云计算（第三版）

CLOUD COMPUTING Third Edition

第 2 章

Google云计算原理与应用（四）

主编：刘鹏 教授

目录

2.5 分布式存储系统Megastore

- 2.5.1 设计目标及方案选择
- 2.5.2 Megastore数据模型
- 2.5.3 Megastore中的事务及并发控制
- 2.5.4 Megastore基本架构
- 2.5.5 核心技术——复制
- 2.5.6 产品性能及控制措施

可用性

可扩展性

- **设计目标及方案选择**

设计目标

设计一种介于传统的关系型数据库和NoSQL之间的存储技术，尽可能达到高可用性和高可扩展性的统一。

关系型数据库：高事务性和复杂数据查询

NoSQL数据库：扩展性和大数据处理能力

【思考】Metastore和Bigtable之间有什么联系和区别？

- **设计目标及方案选择**

设计目标

设计一种介于传统的关系型数据库和NoSQL之间的存储技术，尽可能达到高可用性和高可扩展性的统一。

方法一

针对**可用性**的要求，实现了一个同步的、容错的、适合远距离传输的复制机制。

Paxos

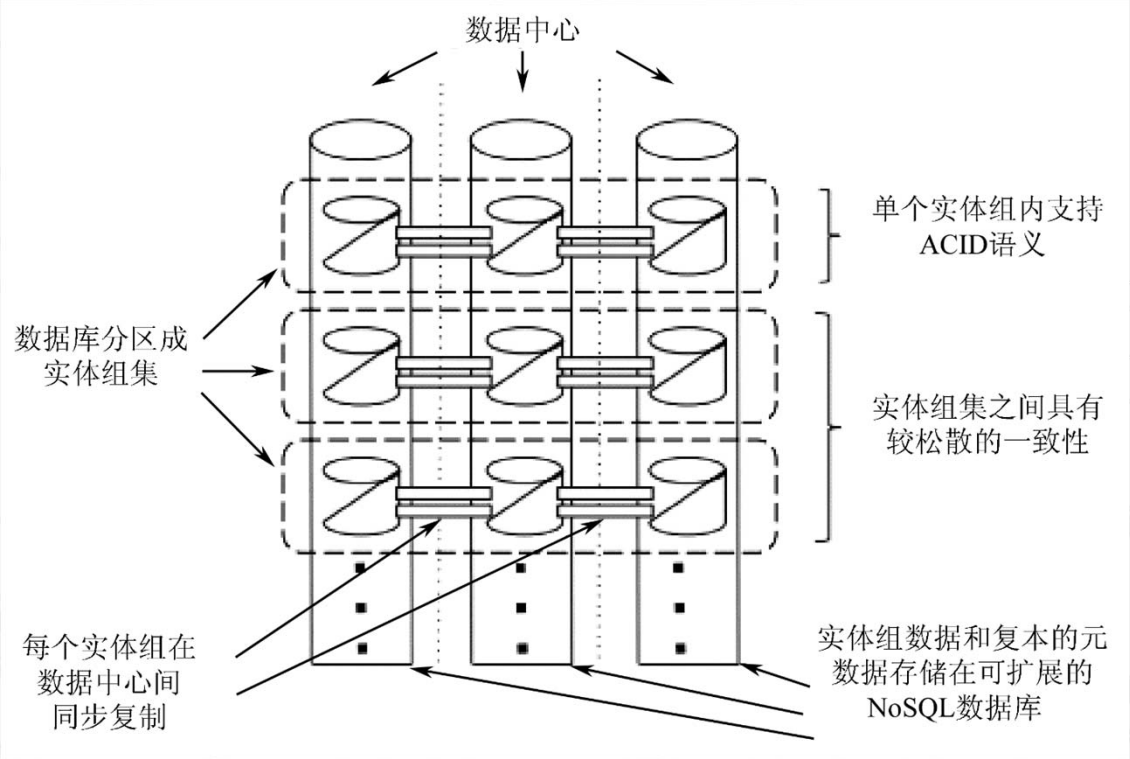
方法二

针对**可扩展性**的要求，将整个大的数据分割成很多小的数据分区，每个数据分区连同它自身的日志存放在NoSQL数据库中，具体来说就是存放在Bigtable中。

子表(tablet)　SSTable

数据的分区和复制

- 在Megastore中，这些小的数据分区被称为实体组集（Entity Groups）。
- 每个实体组集包含若干的实体组，实体组相当于表的概念，多数情况下每个实体组仅有一个子表，分布在一台Bigtable的子表服务器上，若实体组比较大，导致子表分裂，会分布到多台子表服务器上。
- 一个实体组中包含很多的实体（Entity，相当于表中记录的概念）。



ACID：原子性，一致性，隔离性和持久性

1. 同一个Entity Group的数据可以被划分到Bigtable表的同一个数据分区(子表，Tablet)中，这样，同一个Entity Group内的跨行事务是可以轻松支持的。

