# The Reading Report for Google Megastore

**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 replication algorithms within Megastore as well as the deployment issues.

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

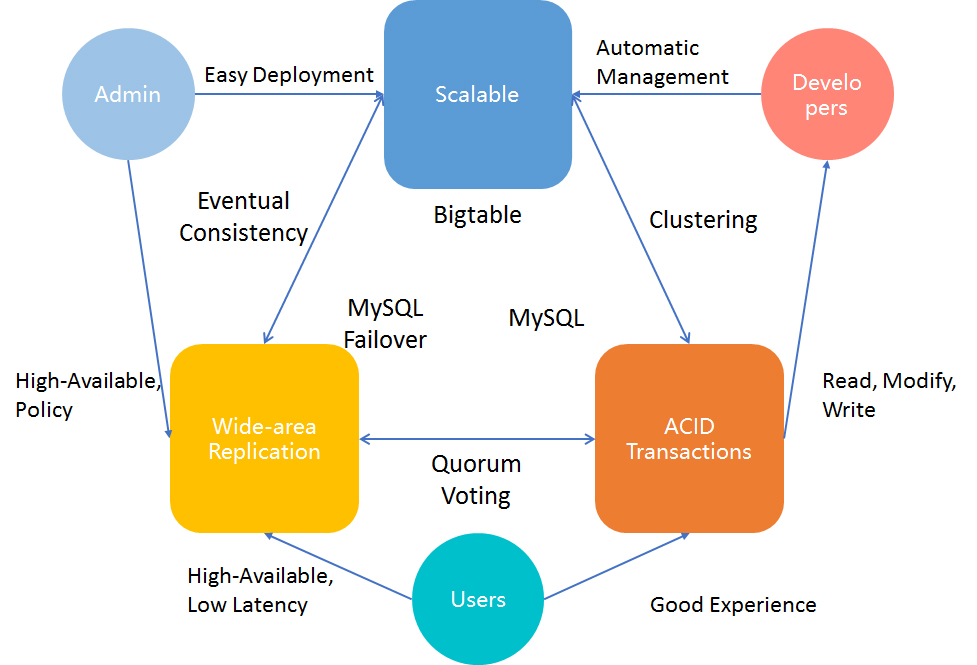
## Introduction

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. But, these seem like conflict requirements.

Take an example. 4 months after the launch event that WeChat 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 a 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 create 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, an uniform platform integrating all the benefits within Bigtable, MySQL and MySQL Failover is necessary for the future development of interactive online services.

## Availability and Scale

Unfortunately, the hardware are 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 an 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).

1. **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.

1. **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.

1. **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.

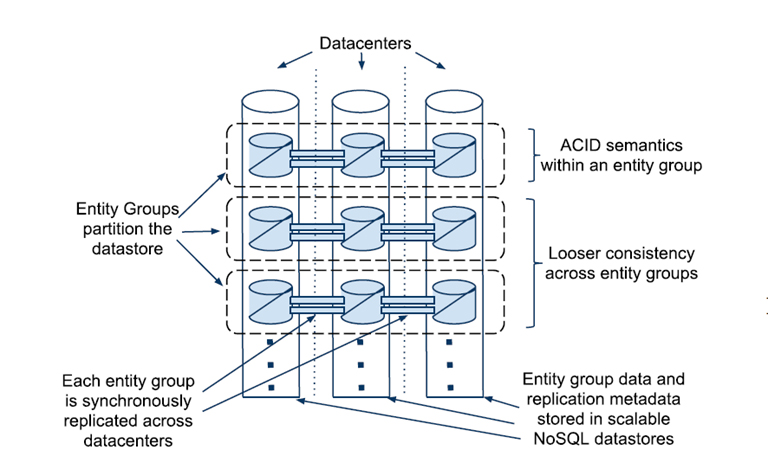
### 2.1.2 Paxos

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.



**Figure 2-1** Scalable Replication

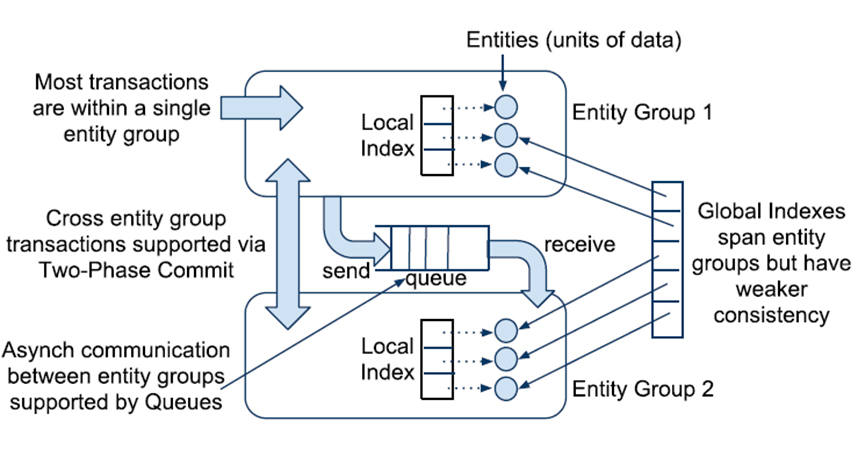
In Fig.2, we assume that there are three datacenters, which show as three vertical cylinders. Also, there are many EGs, but we just place three in this figure to make it simple and clear. All data are stored in a Bigtable within a datacenter.

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*.

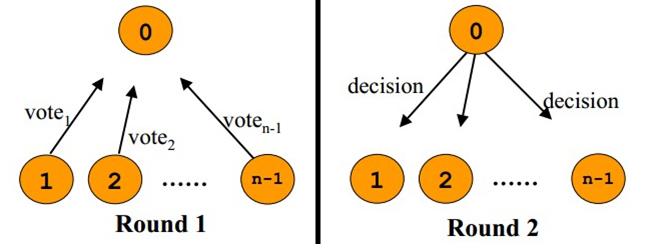


**Figure 2-2** Communication between Entity Groups

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(Fig.4), 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.3).

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

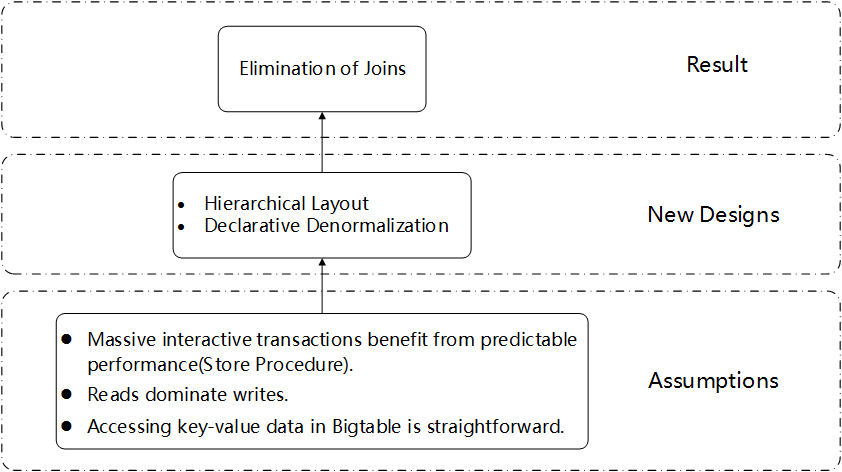
To some extent, Megastore is a database system that had overcome traditional disadvantages of RDBMS, as well as provides a set of brand-new features which motivates the fast-development at the same time.

### 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.5).



**Figure 2-4** 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