessible resources

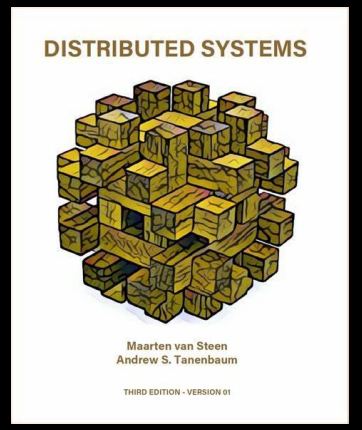
[Code]

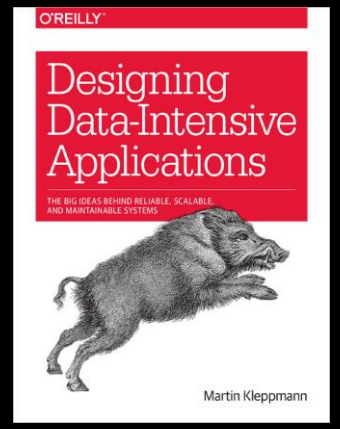
[Zoom]

[Course]

[DDIA (O'Reilly)]

[Distributed System 3rd edition]





Calendar: https://docs.google.com/spreadsheets/d/1RsbGpq1cwNSmYn5hcmT8Hv5O4qssl2HXsTcG82RHVQk/edit?usp=sharing

(Internal) Teams: g078pwd

(Public) Discord

(Public) WeChat: add mossaka or Lin1991Wen

Notion: https://www.notion.so/invite/cd6df70a94e7f67f6d21f4c509783d3c9cfd0e69

YouTube: https://www.youtube.com/playlist?list=PL1voNxn5MODMJxAZVvgFHZ0jZ-fuSut68