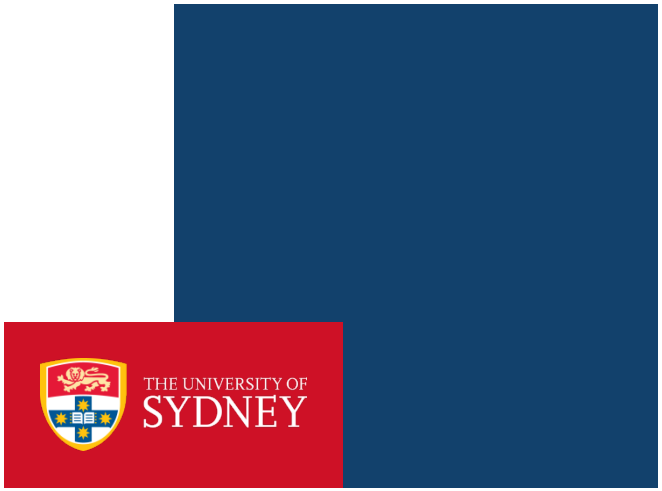
COMP5338 – Advanced Data Models

Week 6: Google Spanner

Dr. Ying Zhou
School of Information Technologies



Outline

- **Motivation**
- **Structure and Data Model**
- **Distributed Qu**

Motivation

- Requirements for cross-datacenter replication
  - Initial use case is the back end of Google's advertising services
  - Data are stored in 5 replicas across 3 or 5 data centers in USA
- Can Bigtable structure support such a scale?
  - Does the underlying file system scale?
  - Would single m
- The limitations o
  - It is not designed as a general pur system and can be difficult to use for some kind of applications, especially OLTP applications

Motivation (cont'd)

- An initial solution was to build a semi-relational data model on top of Bigtable (Megastore)
  - The performance of Megastore is not ideal
  - It still "lacked many traditional database features that application developers often rely on. A key example is a robust query language, meaning that developers had to write complex code to process and aggregate the data in their applications.
  - Used by many we gmail, Picasa, Calendar, etc
- Spanner evolved from a Bigtable-like y-value store into a temporal multi-version database
  - Data is stored in schematized semi-relational tables
  - Data version is automatically timestamped with its commit time
  - It provides a SQL-based query language and supports general-purpose long-lived transactions.

Outline

- Motivation
- **Structure and Data Model**
- Distributed Qu

Spanner Structure

- A spanner deployment is called
  - *Universe master* and *placement d*
    - *Both are singletons*
  - *Universe* consists of many zones
    - *Zone* is the rough analogue of Bigtable cluster

