Megastore Replication

Presented by : Peyman Navidi

Megastore – Replication

- Megastore's replication system provides a **single**, **consistent** view of the data stored in its **underlying replicas**.
- Reads and writes can be **initiated from any replica**, and **ACID** semantics are preserved regardless of what replica a client starts from.
- Replication is done per entity group by **synchronously** replicating the group's transaction log to a quorum of replicas.
- Writes typically require **one round** of inter datacenter communication, and healthy-case reads run locally.
- Current reads have the following guarantees:
  - A read always observes the **last-acknowledged write**.
  - After a write has been observed, all future reads observe that write. (A write might be observed before it is acknowledged.)

Brief Summary of Paxos

PAXOS Algorithm

- A way to **reach consensus** among a group of replicas on a single value.
- Tolerates delayed or reordered messages and replicas that fail by stopping.
- The original PAXOS algorithm **is ill-suited** for **high-latency** network links because it demands multiple rounds of communication so Megastore uses an improved version.
- Databases typically use PAXOS to replicate a transaction log, where a separate instance of PAXOS is used for each position in the log.
- Family of algorithms (by Leslie Lamport) designed to provide **distributed consensus** in a network of several replicas

Brief Summary of Paxos

- **A majority of replicas** must be active and reachable for the algorithm to make progress that is, it allows up to F faults with **2F + 1** replicas.

- New values are written to the log at the position following the last chosen position.

- Basic Paxos not used (poor match for high-latency links)
  - Writes require **at least two inter-replica roundtrips** to achieve consensus (prepare round, accept round)
  - Reads require one inter-replica roundtrip (prepare round)

Master Based Approaches

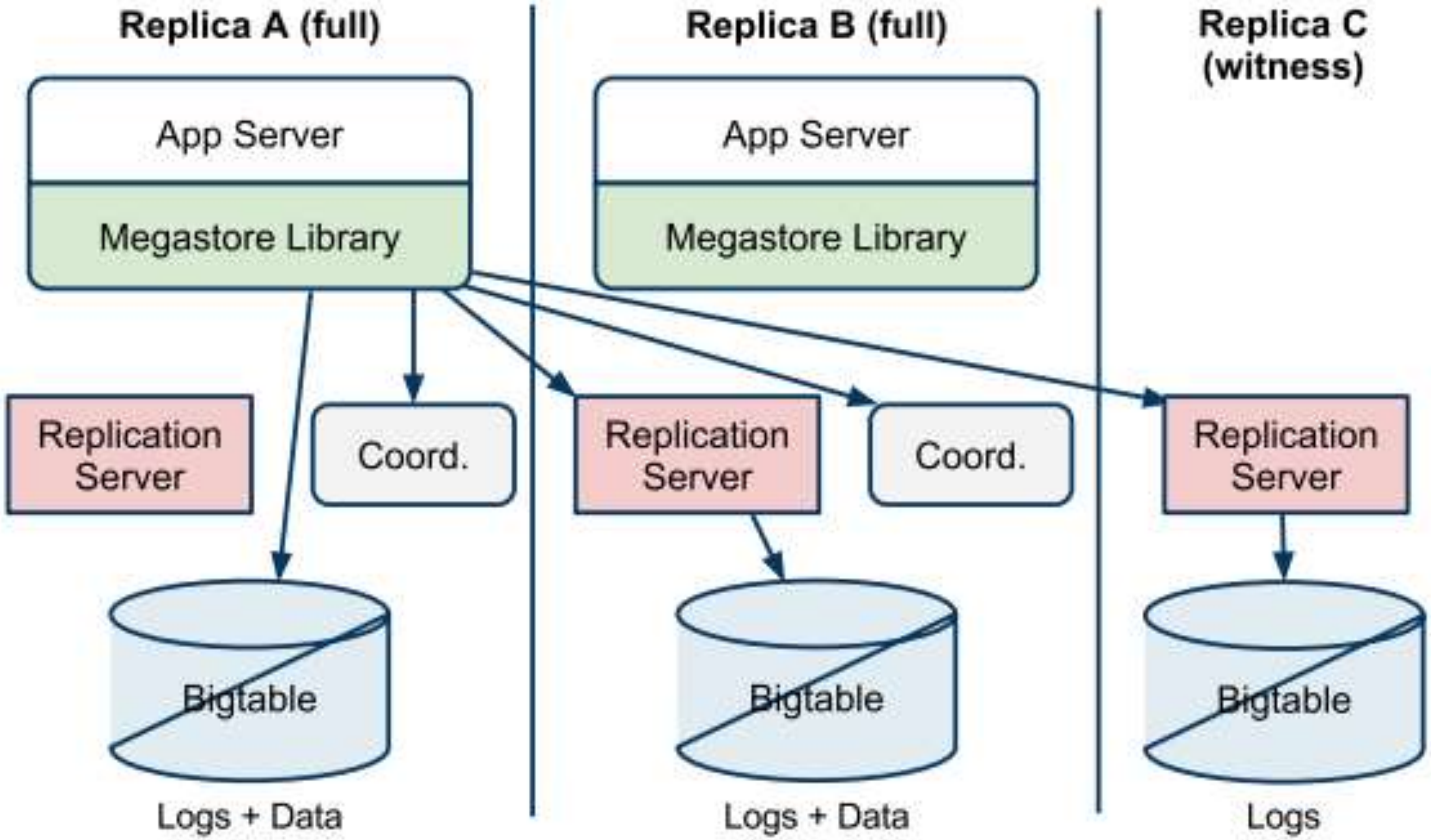
- Approaches using a Master replica
  - Master participates in all writes (state is always **up-to-date**)
  - Master serves reads (current consensus state) ***without additional comm***
  - Writes are **single roundtrip** – piggyback *prepare* for next write on *accepted*
  - Batch writes for efficiency

