The Reading Report for Google Megastore

Lin Jinting, Chen Dian, Zhang Weiming, Liang Jiahui
{2014051795, 2014051796, 2014051797, 2014051786}

ABSTRACTION: Megastore is a storage system designed to meet the demand of today`s interactive online service. Therefore, it possesses the features of scalability and availability. Besides, it still provides fully supported ACID semantics for fast developing, as well as geographical distributed data partitioning storage system to keep data safety. In this paper, we discuss the basic concepts within Megastore and the core replication algorithms within Megastore as well as the deployment issues as supplementary.

Keywords: Megastore, Large database, Paxos, Distributed transaction, Bigtable

1. Introduction

As today`s interactive online services are developing at the speed of light, their natural characteristics had caused huge pressure on traditional cloud computing platform at 3 aspects: **development, deployment and administration** (See Figure 1-1). But, these seem like conflict requirements.

Take an example. 4 months after the launch event that WeChat which is published at 2011, it only takes 4 million users. But the number of active users grows to 800 million at the second quarter this year, what a tremendous growing speed.

From the aspect of service provider, in order to meet the strict requirements that originated from WeChat-like services and their light-speed development, the underlying cloud computing platform must be highly scalable to satisfy the endless growing users. It also requires rapid development to settle the attacks from other familiar services. Furthermore, it clients also demands low-latency respond as it is an instant chatting service. Finally, to keep the user-stickiness, it also has to guarantee the availability for their personal data.

But from the aspect of all three kinds of stakeholders, situations become complicated. Admins wants it to become easy-deployed with high-available policy. Developers wants it can be managed in an automatic way with consistent ACID transactions. And users want it high-available with good experience naturally.

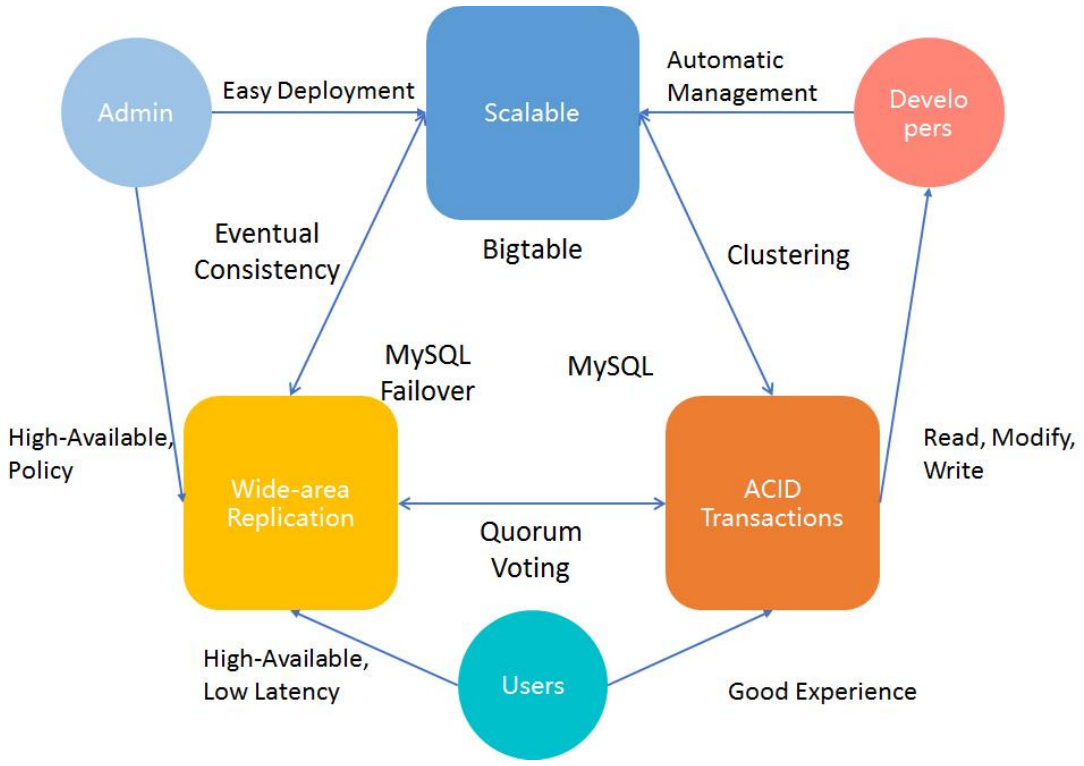


Figure 1-1 Requirements from all Stakeholders

To tackle all this seemingly conflict requirements, we had created tons of new technologies. For example, we use Bigtable to improve the growing scale, MySQL Failover for wide-area replication and MySQL for ACID transaction.

As far as we know, the more modules a single project has, the more complicated it will be. Therefore, a uniform platform integrating all the benefits within Bigtable, MySQL and MySQL Failover is necessary for the future development of interactive online services.

2. Availability and Scale

Unfortunately, the hardware is of limited scale and availability. What`s worse, the data is not stored in the same datacenter as before. Therefore, to settle down this problem, we have to focus on system design, so that the whole system can be bound in a proper way with minimum disadvantages.

So far, we had two practical ways to manage it:

1. **For Availability**

   We had implemented a synchronous, Fault-tolerant log replicator to optimized for long-distance data links (i.e., by Chubby).

2. **For Scale**

   We also introduced NoSQL data storage with its own replicated log (i.e. by Bigtable), to maximize the data scale.

2.1 Replication Algorithms

Before we actually step into the door of Megastore, we should have a brief understanding of current replication algorithms, in order to get a better understanding of Megastore`s approach.

2.1.1 Traditional Algorithms

Currently, we had three kinds of traditional algorithms:

1. **Asynchronous Master/Slave**

Master maintains the writes ahead log. If there are appends to the replication log, master will be acknowledged in parallel with slave through Message Queue(MQ). But it had an obvious weakness: if the target slave is down, data loss will occur. Therefore, an additional consensus protocol is needed.