2. 同一个Entity Group中的数据是相邻存放的，这样可以将多表的关联查询转换为一个顺序的Scan，加速查询。同时，对于数据写入而言，同一个Entity Group中的多条数据在一起写入时，也可有效的降低随机IO数目来提升写入性能。

3. 跨集群数据复制时，一个Entity Group在不同的集群间的数据可保持ACID语义的；

4. Entity Group可以理解为是比Bigtable Tablet（子表）更细粒度的分区，更细粒度意味着并发锁的范围可以更小，从而可以带来更高的并发度。

2.5 分布式存储系统Megastore

- 2.5.1 设计目标及方案选择
- ▶ 2.5.2 Megastore数据模型
- 2.5.3 Megastore中的事务及并发控制
- 2.5.4 Megastore基本架构
- 2.5.5 核心技术——复制
- 2.5.6 产品性能及控制措施

传统的关系型数据库不合适的三个原因

传统的关系型数据库是通过连接（Join）来满足用户的需求的，但是就Megastore而言，这种数据模型是不合适的，主要有以下三个原因：

原因 1	对于高负载的交互式应用来说，可预期的性能提升要比使用一种代价高昂的查询语言所带来的好处多
原因 2	Megastore所面对的应用是读远多于写，因此好的选择是将读操作所需要做的工作尽可能地转移到写操作上
原因 3	在Bigtable这样的键/值存储系统中存储和查询级联数据（Hierarchical Data）是很方便的


Megastore数据模型
怎么设计？

- **数据模型**

同关系型数据库一样，Megastore的数据模型是在模式（schema）中定义的且是强类型的（strongly typed）

Google团队设计的
Megastore
数据模型

模式：包含对象，可以是表、列、数据类型、主键、外键等。可用一个可视化图来表示，显示了数据库对象及其相互之间的关系

强类型：任何变量在使用的时候必须要指定这个变量的类型，并且在程序的运行过程中这个变量只能存储这个类型的数据

- **数据模型**

Google团队设计的
Megastore
数据模型

同关系型数据库一样，Megastore的数据模型是在模式（schema）中定义的且是强类型的（strongly typed）

每个模式都由一系列的表（tables）构成，表又包含有一系列的实体（entities），每个实体中包含一系列属性（properties）

属性是命名的且具有类型，这些类型包括字符型（strings）、数字类型（numbers）或者Google的Protocol Buffers。
这些属性可以设置成必需的（required）、可选的（optional）或可重复的（repeated）

- 照片共享服务数据模型实例

```
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);
```

Megastore中，所有的表要么是实体组根表（Entity Group Root Table），要么是子表（Child Table）

Megastore实例中可以有若干个不同的根表，表示不同类型的实体组集

所有的子表必须有一个参照根表的外键，该外键通过ENTITY GROUP KEY声明

- **照片共享服务数据模型实例**

```
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);
```

- User是一个根表
- Photo就是一个子表，因为它声明了一个外键
- 三种不同属性设置，既有必须的（如user_id），也有可选的（如thumbnail_url），还有可重复的（如tag）

