Consistent Distributed Storage
Megastore System

- Paper is not specific about who is the actual customer of the system
- Guess (supported by Spanner paper): consumer-facing web sites and Google App Engine
  - selling storage as a service
  - not just an internal tool
  - Examples: email, Picasa, calendar, Android Market
What might the customer want?

- 100% available ==> replication, seamless fail-over
- Never lose data ==> don’t ack until truly durable
- Replicated at multiple data centers, for low latency and availability
- **Consistent** for *transactional* operations
- High performance
Transaction Semantics

- Transaction: BEGIN reads and writes END
- Serializable:
  - as if executed one at a time, in some order
  - no intermediate state visible
  - no read-modify-write races
  - transaction’s reads see data at just one point in time
- Durable
Conventional Wisdom

• Hard to have both consistency and performance in the wide area (as consistency requires communication)

• Popular solution: relaxed consistency
  • read/write local replica, send writes in background
  • reads may yield stale data, multiple write operations may not be atomic, RMW races may yield lost updates, etc.
Basic Design

- Each data center: BigTable cluster, application server + Megastore library, replication server, coordinator
- Data in BigTable is identical at all replicas
Browser web requests may arrive at any replica

That is, at the application server at any replica

There is no special primary replica

So could be concurrent transactions on same data from multiple replicas
Transactions can only use data within a single “entity group”

• An entity group is one row or a set of related rows
• Defined by application
• E.g., all my email messages may be in a single entity group; yours will be in a different one

Example transaction:
• Move msg 321 from Inbox to Personal
• Not a transaction: deliver message to both kaiyuan and paul
CREATE SCHEMA PhotoApp;

CREATE TABLE User {
   required int64 user_id;
   required string name;
} PRIMARY KEY(user_id), ENTITY GROUP ROOT;

CREATE TABLE Photo {
   required int64 user_id;
   required int32 photo_id;
   required int64 time;
   required string full_url;
   optional string thumbnail_url;
   repeated string tag;
} PRIMARY KEY(user_id, photo_id),
   IN TABLE User,
   ENTITY GROUP KEY(user_id) REFERENCES User;
## BigTable Layout

<table>
<thead>
<tr>
<th>Row key</th>
<th>User. name</th>
<th>Photo. time</th>
<th>Photo. tag</th>
<th>Photo. _url</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>John</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101,500</td>
<td></td>
<td>12:30:01</td>
<td>Dinner, Paris</td>
<td>...</td>
</tr>
<tr>
<td>101,502</td>
<td></td>
<td>12:15:22</td>
<td>Betty, Paris</td>
<td>...</td>
</tr>
<tr>
<td>102</td>
<td>Mary</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How would you build a wide-area storage system using Paxos? How do you achieve good performance?
Transactions

• Each entity group has a log of transactions
  • Stored in BigTable, a copy at each replica
  • Data in BigTable should be a result of playing log
  • Transaction code in application server:
    • Find highest log entry # (n)
    • Read data from local BigTable
    • Accumulate writes in temporary storage
    • Create log entry: the set of writes
    • Use Paxos to agree that log entry n+1 is new entry
    • Apply writes in log entry to BigTable data
Commit requires waiting for inter-datacenter messages

Only a majority of replicas need to respond

Non-responders may miss some log entries

Later transactions will need to repair this

There might be conflicting transactions
Concurrent Transactions

- Data race: e.g., two clients doing “x = x+1”
- Megastore allows one to commit, aborts the others
  - Conservatively prohibits concurrency within an entity group
  - So does not use traditional DB locking; which would allow concurrency if non-overlapping data
- Conflicts are caught during Paxos agreement
  - Application server will find that some other transaction got log entry n+1
  - Application must retry the whole transaction
Reads

- Must get latest data
- Would like to avoid inter-replica communication
- Ideally would read from local BigTable w/o talking to any other replicas
- Problems?
- Solutions?
Rotating Leader

- Each accepted log entry indicates a "leader" for next entry
  - Leader gets to choose who submits proposal #0 for next log entry
  - First replica to ask wins that right
  - All replicas act as if they had already received the prepare for #0

- Why and when does this help?
Log Format
What if concurrent commits?

- Leader will give one the right to send accepts for proposal #0
- The other will send prepares for higher proposal #
- The higher proposal may still win!
- So proposal #0 is not a guarantee of winning
  - Just eliminates one round in the common case
“Write” Details

- Ask leader for permission to use proposal #0
- If “no”, send Paxos prepare messages
- Send accepts, repeat prepares if no majority
- Send invalidate to coordinator of ANY replica that did not accept
- Apply transaction’s writes to as many replicas as possible
- If you don’t win, return an error; caller will rerun transaction
Failure: Overloaded replica (R1)

- R1 won’t respond
- Transactions can still commit as long as majority respond
- Need to talk to R1 coordinator to clear the flag it maintains for being up-to-date
- Reads at R1 will use a different replica
Failure: replica disconnection

- Designers view this as rare
- Replica won’t respond to Paxos (OK), but coordinator not responding is a problem
  - Write will block
- Paper implies that coordinators have leases
  - Each must renew lease at every replica periodically
  - If it doesn’t/can’t
    - Commits can ignore the replica
    - Replica marks all entity groups as “not up to date”
MegaStore Summary

- High availability through replication, seamless fail-over
- Replicated at multiple data centers, for low latency and availability
- Ack only when truly durable
- Consistency for *transactional* operations
- Performance improvements
Spanner

- Picks up from where MegaStore left off
- Some commonality in terms of mechanisms but a different implementation
- Key additions:
  - general-purpose transactions across entity groups
  - higher performance
  - “TrueTime” API and “external consistency”
  - multi-version data store
Example: Social Network

- Consider a simple schema:
  - User posts
  - Friend lists

- Looks like a database, but:
  - shard data across multiple continents
  - shard data across 1000s of machines
  - replicated data within a continent/country

- Lock-free read only transactions
Read Transactions

- Example: Generate a page of friends’ recent posts
  - Consistent view of friend list and their posts
  - Want to support:
    - remove friend X
    - post something about friend X
- **MegaStore**: transactions within entity groups
- **Spanner**: transactions across entity groups

  How can you support transactions across entity groups, where each entity group is replicated across datacenters?
Spanner Transaction

- Two-phase commit layered on top of Paxos
  - Paxos provides reliability and replication
  - 2PC allows coordination of different groups responsible for different datasets
  - Layering provides non-blocking 2PC
- Uses 2-phase locking to deal with concurrency
Spanner’s TimeStamps

- TrueTime: “Global wall-clock time” with bounded uncertainty
- Returns a lower-bound and upper-bound on wall-clock time

![Diagram showing TrueTime with earliest and latest bounds](diagram.png)
Spanner Transaction

- Each participant selects a proposed timestamp for the transaction greater than what it has committed earlier.
- Coordinator assigns the transaction a timestamp that is greater than these timestamps.
- Coordinator waits until the chosen timestamp is definitely in the past.
- Then notifies the client and the participants of the transaction’s timestamp.
- Participants release the locks.
Read Transactions

- Currently handled at the group leaders
- Two forms: read transactions across multiple groups, read transaction across a single group
- In both cases:
  - check whether there is an ongoing transaction
  - attribute the earliest possible timestamp that is safe
  - wait for a certain period before responding
Summary

- GFS: blob store abstraction
- BigTable: semistructured table abstraction within a datacenter
- MegaStore: limited transactions across multiple datacenters
- Spanner: more general transactions across multiple datacenters