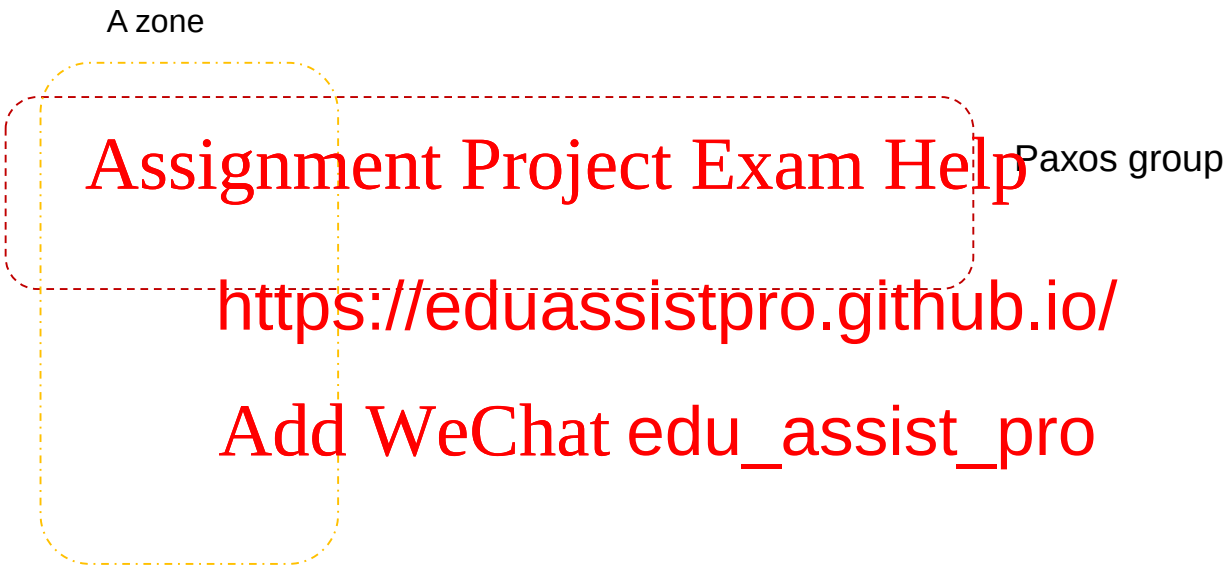
Zone Structure

- *Zone* is the rough analogue of Bigtable cluster
- A zone consists of
  - Many Spannerservers
    - Serve data to clients
  - One Zonemaster
    - Allocate data t
  - Location proxies
    - Help clients to locate the spanser their data
- Each *spansever* manages bet 1000 tablets
- Tablet is similar to Bigtable's tablet abstraction
- It stores versioned data of the format
  - (key:string, timestamp:int64) → string
- Actual data files and logs are stored on append only Colossus (the successor of GFS)

Data Replication

- Two levels of replication
  - Locally within the zone
    - Not managed by Spanner
  - Across zones
    - Managed by Spanner
- Universe master
  - Primarily a cons
- Placemant driver
  - Handles data movement across zones ‘
    - Load balancing
    - Satisfiying replication constraints

Spanner Software Stack



- Paxos algorithm is used to support replication and
  - The set of all replicas of a tablet forms a Paxos group
  - The leader uses locks to implement concurrent write and is act as transaction manager
  - There are many Paxos groups in the whole universe

Coordination activities involved

- A universe is expected to receive many transactions
- Concurrent transactions need to be executed in consistent order in the replicas involved
  - Replica 1: transaction a, b, c
  - Replica 2: transaction a, b, c
- Activities involve
  - Ensure consens mong all replicas
  - Execute the transactions accordi ed order
- **Paxos algorithm** is used to ensure consensus on the order
  - The replicas form a Paxos group
- **Lock mechanism** is used to control the concurrent execution

Paxos algorithm

- The name Paxos is from the original paper “The Part-time Parliament” by Leslie Lamport
  - It refers to a Greek Island
- The paper describe a parliament with legislators constantly wonders in and out of the parliamentary chamber
- Each legislator kee uence of decree passed
  - E.g.
    - 155: The olive t
    - 132: Lamps must use only olive oil
- The challenge is to ensure consisten rs with legislators wondering in and out
- The consensus algorithm proposed is called Paxos
- Determining which transaction to run at which time point fits perfectly with this scenario

Coordination activities involved (cont'd)

- Each transaction may contain a few queries
  - They should satisfy the basic ACID requirement
  - Transactions within the same Paxos group do not need extra coordination
    - Consistency of the replicas are guaranteed by Paxos
  - Transactions ac ated by two phase commit
    - Each involved group's participant in the protocol.
- Data consistency within each zone is managed by underlying file system

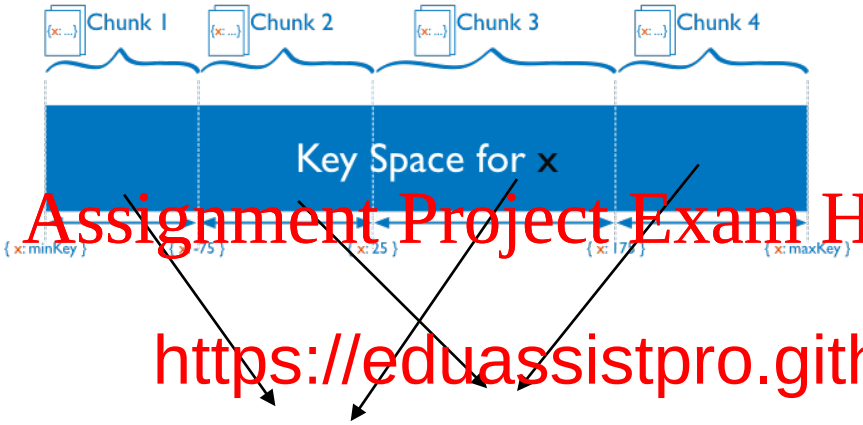
Directory and Placement

- Data in a tablet are organized as *directories*
  - *Directory* is a bucket like abstraction representing a set of **contiguous keys** in the tablet that **shares a common prefix**
- A *directory* is the unit of data placement
- A *directory* is also the unit of data movement between Paxos groups
- Spanner tablet i _____ tablet

dir1 and dir2 belongs to a tablet that is currently replicated on zone 1, 2, 3

dir2 may be moved to another tablet replicated on zone 2, 3 and 4

Revisit: data partition in MongoDB



Each chunk contains a contiguous range of sharding key values (or the hash of it)