- Megastore索引

```
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);
```

主要两类

局部索引：定义在单个实体组中，作用域仅限于单个实体组（ 如PhotosByTime ）

全局索引：可以横跨多个实体组集进行数据读取操作（ 如PhotosByTag ）

- Megastore索引

```
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);
```

额外索引

- STORING子句（STORING Clause）
- 可重复的索引（Repeated Indexes）
- 内联索引（Inline Indexes）

- Bigtable中存储情况

行键（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			

Bigtable的列名实际上是表名和属性名结合在一起得到，不同表中实体可存储在同一个Bigtable行中

2.5 分布式存储系统Megastore

- **Megastore提供的三种读**

实体组相当于表的概念，每个实体组一般只有一个子表

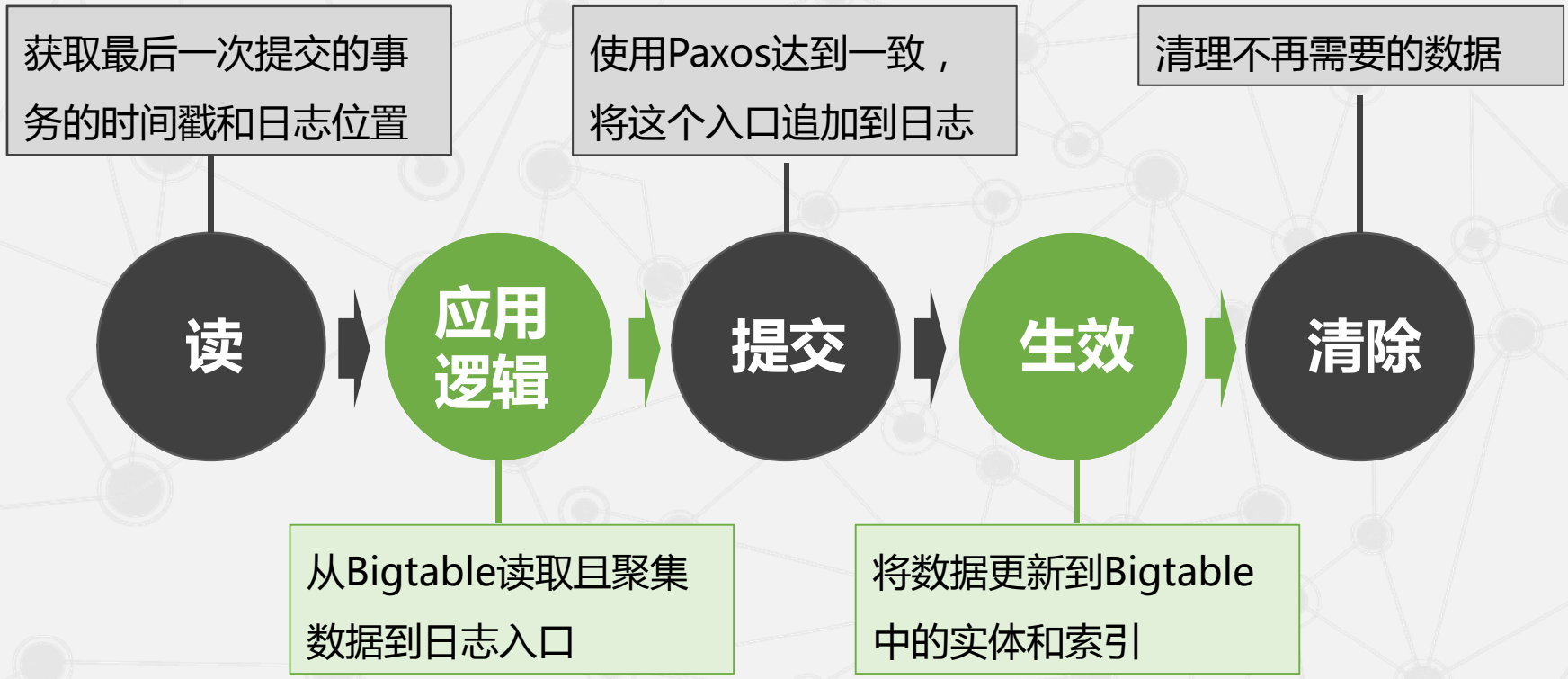
current	● 总是在单个实体组中完成 ● 在开始某次current读之前，需要确保已提交的写操作已经全部生效，然后应用程序再从最后一个成功提交的事务时间戳位置读数据
snapshot	● 总是在单个实体组中完成 ● 系统取出已知的最后一个完整提交的事务的时间戳，接着从这个位置读数据。 ● 读的时候可能还有部分事务提交了但还未生效
inconsistent	● 忽略日志的状态直接读取最新的值 ● 适用于要求低延迟并能忍受数据过期或不完整的读操作

Megastore的写操作

预写式日志	● 只有当所有的操作都在日志中记录下后，写操作才会对数据执行修改。 ● 一个写事务总是开始于一个current读，以便确认下一个可用的日志位置 ● 提交操作将数据变更聚集到日志，接着分配一个比之前任意一个都高的时间戳，然后使用Paxos将数据变更加入日志中。

