**Megastore**

Multi-datacenter storage with strong consistency for interactive services! From the design, can guess that Megastore is what powers Google App Engine's storage. Later on, Spanner paper revealed that many products (gmail, picasa, calendar, …) use Megastore.

Conventional Wisdom

- hard to have both consistency and performance in the WAN
  - why? consistency requires communication to coordinate
- hard to have both consistency and availability
  - why? need two-phase commit across partitions

- popular solution: relaxed consistency (next week)
  - read/write local replica, send updates in the background
  - stale data, RMW races with lost updates, etc.
- megastore: no, no – we want it all!

Several facets of Megastore's design are unusual:

- synchronous replication
  - Paxos for every update operation, across the WAN!
  - one WAN RTT per update; implications?
- ACID transactions across the WAN
- Not a primary/backup; any replica can initiate an update (Paxos)

**Data model**

A schema has a set of tables, containing a set of entities, containing a set of properties.

```
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;
```

- tables look very similar to SQL tables, with additional megastore-specific annotations
- one of them is the "entity group" annotation

Transactions can only use data within a single "entity group".

- entity group is a row, or set of related rows, as defined by an application
- e.g., all my email messages in one entity group; all of your email messages in a different one.
  - so..."move message 321 from inbox to personal" is a transaction
  - but…"delivery message to both Steve and Johnson" is not

- *implementation-wise, a per-entity-group transaction log is stored within the bigtable row of the "root entity" – bigtable permits atomic multi-ops on a row, so can update the transaction log atomically*

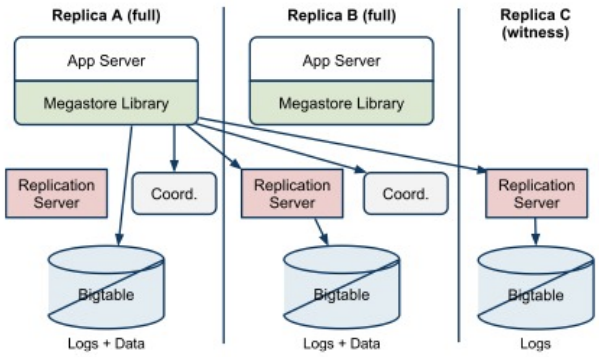
The data model is mapped to an underlying bigtable. Note clever use of row keys; related data items within an entity group end up being lexicographically close to each other, hence live within a single BigTable tablet.

| **Row key** | User. name | Photo. time | Photo. tag | Photo. _url |
|---|---|---|---|---|
| **101** | John | | | |
| **101,500** | | 12:30:01 | Dinner, Paris | ... |
| **101,502** | | 12:15:22 | Betty, Paris | ... |
| **102** | Mary | | | |

**System architecture**

Each data center (i.e., replica) has:

- a bigtable cluster
- an application server + megastore library
- a replication server
- a coordinator




The BigTable stores both data according to the data model, and a set of transaction logs

A browser's request can show up at any replica

- there is no special primary replica
- so, can have concurrent transactions on the same data from multiple replicas – need to deal with this. how? paxos.

**Transactions in megastore**

- each entity group has its own log of transactions
  - the log itself is stored in bigtable, and a copy is at each replica
  - data in BigTable should be the result of playing the log
- the transaction code in an application server looks roughly like:
  - find the highest paxos log entry number (N)
  - read data from the local bigtable
  - accumulate writes in temporary storage
  - create a log entry that is the set of writes
  - **use paxos to agree that the new entry is paxos log entry N+1**
  - apply writes in log entry to bigtable data

- Notes:
  - transaction commit requires waiting for inter-datacenter messages (Paxos)
  - only a majority of replicas needs to respond (Paxos)
    - non-responders might miss some log entries, and later transactions will need to repair this
  - there might be conflicting concurrent transactions!

- concurrent transactions
  - data race: e.g., two clients doing "x = x + 1"
  - megastore will allow one to commit, and aborts the other
    - actually, conservatively, megastore prohibits all concurrency within an entity group, whether or not a conflict occurs
    - does not use traditional DB locking, which would permit concurrency if non-overlapping data were in use
  - concurrent transactions are caught during Paxos agreement
    - one application server will find another got log entry N+1
    - must retry the entire transaction if so

- Reads
  - must get the latest data
    - could just use paxos for reads, but want to avoid inter-replica communication
    - i.e., ideally, read from local bigtable without talking to any other replicas
      - issue #1: maybe this replica missed some operations
      - issue #2: maybe replica's log has operations not yet applied
  - solution
    - the coordinator
      - paxos writes update each coordinator
      - a coordinator tracks the set of keys for which the local entity group is up-to-date, i.e., has seen all writes
      - big issue – in the write algorithm, each replica must either (a) accept the write or (b) have its coordinator invalidated. So, if a replica is unavailable, this can cause unavailability of writes in the system!!
    - in practice, if a coordinator fails, will lose its chubby locks
      - if a writer can't talk to a coordinator, monitors locks, and when lock is lost by coordinator, writer can grab lock to cause coordinator to invalidate itself on restart
      - can cause tens of seconds of outage when coordinator fails

**Performance**

- reads take 10s of milliseconds
- writes take 100s of milliseconds

Is this OK?