Each shard stores a number of chunks

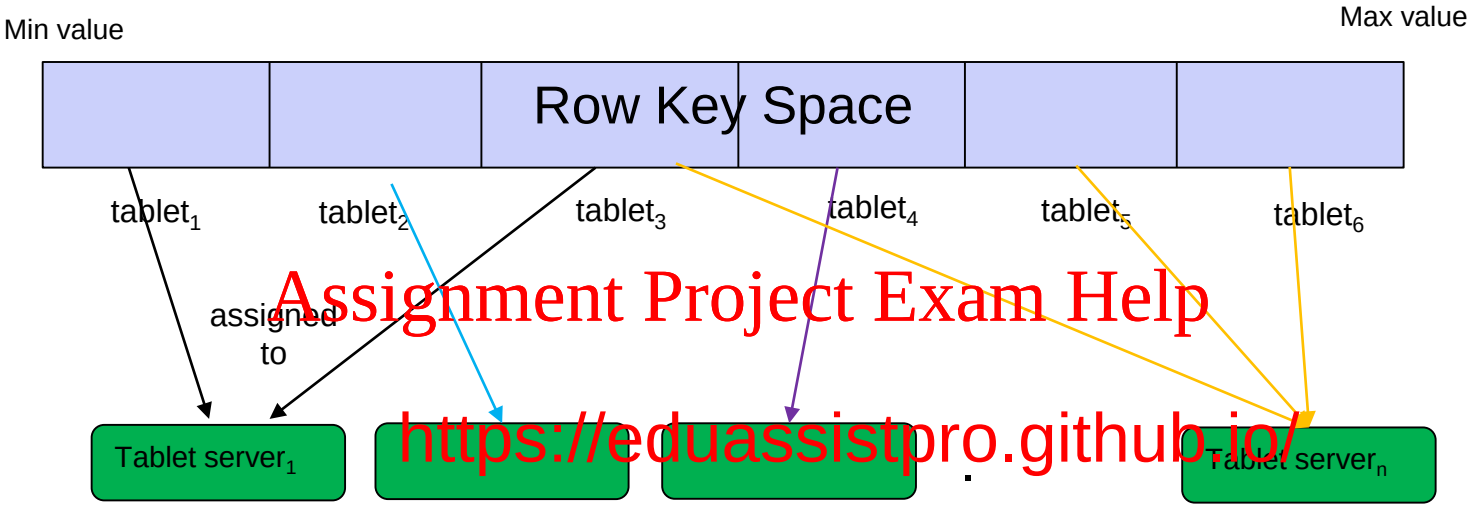
The chunks belonging to a shard do not have to be next to each other in terms of sharding key space

Each shard does not manage a contiguous range of sharding key values (or the hash of it)