2. **Synchronous Master/Slave**

Master informs its slaves after the changes are applied. But it also has its own weakness: needs an external system to keep the time.

3. Optimistic Replication

Mutations are propagating through the group asynchronous. As the order of propagation is unpredictable, it is impossible to implement transaction with such algorithm.

2.1.2 Paxos

In Paxos, there are no distinguished master and slave. It uses vote and catch up and to keep data consensus. This algorithm is selected by Google, and we will give a thorough explanation with examples to clarify the principles within it.

2.2 Entity Group

To get a complete understanding of Megastore, we have to understand the concept of Entity Group (EG).

An entity group is an abstract concept for a set of related data that stored in Megastore.

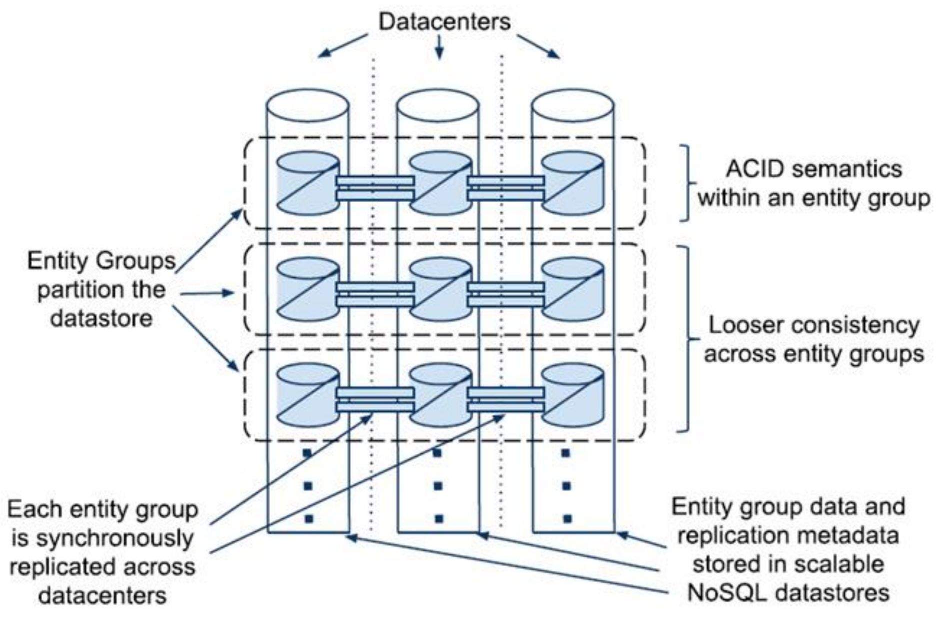

Datacenters
ACID semantics within an entity group
Entity Groups partition the datastore
Looser consistency across entity groups
Each entity group is synchronously replicated across datacenters
Entity group data and replication metadata stored in scalable NoSQL datastores

Figure 2-1 Scalable Replication

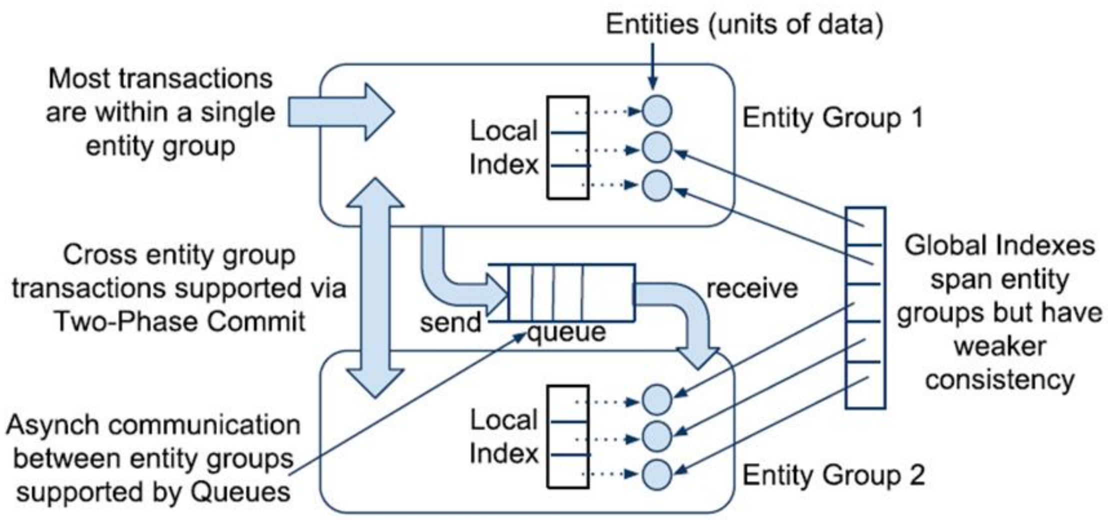
In Figure 2-1, we assume that there are three datacenters, which show as three vertical cylinders. Also, there are many EGs, but we just place three in this figure to make it simple and clear. All data are stored in different Bigtables in different datacenter.

Getting deep in it, we will see that, there are EG partitions, which store the different parts of data within an EG in some geographical distributed datacenters. Partitions can be the same copy of some instances within an EG, but it also can be part of an instance, which depends on the storage dispatch algorithm in Megastore.

Within a single EG, Megastore provides full ACID semantics, while a looser consistency across EGs, for unpredictable interventions during cross-EG committing.

2.3 Operations Across Entity Groups

Obvious, EGs need to communicate with each other to make a practical program. Up till now, we have two ways to transport data (messages) between EGs: **Two-Phase Commit** and **Message Queue.**



Entities (units of data)
Most transactions are within a single entity group
Local Index
Entity Group 1
Cross entity group transactions supported via Two-Phase Commit
send
queue
receive
Global Indexes span entity groups but have weaker consistency
Asynch communication between entity groups supported by Queues
Local Index
Entity Group 2

Figure 2-2 Communication between Entity Groups

This situation is limited within logically distributed EG, not geographical distributed replicas. The replication algorithm for geographical distributed replicas is much more complicated, we will discuss it in detail later.