- Issues with using a Master
  - Need to place transactions (readers) **near master** replica to avoid latency
  - Master must have **sufficient processing resources** (side effect: replicas waste resources since they must be capable of becoming masters)
  - Master failover **requires lots of timers and a complex state machine** (side effect: user visible outages)

Megastore’s Approach

- Coordinators
  - Tracks set of entity groups for which its replica has observed all Paxos writes
- Fast Reads
  - Local reads from *any* replica avoid inter-replica RPCs (remote procedure call)
  - Yield better utilization, low latencies in all regions, fine-grained read failover, simpler programming experience
- Fast Writes
  - Uses same pre-preparing optimization as Master approaches (*accepted* implies next *prepare*)
  - Uses leaders (*coordinators*) instead of masters and runs a Paxos instance for each log position – leader arbitrates which writer succeeds
- Replica Types
  - ***Witness Replicas***: participate in voting (tie-breakers) and store log entries (no data)
  - ***Read-only Replicas***: non-voting replicas containing snapshots

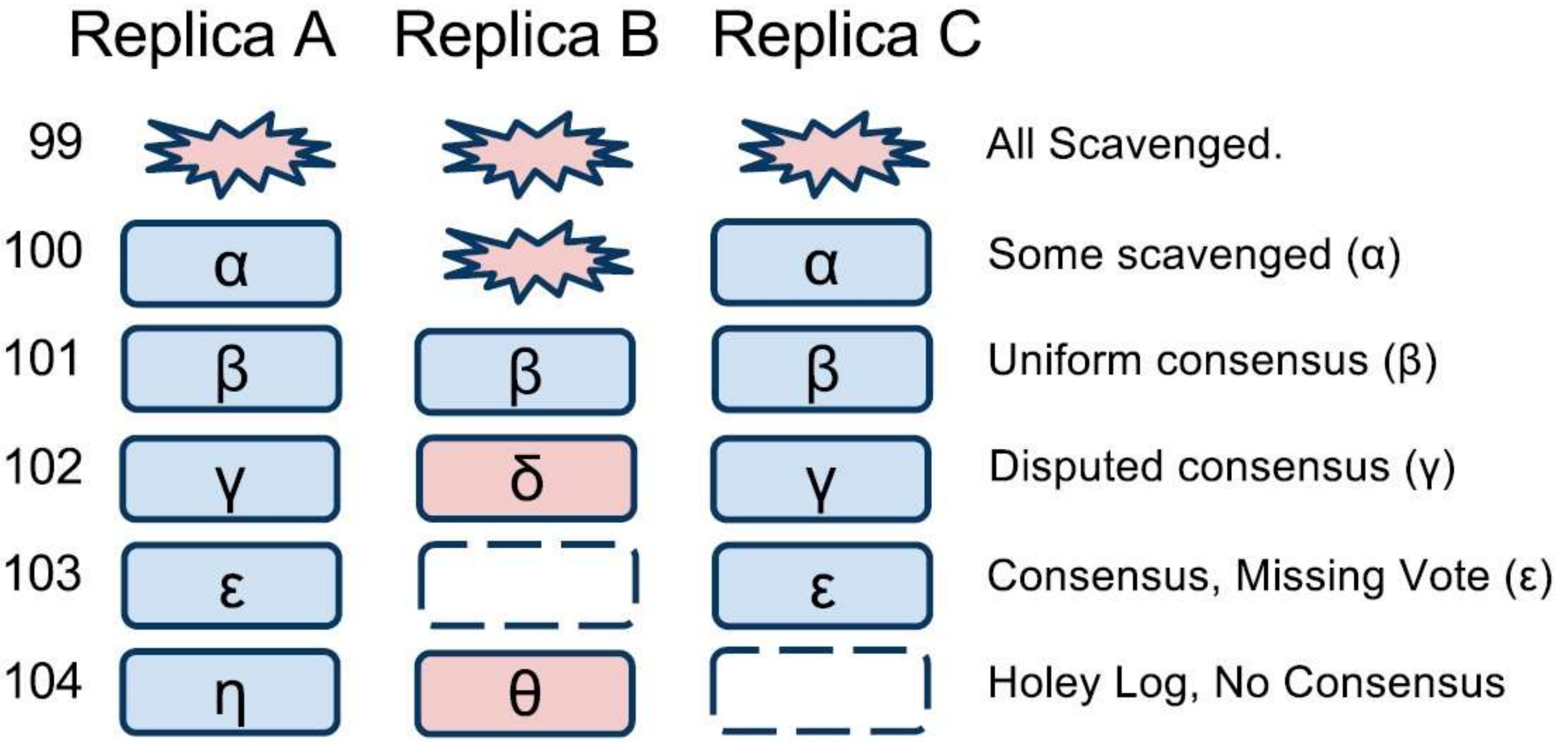
Megastore Architecture



Architecture

- Megastore is deployed through a **client library** and **auxiliary servers**.
- Applications link to the client library, which implements Paxos and other algorithms: selecting a replica for read, catching up a lagging replica, and so on.
- Each application server has a designated local replica.
- To minimize **wide-area roundtrips**, the library submits **remote Paxos** operations to stateless intermediary replication servers communicating with their local Bigtables.
- Replication servers **periodically scan** for incomplete writes and propose no-op values via Paxos to bring them to completion.

Write Ahead Log



	Replica A	Replica B	Replica C	
99	✸	✸	✸	All Scavenged.
100	α	✸	α	Some scavenged (α)
101	β	β	β	Uniform consensus (β)
102	γ	δ	γ	Disputed consensus (γ)
103	ε	⬚	ε	Consensus, Missing Vote (ε)
104	η	θ	⬚	Holey Log, No Consensus

Read algorithm

1. **Query Local:** Query the local replica's coordinator to determine if the entity group is up-to-date locally.

2. **Find Position:** Determine the highest possibly committed log position, and select a replica that has applied through that log position.

   (a) (***Local read***) *If step 1 indicates that the local rep*lica is up-to-date, read the highest accepted log position and timestamp from the local replica.

   (b) (***Majority read***) *If the local replica is not up-to-*date, read from a majority of replicas to find the maximum log position that any replica has seen, and pick a replica to read from. We select the most respective or up-to-date replica, not always the local replica