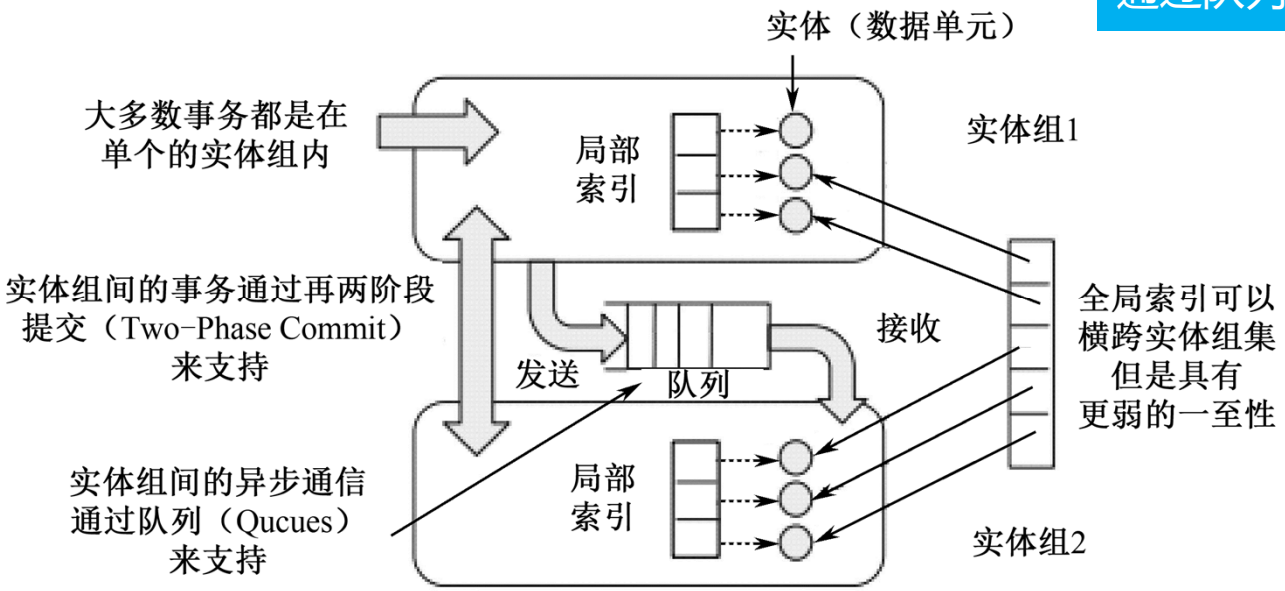
乐观并发（Optimistic Concurrency）

- **完整的事务周期**


获取最后一次提交的事务的时间戳和日志位置
使用Paxos达到一致，将这个入口追加到日志
清理不再需要的数据
读
应用逻辑
提交
生效
清除
从Bigtable读取且聚集数据到日志入口
将数据更新到Bigtable中的实体和索引

Megastore中的事务机制

Megastore中事务间的消息传递通过队列（Queue）实现



两阶段提交：请求阶段和提交阶段

- Megastore中的消息能够横跨实体组，在一个事务中分批执行多个更新或者延缓作业；
- 在单个实体组上执行的事务除了更新它自己的实体外，还能够发送或收到多个信息；
- 每个消息都有一个发送和接收的实体组；
- 如果这两个实体组是不同的，那么传输将会是异步的

2.5 分布式存储系统Megastore

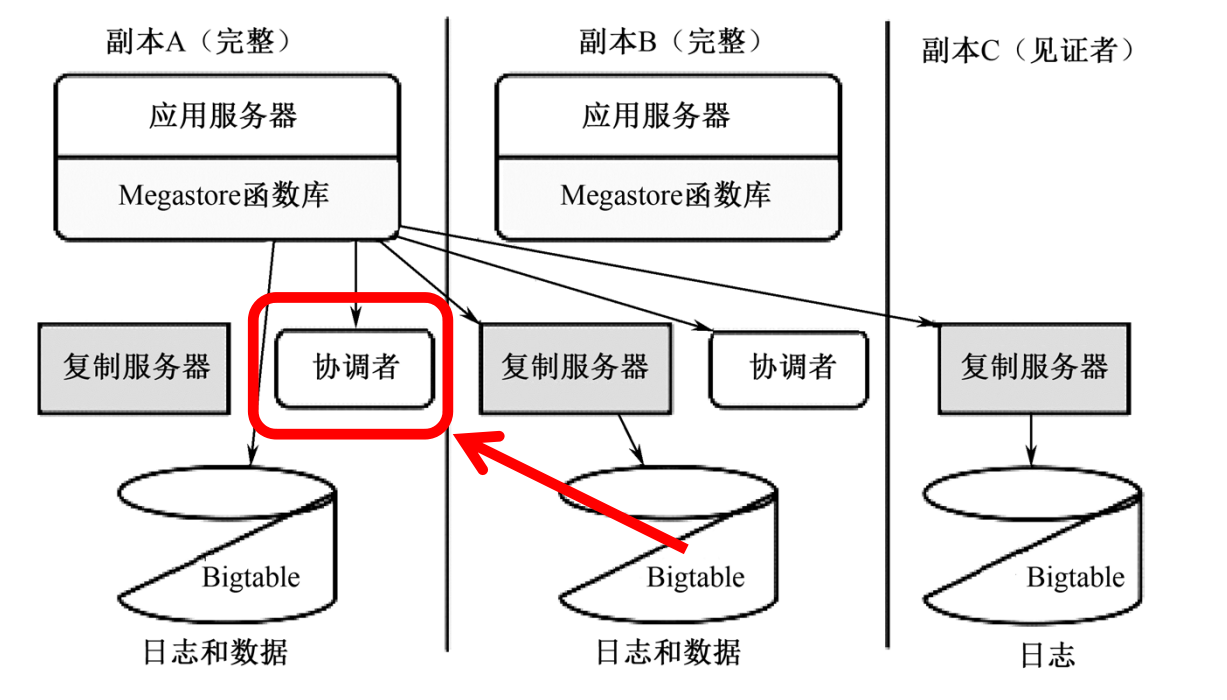
Megastore基本架构

在Megastore中共有三种副本

完整副本
(Full Replica)

见证者副本
(Witness Replica)

只读副本
(Read-only Replica)