2.3.1 Two-Phase Commit

Within a single database, we prefer to use Two-Phase Commit (2PC), to reach data consensus.

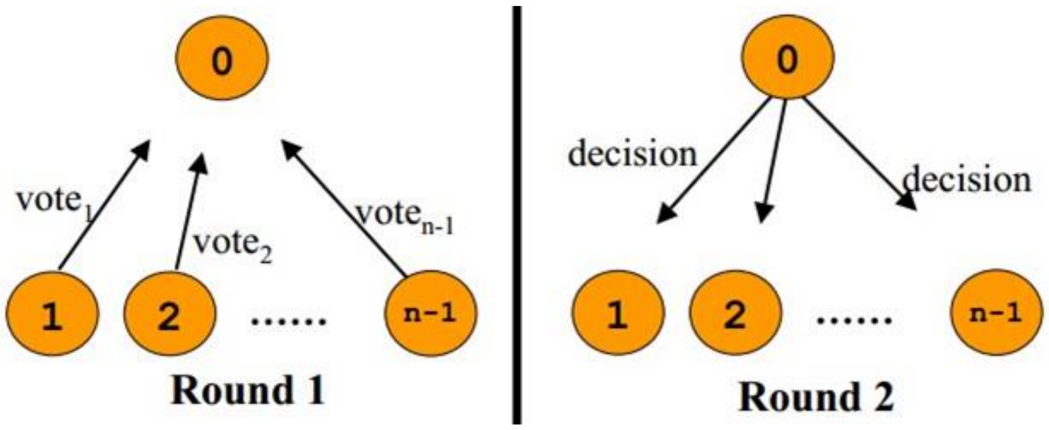


Figure 2-3 Two Phase Commit

Since 2PC require at least 2 rounds communication for **Voting Phase** and **Commit phase** (See Figure 2-3), it is unacceptable in a geographical distributed environment.

2.3.2 Asynchronous Message Queue

More conveniently, we use **Asynchronous Message Queue** to achieve our goal (See Figure 2-2).

Communicate with Asynchronous Message Queue requires less rounds. Sender only simply need to push their messages into the queue, and dispatcher will handle the rest.

2.4 Boundary of an Entity Group

To finish the final section for understand the Megastore, we still have to discuss another topic: how to define the boundary of an EG. If we want to make it clear, we have to understand what components can be put together in an EG first.

Take the email service as an example. Naturally, each email account is an EG: all of his emails and metadata. And emails from different accounts will be regarded as cross-EG transaction.

But when it comes to a map, the situation changes. Forms an EG by country or by continent will be a good choice. But since countries and continents are huge different in size, the size gap between EG will be wide too, which is bad for load balancing between datacenters. Therefore, we will break the map into several patches in the same size.

Fortunately, nearly all applications built on Megastore have found their proper ways to draw entity group boundaries.

3. Megastore

To some extent, Megastore is a database system that had overcome traditional disadvantages of RDBMS, as well as provides a set of brand-new features which motivates the fast-development at the same time.

3.1 Assumptions and Philosophy for API Design

Before we actually start to design the new service, we have to understand the situation we are facing.

1. Batched transactions suffer lesser performance lose when using Store Procedure than using query command directly.
2. Read dominates write, as most transactions require database read operations.
3. It is convenient to access the value with provided key in NoSQL storage platform like Bigtable.

Hence, we are able to draw our new design base on these facts (See Figure 3-1).

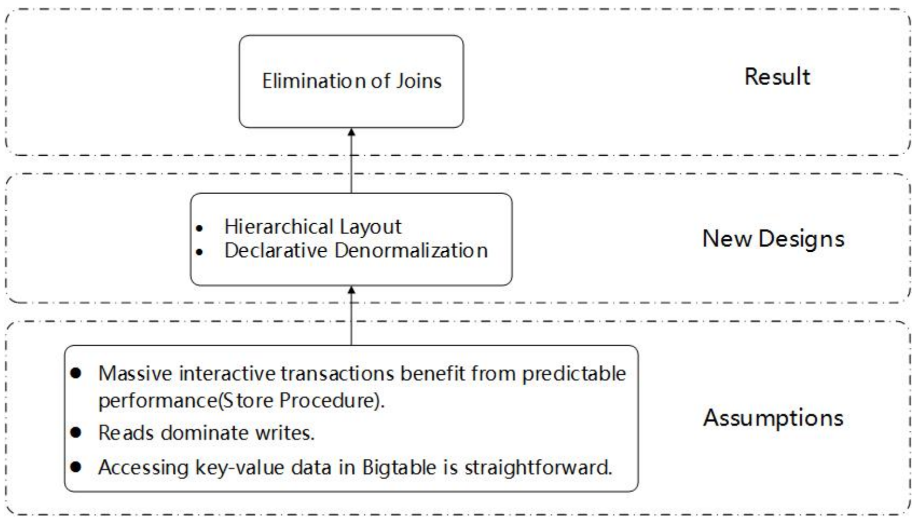
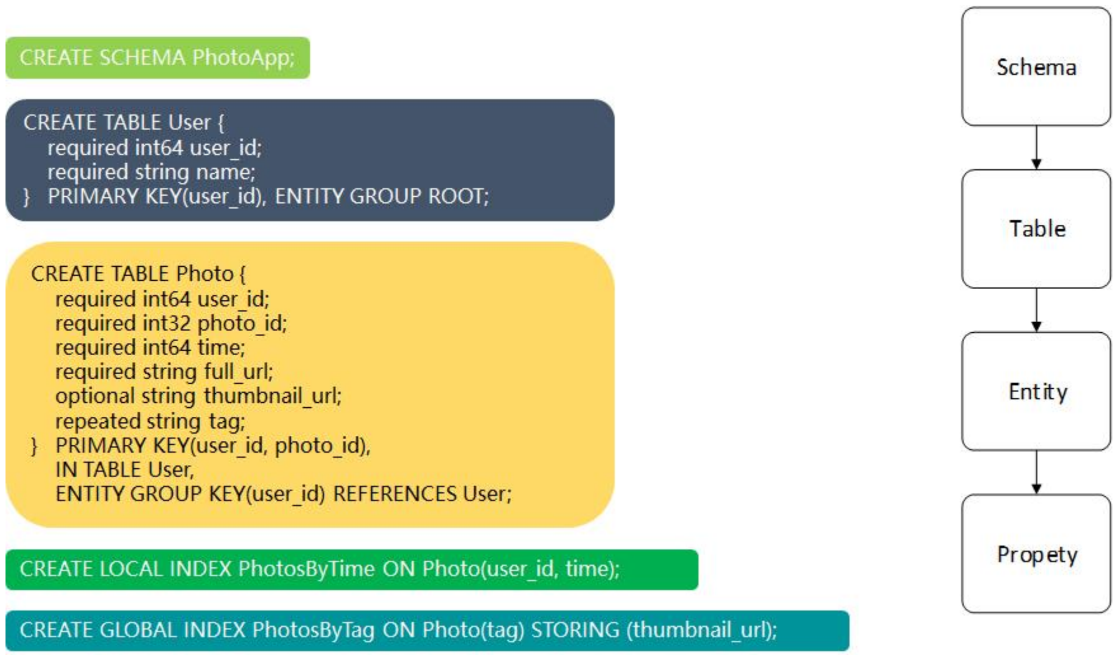