Chunks can move around shards

Each shard is a replica set

Revisit: data partition in Bigtable



Each tablet contains a contiguous range of ro

Each tablet server manages a number of tablets

Bigtable tablets may vary in size; while MongoDB chunks are of fixed size

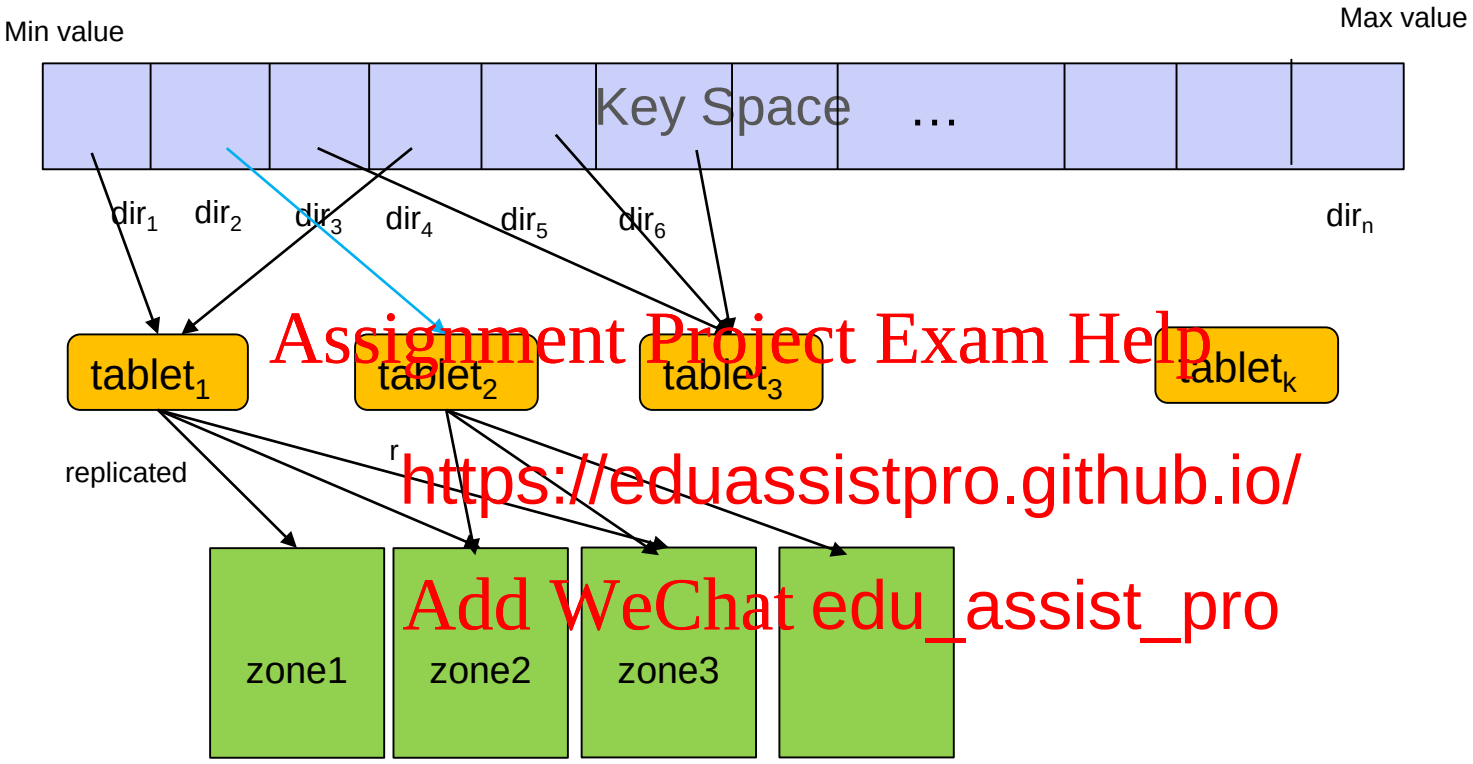
Each tablet server does not manage a contiguous range of row keys

Tablet may change its managing servers, but that does not necessarily result in data movement

Tablet may merge or split

Replication is managed by underlying GFS

Spanner data partition



Each directory contains a contiguous range of keys sharing a common prefix

Each tablet contains a number of directories not necessarily next to each other

Each tablet does not contains a contiguous range of keys

Each tablet is replicated to a number of zones, the replication is managed by Spanner

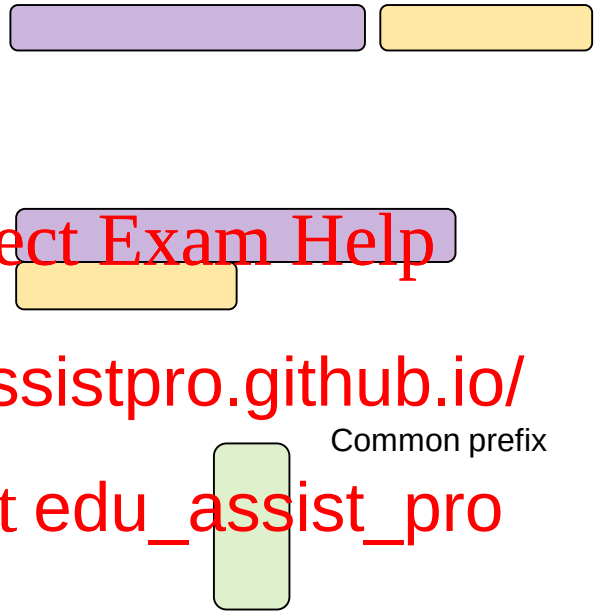
Within zone, the replication is managed by the underlying file system: Colossus

Data Model

- Each table is required to have one or more columns specified as **primary key**
- Each table defines a mapping from the primary-key columns to the other columns
  - (primary-key:string, timestamp:int64) → other columns
- Tables can have [illegible] the client when creating the tabl
- The hierarchy determines the k[illegible]airs in directories and in a tablet
  - A spanner tablet may contain data from more than one tables

Data Model Example

- Schema for storing photo meta data on a per-user, per-album basis
  - The **Users** table is declared as the parent table
  - The **Albums** is d the child table
  - The child table is co-located with the parent table
  - Similar to pre-joined tables in some RDBMS
- The child table includes the parent table's key as its primary key



