ECE 473/573
Cloud Computing and Cloud Native Systems
Lecture 13 Distributed Database Systems II

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

September 29, 2025

Outline

Spanner

Reading Assignment

- This lecture: Spanner: Google's Globally-Distributed Database `http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//archive/spanner-osdi2012.pdf`
- Next lecture: Introduction to Cloud Security

Outline

Spanner

Google Spanner

- A distributed multi-version database.
  - Semi-relational with SQL support.
  - Transactions with ACID guarantee.
  - Globally-distributed and horizontally scalable.
- Sharding automatically help to balance loads as data grow and as servers join and leave the cluster.
- Replication for global availablity and geographic locality.
  - Configurable by applications.
  - Location of replicas: read latency, write latency.
  - Number of replicas: read performance, durability and availability.

Spanner Deployment

- A Spanner deployment is called a universe.
  - E.g. a universe for testing, and another for production.
- Physical servers are managed as zones.
  - Each zone is the unit of administration and physical isolation.
  - E.g. a datacenter may contain multiple zones, one for each application that need to be isolated.
- Each zone has one zonemaster and a number (hundreds to thousands) of spanservers.
  - Zonemaster assigns data to spanservers.
  - Spanservers serve data to clients.

Tablets, Directories, and Placement

- Each spanserver manages many tablets.
- Each tablet maintains many versioned key-value pairs.
  - I.e., past updates to a key-value pair are recorded.
  - Tablets are replicated across many spanservers.
- Key-value pairs within a tablet are grouped into directories.
  - Keys in each directory share a common prefix.
- In other words, the common prefix determines where the key-value pairs are stored and replicated.
  - A directory is the smallest unit whose placement, i.e. geographic replication properties like 5 replicas in US, can be configurated by applications.

Data Model

- Similar to relational databases.
- An application can create multiple databases in a universe and each database consists of multiple schematized tables.
- Each table consists of rows and each row has a predefined list of columns, some as the key and the rest as the value.
  - Each row corresponds to a key-value pair in a tablet so its update history is recorded.
- Key columns are ordered and part of them are use for the common prefix defining the directory this row belongs to.

Data Model (Cont.)

- Unlike Cassandra, Spanner supports SQL features like joins.
- Support transactions across rows in a distributed manner.
- Provide consistency and partition tolerance, while let applications handles availability issues.
- Indeed, the CAP theorem says it is not possible to have 100% availability with consistency and partition tolerance, but in practice we don't always need 100% availability.

Consistency across Replicas of the Same Row

- Consensus: if multiple writes to the same row arrive at different servers, which one will succeed?
- For Cassandra, eventually consistent requires all replicas of the same row to be the same eventually when there is no more writes.
  - If writes are not acknowledged from all replicas, then there is no guarantee reads from the replica not acknowledged will return the same as reading other replicas – no consensus at all.
- For Spanner, a consensus protocol ensures replicas of the same tablet across multiple spanservers record the <u>same history</u>.
  - Reading any replica will give the <u>same history of writes</u> – only some replicas have more recent history than others.
  - We will introduce the consensus protocol Paxos toward the end of the semester.

Cross-Row Transactions

- However, the consensus protocol does not guarantee anything for writes to different rows not in the same tablet.
- Recall our social network example.
  - TABLE Friends stores rows for friendship relation.
  - TABLE Posts stores rows for posts.
- If a user A removes a friend X and then creates a post P, then A does not want X to read P.
- Three transactions are of interests for this scenario.
  - A0: remove X from A's friend list and A from X's friend list.
  - A1: add P to A's posts.
  - X0: read friend list of X, then list posts for each friend of X.
- A0 and A1 need to write to rows in different tablets and replicas and X0 need to read them.
  - The replicas containing A's friend list.
  - The replicas containing X's friend list.
  - The replicas containing A's posts.

Cross-Row Transactions (cont.)

- What if X0 reads a more recent replica with A's posts than a replica with X's friend list?
  - Output of X0 will include P which it should not.
  - As if A1 completes before A0.
- No, one cannot wait for all replicas to have the most recent data before executing X0.
  - There may be other transactions updating the replicas constantly.
  - Those transactions have to run concurrently, and cannot be blocked for availability and performance reasons.
- How does traditional relational database solve this problem with ACID guarantees?
- What prevents distributed databases to do the same?

ACID Guatantees

- Traditional relational databases provide ACID guarantees.
  - We can understand the overall effects of these transactions by inspecting all possible orderings assuming they execute and complete one after another.
- Six possible orderings of A0, A1, and X0
  - Three orderings have A0 completes before A1
  - The other three have A1 completes before A0
- For the correctness of transaction execution, we would expect the three with A0 before A1.
  - X0, A0, A1: X only sees post for A before A removes X
  - A0, X0, A1: X don't see any post from A
  - A0, A1, X0: A posts P but X don't see any post from A
- What prevents A1 to complete before A0?
  - Time causality on the single server: since user A wait for A0 to complete before starting A1, a local clock on that single server ensures A1 to start after A0 completes.

External Consistency

- In a distributed database when A0 and A1 write to rows on different servers, these servers have different local clocks.
  - X1 may see A1 completes before A0 using their local timestamps.
- External consistency: still, from the viewpoint of A's local clock, A0 does complete before A1 starts.
  - But how can such timing information be incorporated, which is <u>external</u> to the database system, into the transactions?
- Will it help if all local clocks synchronize with a global clock?
- Does such a global clock exist at all?

Version Data and Global Clock

- Since Spanner keeps versioned data, if the versioned data are stamped with a global clock, here is a possible solution.
  - X0's query into A's posts depending on a query on X's friend list. Therefore, it should not use any data from A's post more recent than from X's friend list.
  - X0 can be thought to happen sometime back in the history and correctness is achieved!
- For multiple transactions reading X's friend list,
  - Reading different replicas will result in different times back in the history those transactions thought to happen.
  - The consensus on the history among all replicas ensures their outcomes to follow external consistency.
- Can we maintain a global clock that multiple servers distributed to different locations can synchronize with?
  - But special relativity says there is no such global clock.

TrueTime

- GPS and atomic clocks can provide accurate time for local clocks and can compensate for each other as they have different failure modes.
- With an algorithm to synchronize time between local clocks, we can have the illusion of a global clock.
  - Each local clock has a time uncertainty with respect to the global clock that can be measured.
  - No violation of special relativity since uncertainty will increase as distances increase.
- Each transaction will use the local clock to stamp its writes.
  - To ensure that the timestamps from transactions to follow their commit order, transactions will need to wait twice of the uncertainty bound.

TrueTime Example

- Global clock uncertainty: 500ms
  - Local clocks on servers are less then 1s away from each other.
  - Servers have no other knowledge of local clocks of each other.
- Consider two servers
  - XF: the one containing X's friend list.
  - AP: the one containing A's posts.
- First A is removed from X's friend list
  - Stamped with local time of XF: 8:00:00.000
  - Local time of AP: 7:59:59.001
- Then P is added to A's posts.
  - Local time of AP should be at least 7:59:59.001
  - It is incorrect to stamp the event with 7:59:59.001.
  - Wait 2x500ms and stamp the event with 8:00:00.001.
- All queries now see P is added after A is removed.
- What if local time of AP is 8:00:00.999 when local time of XF is 8:00:00.000?