完整副本：可投票，存储数据和日志
见证者副本：可投票，只存储日志
只读副本：不可投票，读取最近过去某个时间点的一致性数据，数据不是最新的

Megastore最底层的数据存储在Bigtable中，不同类型的副本存储不同的数据

快速读与快速写

快速读

由于写操作基本上能在所有副本上成功，一旦成功认为该副本上的数据都是相同的且是最新的，就能**利用本地读取实现**快速读，带来更好的用户体验及更低的延迟

关键是保证选择的副本上数据是最新的

协调者是一个服务，该服务分布在每个副本的数据中心里面。它的主要作用就是跟踪一个实体组集合，集合里的实体组的副本已经观察到所有的Paxos写。

协调者的状态是由写算法来保证

快速写

如果**一次写成功**，那么**下一次写**的时候就**跳过准备过程**，**直接进入接受阶段**

Megastore没有使用专门的主服务器，而是使用leaders

leader主要是来裁决哪个写入的值可以获取0号提议

复制服务器会定期扫描未完成的写入并通过Paxos算法提议没有操作的值来让写入完成。

2.5 分布式存储系统Megastore

- 2.5.1 设计目标及方案选择
- 2.5.2 Megastore数据模型
- 2.5.3 Megastore中的事务及并发控制
- 2.5.4 Megastore基本架构
- ▶ 2.5.5 核心技术——复制
- 2.5.6 产品性能及控制措施