Elimination of Joins
Result
Hierarchical Layout
Declarative Denormalization
New Designs
Massive interactive transactions benefit from predictable performance(Store Procedure).
Reads dominate writes.
Accessing key-value data in Bigtable is straightforward.
Assumptions

Figure 3-1 Hierarchical Layout

With hierarchical layout, it is easy to construct the Megastore above Bigtable, and use the declarative demoralization to help making data storage mapped to Bigtable. Finally, the elimination of joins will be the last result.

3.2 Data Model with Example

With a hierarchical philosophy in mind, we had implemented a layered layout (See Figure 3-2).



```
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;

CREATE LOCAL INDEX PhotosByTime ON Photo(user_id, time);

CREATE GLOBAL INDEX PhotosByTag ON Photo(tag) STORING (thumbnail_url);
```

Figure 3-2 Data Model

Figure 3-2 is a data model sample which contains a 4-layer layout: **SCHEMA**,

TABLE, **ENTITY**, **PROPERTY**. In this sample, we created a SCEMA named PhotoApp, a user TABLE under PhotoApp, some Photo table, and 2 indexes.

3.2.1 Elimination of Joins

All tables have their own Primary Keys (PKs). In table User, the PK is user_id. In table photo, the PKs is photo_id and user_id, which refers to the user_id in table User.

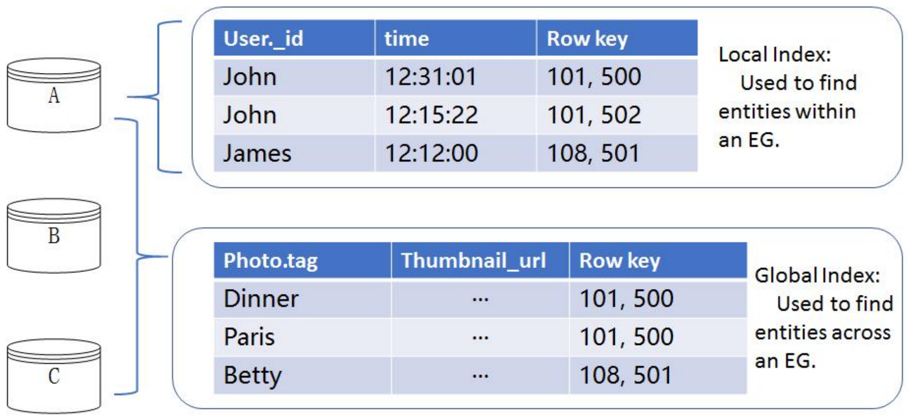
Traditionally, it will result in a join operation when a query request demand for information in table Photo. But as we want it become simple and clear, we implement it as Table 3-1, some photo instances follow their root user. Therefore, when the query for information in table Photo, Megastore can directly get to target rows with the help of local index. Hence, the existence of joins is eliminated.

Table 3-1 Data Stored in Bigtable

Row Key	user.name	photo.time	photo.tag	photo.url	…
101	John				
101, 500		12:31:01	Dinner, Paris	…	…
101, 502		12:15:22	Betty, Paris	…	…
102	Mary				
103	Jane				
103, 19		08:32:11	Office	…	…

3.2.2 Indexes

In Megastore, we provide two types of indexes in general: **global index** and **local index**. Just as their names implies, global index is responsible for the cross-EG query, and the local index is responsible for the local query (See Figure 3-2).


A
B
C
Local Index: Used to find entities within an EG.
Global Index: Used to find entities across an EG.

User._id	time	Row key
John	12:31:01	101, 500
John	12:15:22	101, 502
James	12:12:00	108, 501

Photo.tag	Thumbnail_url	Row key
Dinner	···	101, 500
Paris	···	101, 500
Betty	···	108, 501

Figure 3-2 Indexes

Apart from the well-known global index and local index, Megastore also provide additional index features.

1. Storing Clause

In global index PhotoByTag (See Figure 3-1), we can store a little additional information for the convenience of retrieval. For example, we store the thumbnail_url with the index itself. Therefore, when application server query for the thumbnail_urls, it is unnecessary for Megastore to search in Photo explicitly in table Photo, with rounds of communication saved.

2. Repeated Indexed

Each repeated tag has its own index entry (See Figure 3-1).

3. Inline Indexes

Extracting slices of info from child entities and storing it in the parent for fast access.

With all these elements stored in the same Bigtable in an optimized way, the actual data storage should look like this (See Table 3-1).