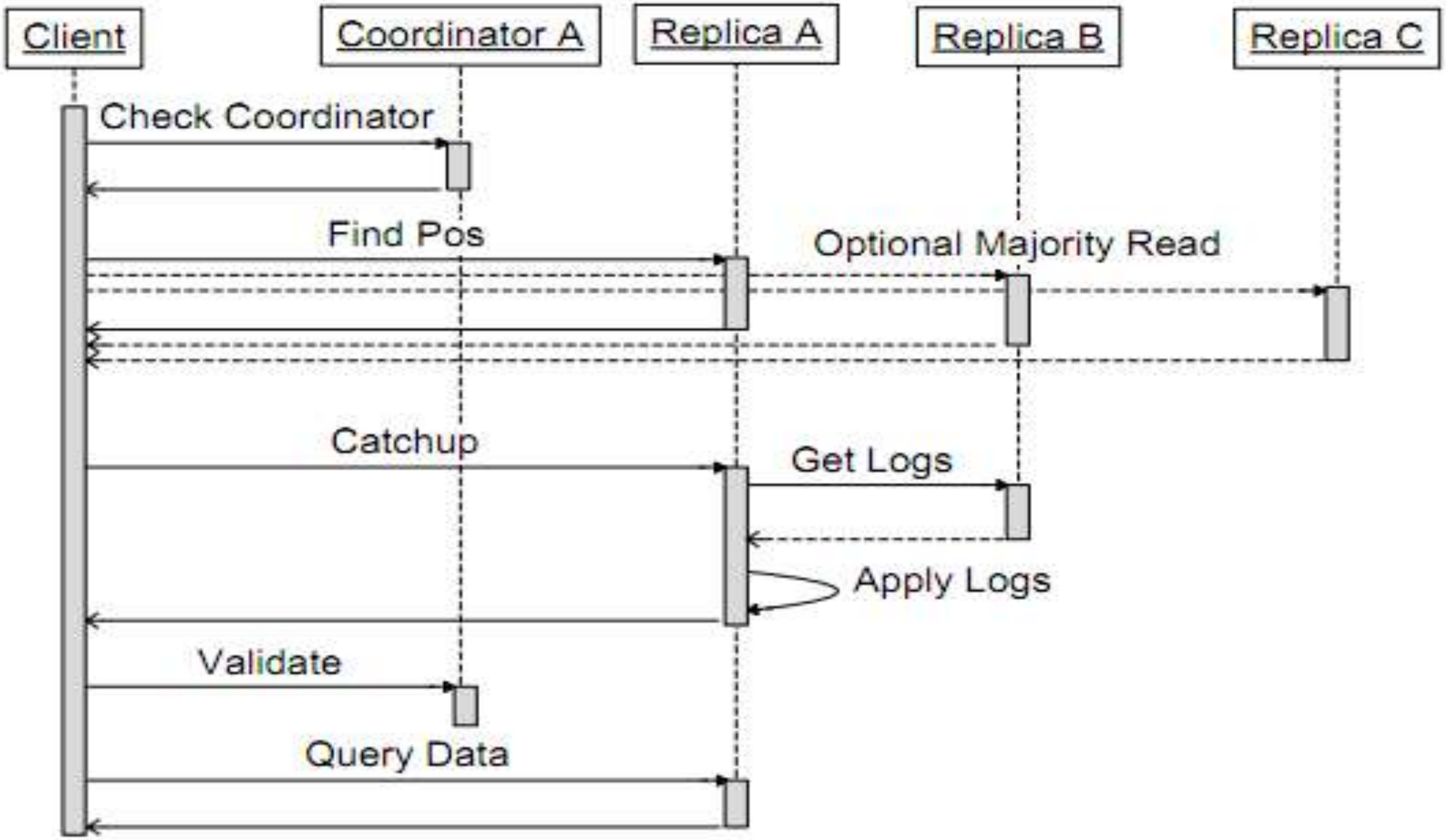
Read algorithm

3. **Catchup:** As soon as a replica is selected, catch it up to the maximum known log position

4. **Validate:** If the local replica was selected and was not previously up-to-date, send the coordinator a validate message.

5. **Query Data:** Read the selected replica using the timestamp of the selected log position. If the selected replica becomes unavailable, pick an alternate replica, perform catchup, and read from it instead.

Megastore Reads



Write algorithm

1. **Accept Leader:** Ask the leader to accept the value as proposal number zero. If successful, skip to step 3.