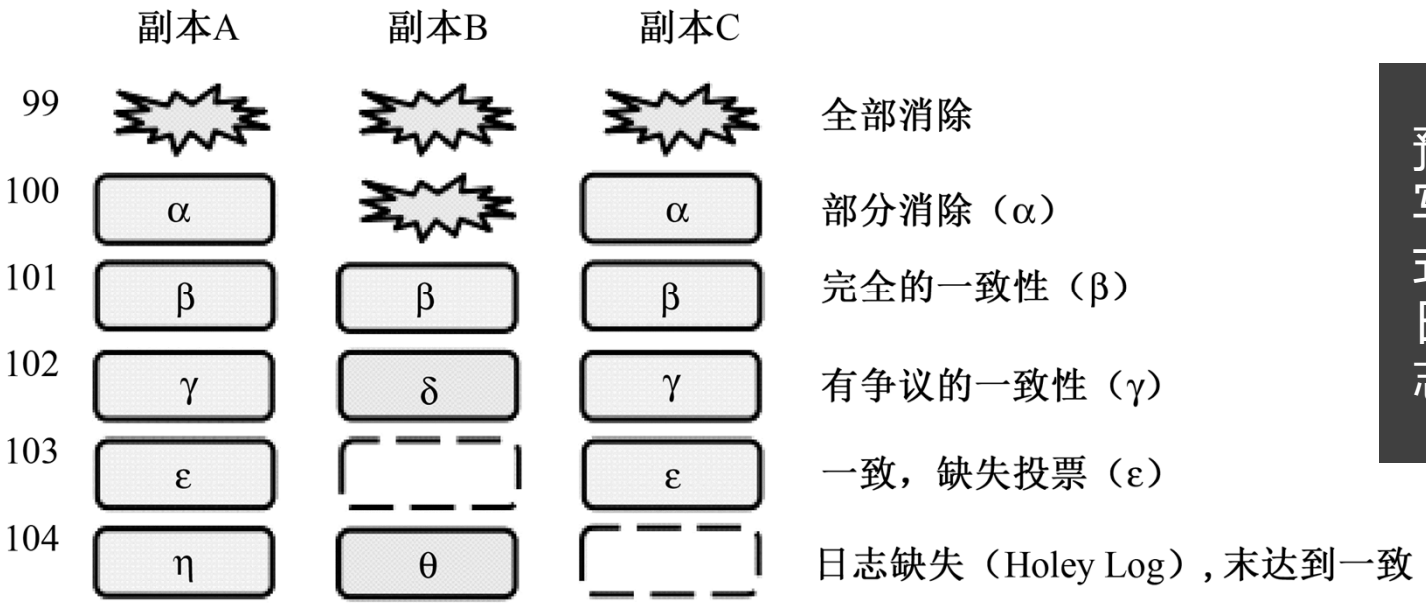
通过复制，保证所有最新的数据都保存有一定数量的副本，能很好地提高系统的可用性

- **复制的日志**

即使这些数据记录正从一个之前的故障中恢复数据，副本也要保证其能够参与到写操作中的Paxos算法。

每个副本都存有记录所有更新的数据。

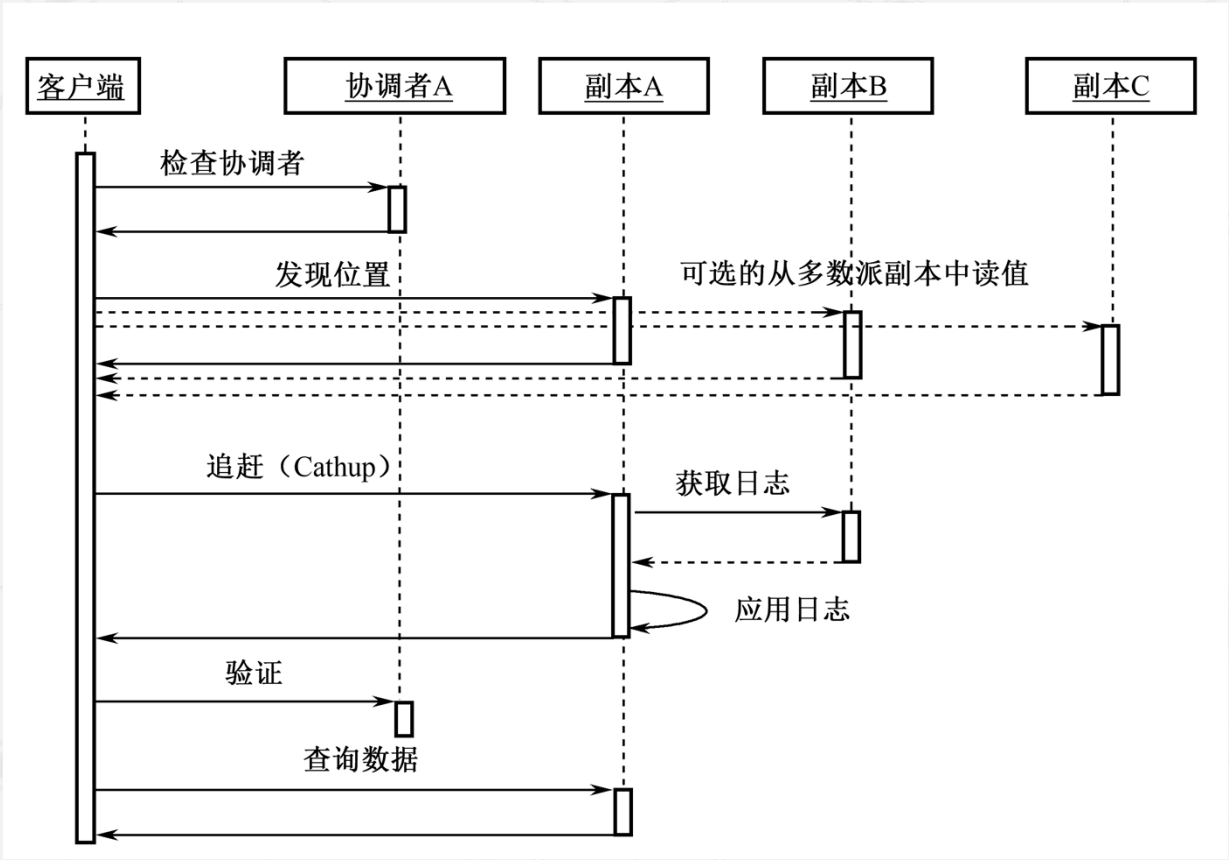
Megastore允许副本不按顺序接受日志，这些日志将独立的存储在Bigtable中。