3.3 ACID Semantics

It is necessary for a database system to provide fully supported ACID Semantics. Therefore, Megastore uses Write Ahead Log (WAL) and Multi-Version Concurrency Control (MVCC) to guarantee the ACID semantics, as well as different types of reads to satisfy different requirement.

3.3.1 Write Ahead Log

Megastore will write it before apply the changes. It can be used for fail recovery or transaction rollback. We will detail the WAL in later section.

3.3.2 Multi-Version Concurrency Control

MVCC is the key technology to keep the ACID semantics in Megastore. In Megastore, Different values can be stored in a single Bigtable cell, with their timestamps attached (See Table 3-2). And reader uses timestamps to identify the latest value for target property in a fully updated transaction.

Table 3-2 Sample for MVCC

Photo.tag
[(Dinner, Paris), 12:30:01], [(Father, Mother), 12:31:01]
[(Betty, Paris), 12:15:22], [(Betty), 12:16:22]

What`s more, reads and writes are isolated, as there are multiply versions. If a writer is appending the latest value to Bigtable, reads will fetch the one version older value. But still the latest value until the writer finishes.

3.3.3 Reads

In Megastore, we provide three types of reads to meet the different from different application logics.

1. **Current Read**

Always done within a single EG. Read info before confirming all previous transactions are applied.

2. **Snapshot Read**

Picks up the latest known fully applied version, though there may be some transactions waiting for applied, for example, transactions delayed in Asynchronous Message Queue for network problems.

3. **Inconsistent Read**

Reads the value in Bigtable directly regards the log status.

Their differences can be seen in Table 3-3.

Table 3-3 Difference in all Reads

Type	Data Consensus	Latency
Current Read	High	High
Snapshot Read	Medium	Medium
Inconsistent Read	Low	Low

Hence, under MVCC and WHL as well as different reads, the complete lifecycle for a transaction should looked like this:

1. **Current Read**

   Uses a current read to determine the next available log position.

2. **Application Logic**

   Prepare the data to be written together, and designate it a latest log position. Also, batching writes to a front-end server can reduce the possibility of contention.

3. **Commit**

   Client submit mutations and the server will use Paxos to vote a consensus value across all replicas.

4. **Apply**

   Write mutations that win the Paxos procedure into the Bigtable, with its own timestamp attached.

5. **Clean Up**

   Clean all unnecessary values. For example, older version of an updated value.

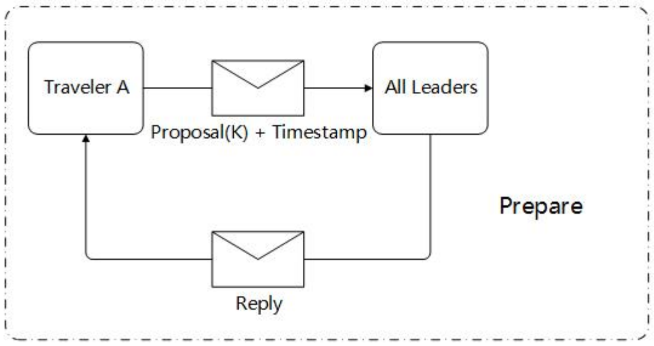
4. Replication

In this section, we will focus on our implementation of Paxos which is the heart of our synchronous replication scheme. After that, we will turn to operational issues and measurement.

4.1 Original Paxos

A way to reach consensus among a distributed system on a given value by winning more than half votes. There are 2 phases: Prepare and Accept.

Suppose 25 travelers need to reach a consensus about where to go with 5 additional leaders. All travelers will send their recommendation on where to go to all leaders, with timestamp attached (See Figure 4-1).



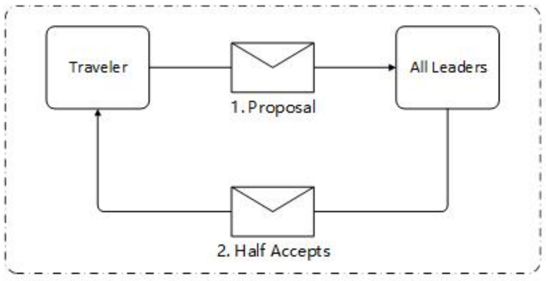
Leaders will only reply the latest message he receives.

With at least half of the leaders` accept(i.e., 3 leaders), this traveler can enter the Accept phase.

Figure 4-1 Prepare Phase

There are 2 possible situations if a traveler getting into the Accept Phase (See Figure 4-2).

1. **If none of the leaders had made decision. This traveler will send message to all leaders with his proposal (See the left side).**
   1) If more half leaders reach a consensus, this will be the decision.
   2) If it is the other situation, he has to retry from Prepare phase.
2. **At least 1 leader had made decision (See the right side).**
   1) If more half leaders reach a consensus, this will
   2) be the decision. If all leaders had not reached consensus, he will support the latest decision.



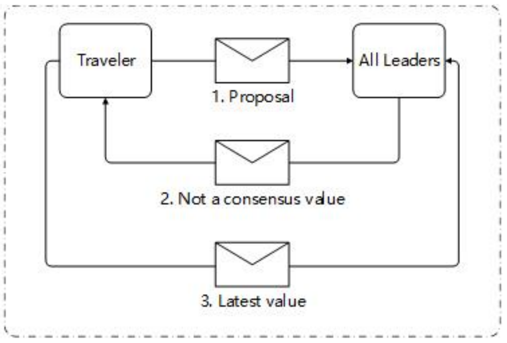


Figure 4-2 Accept Phase

Apparently, it is ill-suited for possible high latency network as it requires multiple rounds of communication.

4.2 Fast Read and Write in Megastore

To overcome the disadvantages that make implementing Paxos in Megastore impossible, we introduce two innovative measurements to do it.

4.2.1 Fast Read

As almost all reads are successful in most part of replicas, therefore, allowing local reads with lower latencies and better utilization is reasonable.

Also, to make it simple and safe, we introduce a service named Coordinator (See Fig.13). A coordinator is a server tracks a set of entity groups for which its replica has observed all Paxos writes [2]. In short, coordinator supervise all its slave servers and knows all.