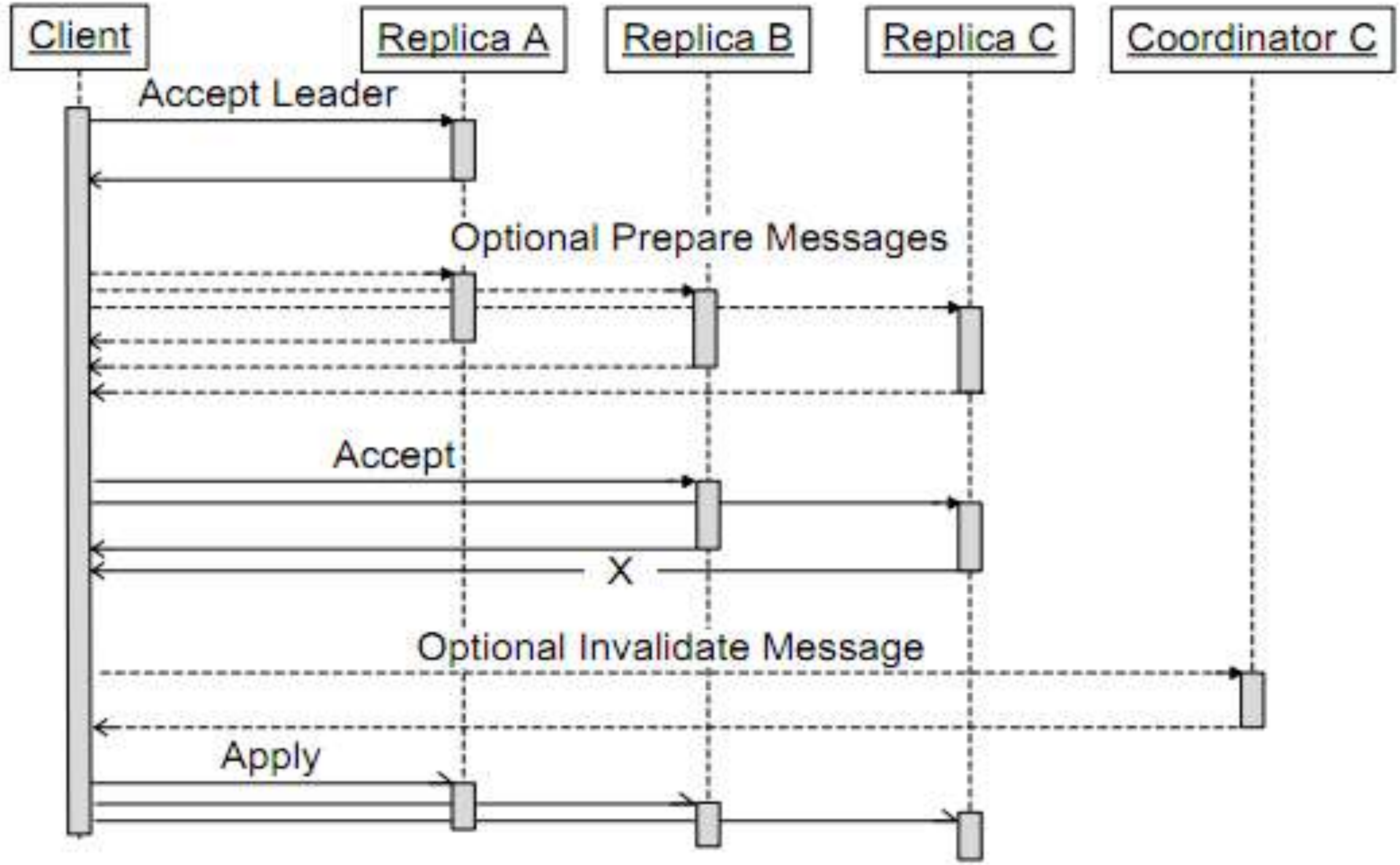
2. **Prepare:** Run the Paxos Prepare phase at all replicas with a higher proposal number than any seen so far at this log position. Replace the value being written with the highest-numbered proposal discovered, if any.

3. **Accept:** Ask remaining replicas to accept the value. If this fails on a majority of replicas, return to step 2 after a randomized backoff.

4. **Invalidate:** Invalidate the coordinator at all full replicas that did not accept the value.

5. **Apply:** Apply the value's mutations at as many replicas as possible. If the chosen value differs from that originally proposed, return a conflict error.

Megastore Writes



Production Metrics

- Megastore has been deployed within Google for several years, more than 100 production applications use it as their storage service.