Outline

- Motivation
- Structure and Data Model
- Distributed Qu

Query Plan Generation

- General process
  - The query compiler first transforms an input query into a relational algebra tree
  - Based on the actual schema and data properties, the optimizing compiler rewrites the initial tree into an efficient plan via transformation r
    - Well-known tr timization
    - Spanner speci query
- General principles for distribute n
  - Always do local operations first (in parallel) then merge the results
  - Several explicit distribution operator
  - A *Distributed Union* operator is used to ship a subquery to each shard, and to concatenate the results

$$\text{Scan}(\text{T}) \Rightarrow \text{DistributedUnion}[\text{shard} \subseteq \text{T}](\text{Scan}(\text{shard}))$$

Distributed query compilation

```
SELECT ANY_VALUE(c.name) name,
       SUM(s.amount) total
FROM Customer c JOIN Sales s ON c.ckey=s.ckey
WHERE s.type = 'global' AND
      c.ckey IN UNNEST(@customer_key_arr)
GROUP BY c.ckey
ORDER BY total DESC
LIMIT 5
```


The query returns t
the list **customer_key**
by total sum of sales of a particular ki
'**global**'.

Customer table is sharded on **ckey** and **Sales** table is interleaved in **Customer** and sharded by the same key

General Process

- Put a *Distributed Union* operator at the bottom of the tree, above every table in the query
- Pull up this operator as much as possible
- Similar as pushing down as much as possible other operators
- Any operators p ed Union should satisfy a property called *partitio*
  - An operator F satisfying partitiona that *performing an ordered union of the results of applying F to each shard in table key order gives the same outcome as applying F to the results of a global scan*

Operators that be pushed down

- Basic operations like **projection** and **filtering** below *Distributed Union*
- Joins between interleaved tables if the join key is the sharding key
  - Those tables are co-located based on the sharding key
- Operators that c d locally
  - E.g. Global Top ting local TopNs on each shard

Execution Plan

CrossApply(input, map) evaluates ‘map’ for every tuple from ‘input’ and concatenates the results of ‘map’ evaluations.

Distributed Execution

- At runtime, Spanner needs to work out where to send the subqueries: the shards and the servers managing the shards
  - Note: Google uses **shard** and **tablet** interchangeably in 2012 paper, and only **shard** in 2017 paper.
  - It has slightly dif in MongoDB
- If the query expr based on the sharding key
  - This filtering would be pushed furt
  - A subset of shards can be obtained and contacted
- If the query needs to visit every shard
  - A single call will be send to every server that managing some shards of the table

Query Distribution API

- The single-consumer API is used when a single client process consumes the results of a query.
  - Typical API supported by most storage systems
- The parallel-consumer API is used for consuming query results in parallel by multiple processes. E.g. in map-reduce style processing
  - Special API for

Single-Consumer API

- Spanner does not have designated coordinator for handling distributed query
  - In MongoDB, **mongos** is the dedicated coordinating service
- Theoretically any server hosting the data can function as a **root server** to coordinate the execution
- Using the server f the data would reduce unneces
  - Root server may need to merge p or do further processing
- Spanner uses a mechanism called location hint to ensure that would happen most of the time

Parallel-consumer API

- Use case:
  - Query results need to be further processed
  - Query results are too big to be processed in a single machine
- Solution
  - Sending partial results directly to the processing machines
- Restriction
  - Only queries that [illegible] parallel consumer API
    - Distributed Unio
    - E.g. the final result is just a simple co [illegible] e subqueries' results
  - Subquery results are sent direct to pr
- Process
  - The API needs to know the desired degree of parallelism and work out a set of opaque query partition descriptors
  - The query is executed on individual partitions, initiated from the processing nodes, e.g. the processing nodes become the clients

Key takeaway points

- A possible solution to distributed querying involving multiple tables
  - MongoDB shows an example of running distributed query involving single collection
  - Spanner uses co-located parent-child tables (pre-join solution)
- General rules of building query plan for complex distributed queries
- Special support for big data processing framework
- Difference between logical and physical data models
  - It is possible to have a ical layers totally different to classic RDBMS
- Indexing on non prima ery likely not supported
- Fault tolerance is a key issue but is not ex ugh detail
  - E.g. There is a whole section on Query R ch technical details are given
- Transaction support is the main topic in OSDI'12 paper, check the presentation video for more details
  - https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

References

- Corbett, James C, et al , **Spanner: Google's Globally-Distributed Database** *Proceedings of OSDI, 2012*
- Bacon, David F., et al. "**Spanner: Becoming a SQL System**." *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017.