4.2.2 Fast Write

We also implement the Fast Write in Megastore with features below.

1. **Implied prepare message**

   Each successful write implied a prepare message for next log position it needs to perform next write. So, 1 round for each subsequent writes is saved.

2. **Use the closest replica**

   Select the replica with most submitting in this region.

4.3 Replica Types

So far, we had implemented three types of replicas in Megastore: Full Replica, Witness Replica, and Read-Only Replica. And their differences will be list as table below.

Table 4-1 Differences between Replica Types

	Full Replica	**Witness Replica**	**Read-Only Replica**
Current Read	Yes	No	No
Log and Data Storage	All	Not-applied log, No data and indexes.	Full data snapshot
Vote	Yes	Yes	No
Usage	All	Tie breakers, Voting	Dissemination - CDN

4.4 Architecture

In Megastore, all key components are locating in different layers (See Figure 4-3).

1. **Application Layer:** Application Server, Megastore Library.
2. **Megastore Layer:** Replication Server, Coordinator.
3. **Physical Layer:** Bigtable Server with Logs and Data.

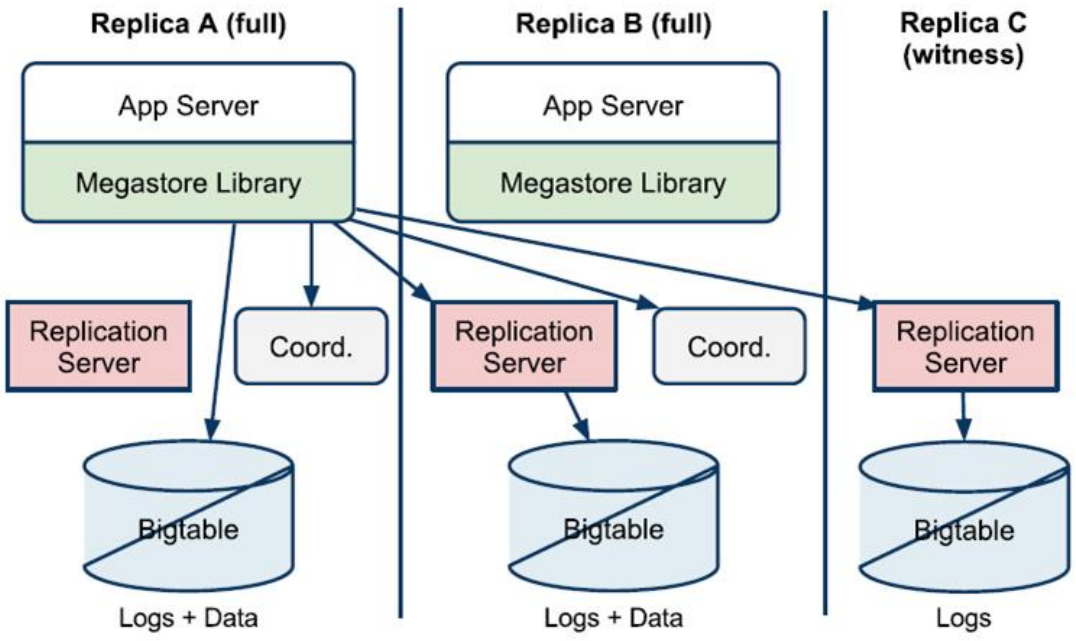


Figure 4-3 Architecture

4.5 Replicated Logs

The functioning of Megastore depends on the WHL we discuss in Section 3.3.1. Because there is numerous transaction submitted to Megastore, the status of WHL will be various (See Figure 4-4).

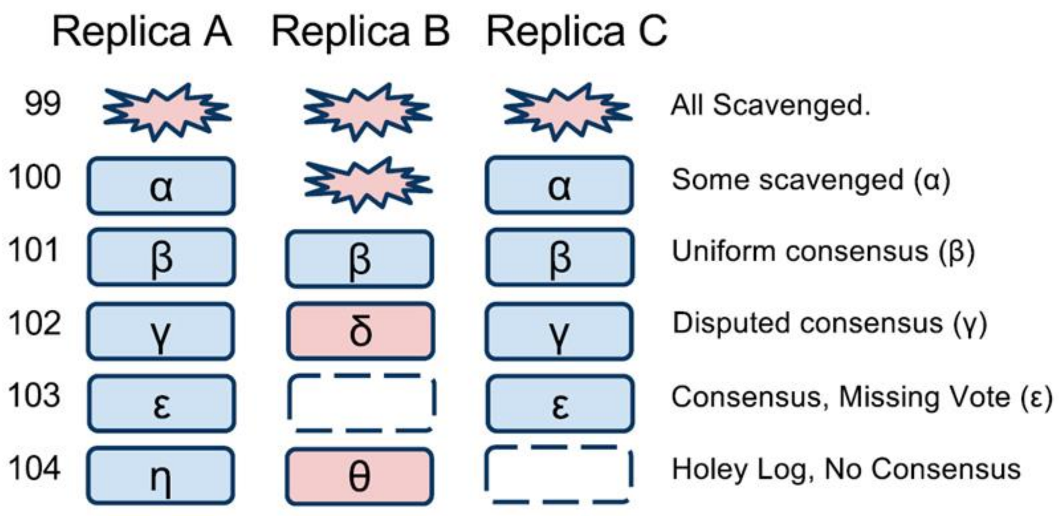


Figure 4-4 Write Ahead Log

Fig.14 shows 6 different status in different rows: shows 6 different statuses in different rows:

1. **Row 99**, all log position is outdated, therefore scavenged to save storage space.
2. **Row 100**, the scavenger is processing with one column cleared.
3. **Row 101**, all replicas reach consensus, therefore this log positon should not be scavenged until replace.
4. **Row 102**, under the Paxos process, some replicas had not agreed on a specific value proposed by other replicas.
5. **Row 103**, there may be problem with Replica B, therefore its voting is mission.
6. **Row 104**, probably in the Prepare Phase with different votes from all replicas with possible latency or disconnection.