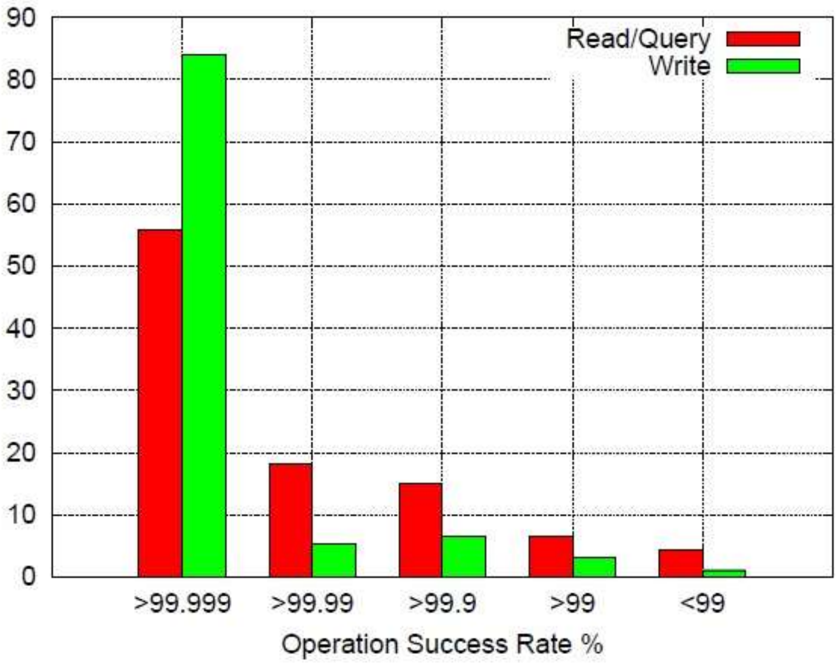
- Most of our customers see extremely high levels of availability.

- Observed average read latencies are **tens of milliseconds**, depending on the amount of data, showing most reads as local.

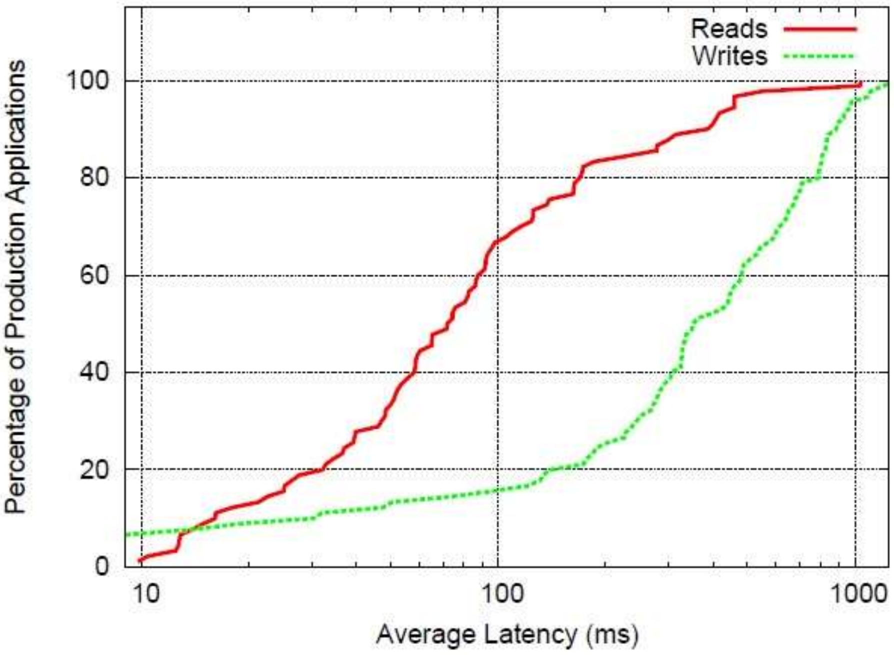
- Observed average write latencies are in **100-400 milliseconds** range, depending on the distance between datacenters, the size of data and number of full replicas.

Availability and Performance

The distribution of availability, measured on a per-application, per-operation basis



The distribution of average latency for read and commit operations



Benefits

- For admins
  - Linear scaling
  - Transparent failover
  - Symmetric deployment

- For developers
  - ACID transactions (read-modify-write)
  - Many features (indexes, backup, encryption, scaling)
  - Little need to handle failures

- For end Users
  - Fast up-to-date reads, acceptable write latency
  - Consistency

- **Available on Google App Engine as HRD (High Replication Datastore)**

Conclusion

- In this paper we present Megastore, a scalable, highly available datastore designed to meet the storage requirements of interactive Internet services.

- Megastore's over 100 running applications and infrastructure service for higher layers, is evidence of its ease of use, generality and power.