预写式日志

数据读取

追赶：在一次Current读之前，要保证至少有一个副本上的数据是最新的，所有之前提交到日志中的更新必须复制到该副本上并确保在该副本上生效



本地查询

发现位置

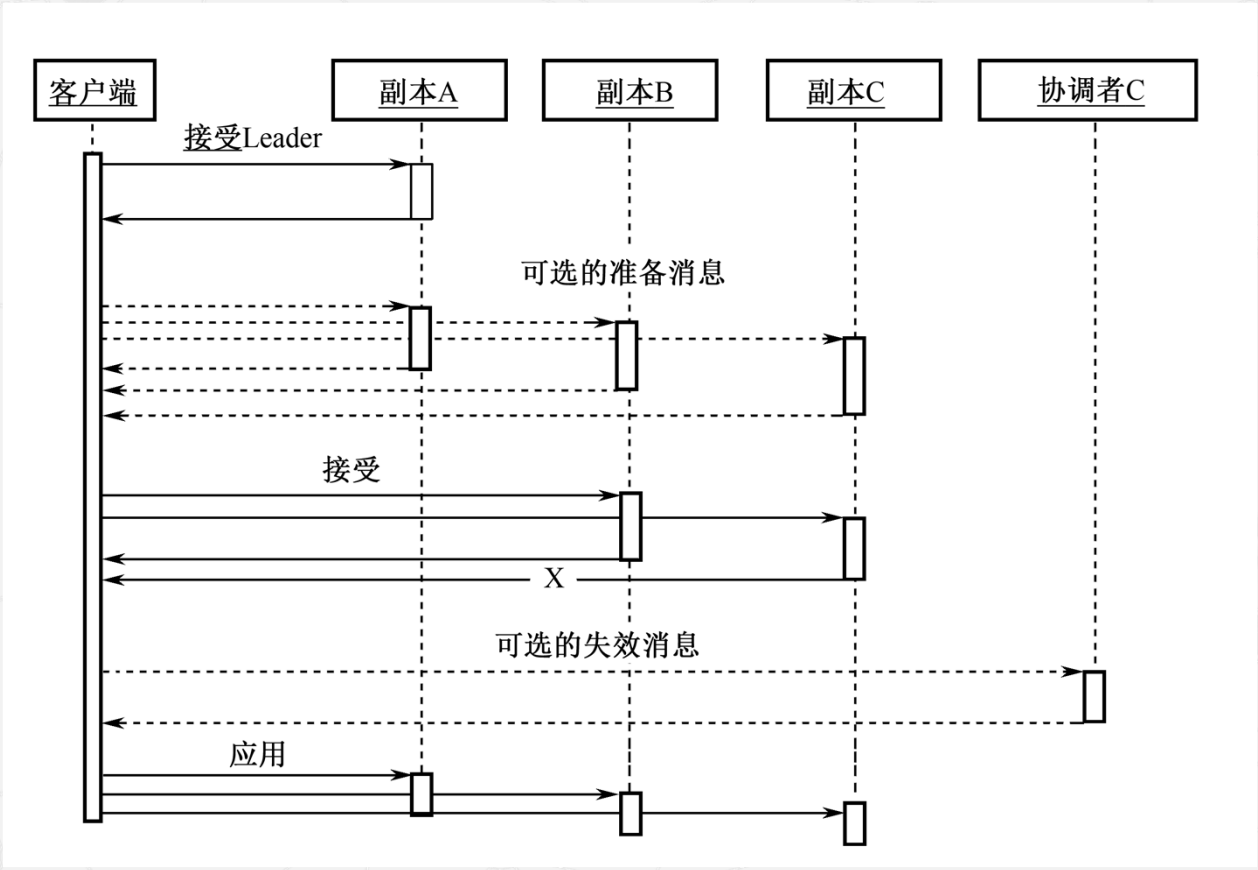
追赶

验证

查询数据

1. 本地查询：查询本地副本的协调者来确认这个实体组上的数据是否是最新的。
2. 发现位置：获取已Commit的最大的Log Position，而后基于该Log Position选择最合适的副本：如果第1步中Local Replica的数据是最新的，则直接读取Local Replica中已接收的最大的Log Position以及Timestamp信息。如果Local Replica中的数据不是最新的，则从大多数副本中获取已被接收的最大的Log Position信息以及Timestamp信息，然后选择一个通常能快速响应的Replica或者是数据最新的Replica。
3. 追赶：进行数据追赶，需要补齐所有的缺失的数据。
4. 验证：如果Local Replica被选中了，而且Local Replica的数据并不是最新的，这个时候需要给Coordinator发送一个验证信息（validate）来确认要读取的Entity Group在该Replica中的数据是完整的。
5. 查询数据：基于第2步获取到的Timestamp信息开始读取数据。如果读取过程中，所选择的Replica变得不可用了，则需要重新更换一个Replica，然后同样需要执行数据追赶操作。

- 数据写入



接受leader

准备

接受

失效

生效

- **协调者的可用性**

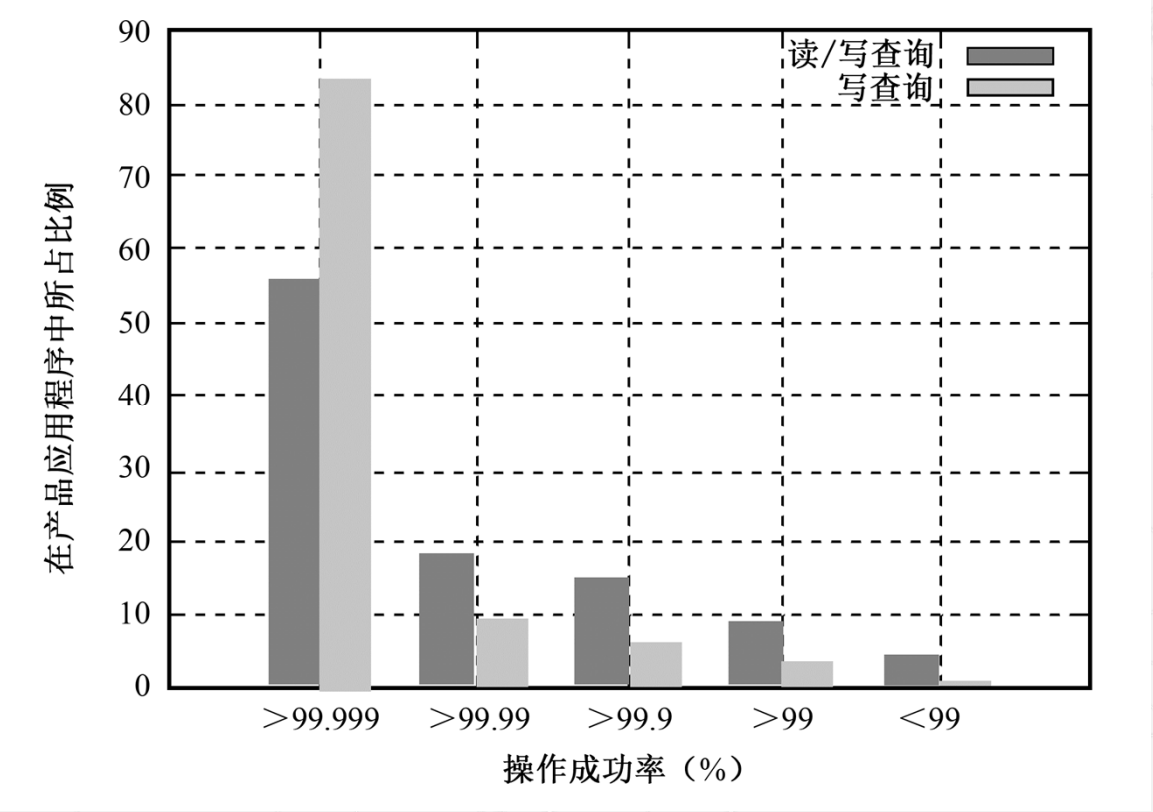
协调者在系统中是比较重要的——协调者的进程运行在每个数据中心。每次的写操作中都要涉及协调者，因此协调者的故障将会导致系统的不可用