4.6 Read and Write with Paxos in Megastore

Under the implementation Paxos, we had made modifications on our read and write algorithms to make them a highly available and low-latency service.

4.6.1 Reads

There are 5 steps for read algorithm in Megastore (See Figure 4-5):

1. **Query Local Determine if the entity group is up-to-date locally [2].**
2. **Find Position Get the highest log position, and select the corresponding replica.**
    1) Local read. Get the log position and timestamp locally.
    2) Majority read.
3. **Catchup**
    1) Get unknown value from other replica; run Paxos for any non-consensus values.
    2) Apply all consensus value, and push the state up-to-date.
4. **Validate Send its coordinator a message asserting itself up-to-date.**
5. **Query Data**

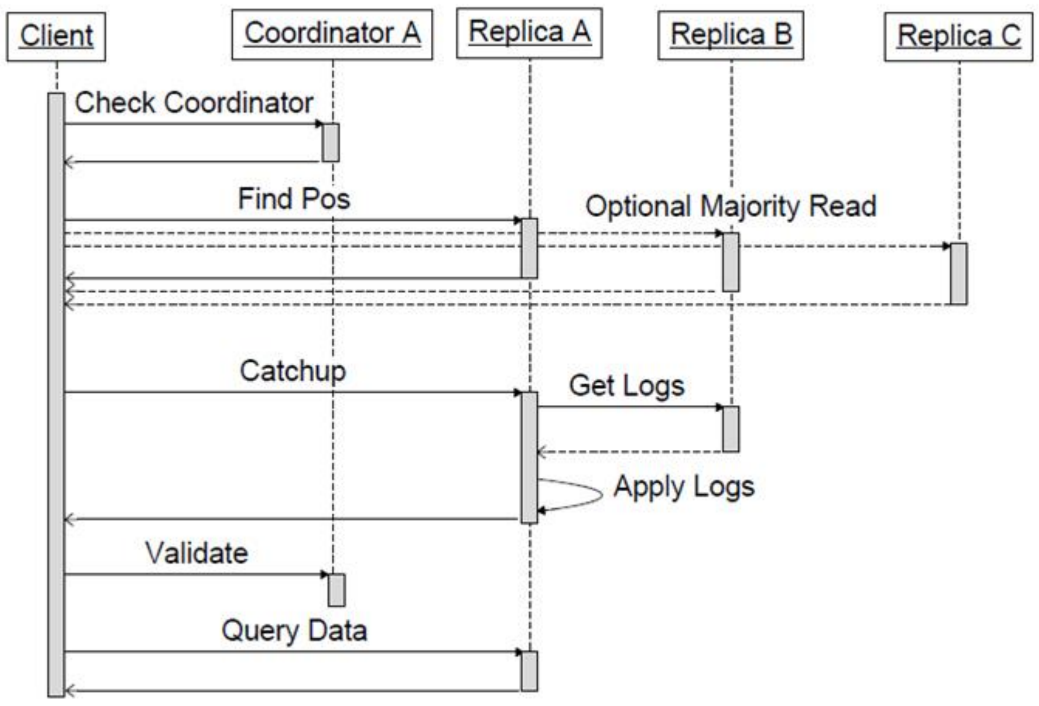


Figure 4-5 Timeline for Read Operation

4.6.2 Writes

There are 5 steps for write algorithm in Megastore (See Figure 4-6):

1. **Accept Leader.** Ask the leader to accept the value as proposal number zero [2].
2. **Prepare.** Run Paxos Prepare phase at all replicas [2].
3. **Accept.** Ask the rest of replicas to accept the proposal, as the Accept Phase in Paxos.
4. **Invalidate**. If a full replica doesn't accept this value, invalidate its coordinator.
5. **Apply.**

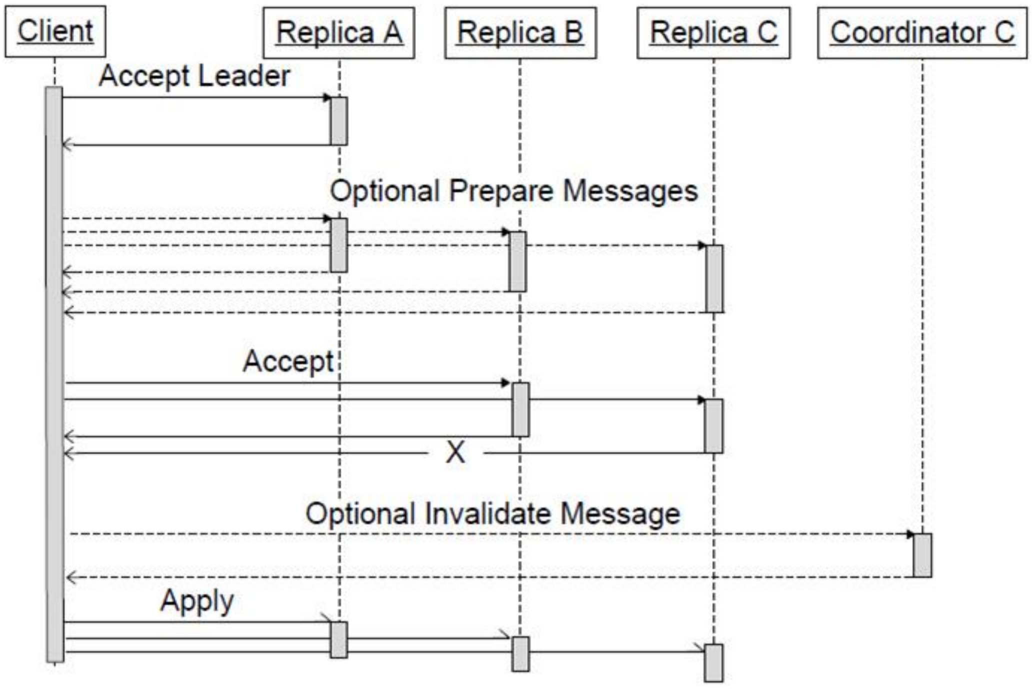


Figure 4-6 Timeline for Write Operation

4.7 Coordinator Availability

Coordinator is a simpler process than Bigtable with much more stability. But it still has the risk to crash or other situations that cause its unavailable. Therefore, readers and writers have different strategies against the instability introduce from coordinator:

1. **Reader**

   To process a request, a coordinator must hold a majority of its locks [2].