Megastore使用了Chubby锁服务，为了处理请求，一个协调者必须持有多数锁。一旦因为出现问题导致它丢失了大部分锁，协调者就会恢复到一个默认保守状态——认为所有它能看到的实体组都是失效的。

除了可用性问题，对于协调者的读写协议必须满足一系列的竞争条件

2.5 分布式存储系统Megastore

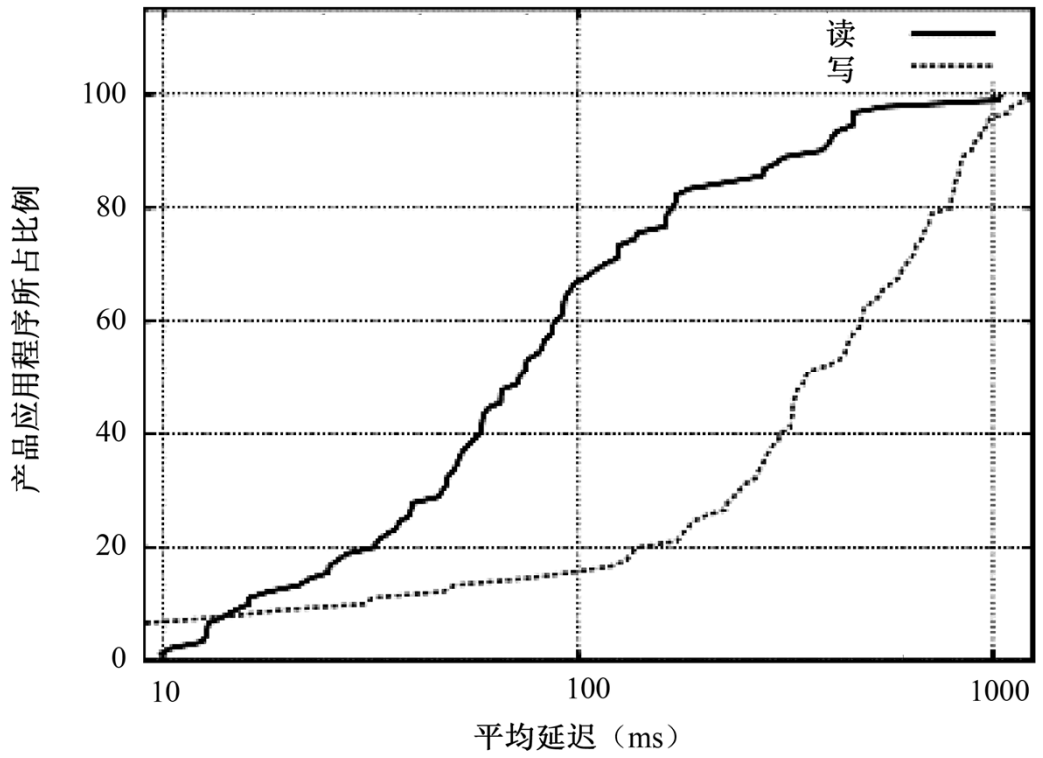
- 可用性的分布情况



Megastore在Google中已经部署和使用了若干年，有超过100个产品使用Megastore作为其存储系统

从图中可以看出，绝大多数产品具有极高的可用性（>99.999%）。这表明Megastore系统的设计是非常成功的，基本达到了预期目标

- 产品延迟情况的分布



应用程序的平均读取延迟在**万分之一毫秒**之内，平均写入延迟在100至400毫秒之间

避免Megastore的性能下降，可采取以下三种应对方法：

（1）重新选择路由使客户端绕开出现问题的副本

（2）将出现问题副本上的协调者禁用，确保问题的影响降至最小。

（3）禁用整个副本


本章未完待续