2. **Writer**

   Test the coordinator whether it still has locks before a writer submits its transactions.

4.8 Tests

In short, we had made tests on Availability and Latency.

In our test for availability, we observe an availability for at least 99% at almost all involved hosts in this test (See Fig.17), which proves the availability on Megastore.

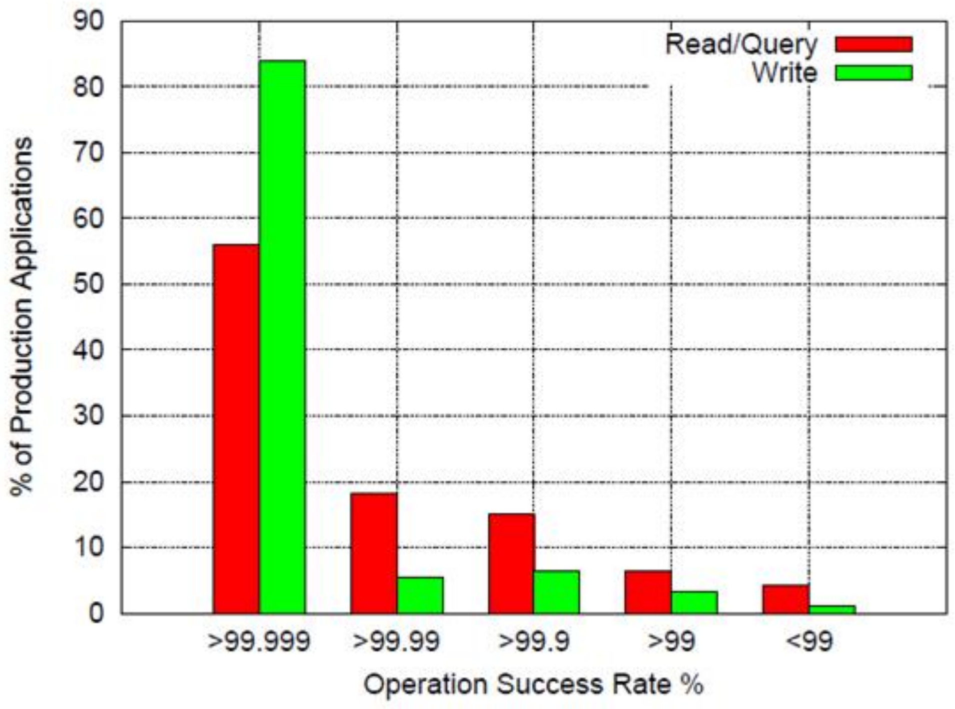


Figure 4-7 Availability

In our test for latency, we observe an average latency at 100ms for reads and 500ms for writes in more than half host involved in this test (See Fig.17), which proves the low-latency feature on Megastore.

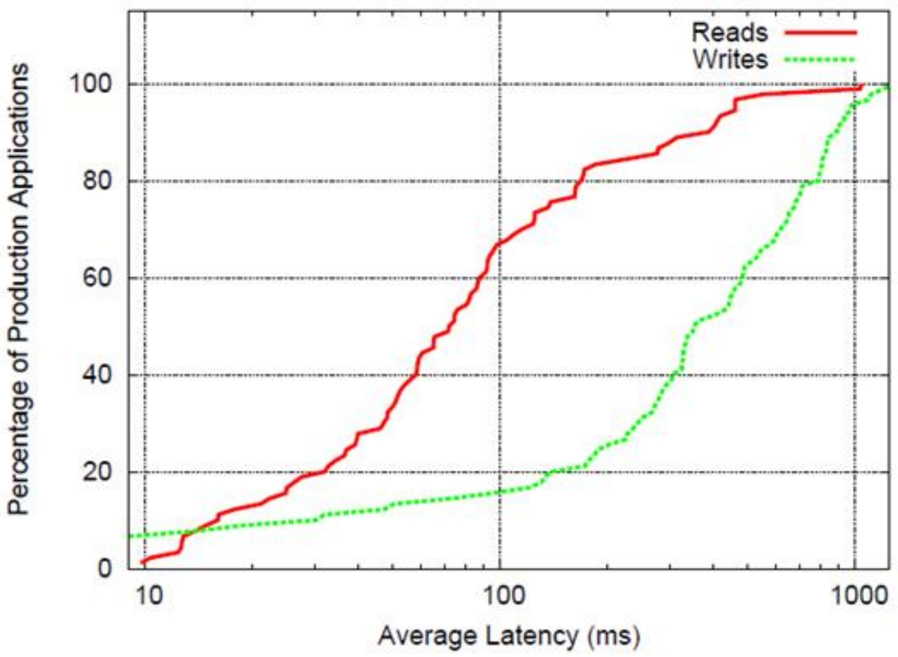


Figure 4-8 Latency

5. Conclusion

In this presentation, we make introduction for the following concepts: Entity Groups, Data Model and MVCC, Paxos in Megastore, Replication algorithm.

Since Megastore is the origin of many distributed systems, we truly hope our audience can fully understand the revolutionary concept and theory that Megastore brings to us.

One step further, keep the motivation for learning and innovating is the fundamental motive power towards truth.

6. Reference

[1] 演员，微信月活跃用户数量破 8 亿 ；2016 年腾讯每天收入 3 亿，https://buluo.qq.com/p/detail.html?bid=261433&pid=7574448-1471501699

[2] Baker J, Bond C, Corbett J, et al. Megastore: Providing Scalable, Highly Available Storage for Interactive Services.[C]// CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings. DBLP, 2011:223-234.