获取教材和讲义 PPT 等各种课程资料请访问 http://dblab.xmu.edu.cn/node/422

=课程教材由林子雨老师根据网络资料编著=

厦门大学计算机科学系教师 林子雨 编著

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本教程由厦门大学计算机科学系教师林子雨编著，可以作为计算机专业研究生课程《大数据技术基础》的辅助教材。

本教程的主要内容包括：大数据概述、大数据处理模型、大数据关键技术、大数据时代面临的新挑战、NoSQL 数据库、云数据库、Google Spanner、Hadoop、HDFS、HBase、MapReduce、Zookeeper、流计算、图计算和 Google Dremel 等。

本教程是林子雨通过大量阅读、收集、整理各种资料后精心制作的学习材料，与广大数据库爱好者共享。教程中的内容大部分来自网络资料和书籍，一部分是自己撰写。对于自写内容，林子雨老师拥有著作权。

本教程 PDF 文档及其全套教学 PPT 可以通过网络免费下载和使用（下载地址：http://dblab.xmu.edu.cn/node/422）。教程中可能存在一些问题，欢迎读者提出宝贵意见和建议！

本教程已经应用于厦门大学计算机科学系研究生课程《大数据技术基础》，欢迎访问 2013 班级网站 http://dblab.xmu.edu.cn/node/423。

林子雨的 E-mail 是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

林子雨于厦门大学海韵园

2013 年 9 月

第 12 章 Google Spanner

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程网址：http://dblab.xmu.edu.cn/node/422

2013 年 9 月

第 12 章 Google Spanner

Spanner 是一个可扩展、多版本、全球分布式并且支持同步复制的数据库，它是 Google 的第一个可以全球扩展并且支持外部一致性的数据库。Spanner 能做到这些，离不开一个用 GPS 和原子钟实现的时间 API。这个 API 能将数据中心之间的时间同步精确到 10ms 以内。因此，Spanner 有几个给力的功能：无锁读事务、原子模式修改、读历史数据无阻塞。

本章介绍 Google Spanner 相关知识，内容要点如下：

- Spanner 背景
- 与 BigTable、Megastore 的对比
- Spanner 的功能
- 体系结构
- Spanserver
- Directory
- 数据模型
- TrueTime
- Spanner 的并发控制

12.1 Spanner 背景

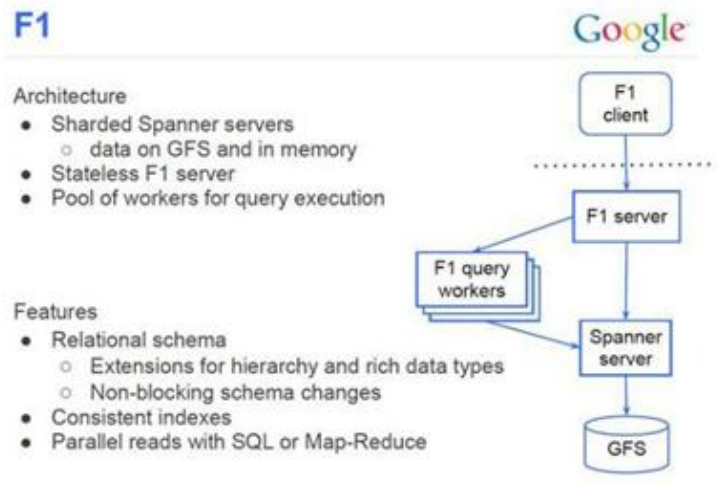

图 12-1 Spanner 在 Google 公司中的定位

要想深刻理解 Spanner 的原理，必须要首先了解 Spanner 在 Google 的定位。从图 12-1 可以看到。Spanner 位于 F1 和 GFS 之间，承上启下。所以，这里先简要介绍 F1 和 GFS。

12.1.1 F1

和众多互联网公司一样，在早期 Google 大量使用了 MySQL。MySQL 是单机的，可以用 Master-Slave 来容错，通过分区来实现扩展。但是，需要大量的手工运维工作，有很多的限制。因此，Google 开发了一个可容错、可扩展的 RDBMS——F1。和一般的分布式数据库不同，F1 对于 RDBMS 应有的功能毫不妥协。起初 F1 是基于 MySQL 的，不过会逐渐迁移到 Spanner。

F1 有如下特点：

- 7×24 高可用，哪怕某一个数据中心停止运转，仍然可用；
- 可以同时提供强一致性和弱一致；
- 可扩展；
- 支持 SQL；
- 事务提交延迟 50-100ms，读延迟 5-10ms，高吞吐。

众所周知，Google BigTable 是重要的 NoSQL 产品，提供很好的扩展性，开源世界有 HBase 与之对应。为什么 Google 还需要 F1，而不是都使用 BigTable 呢？这是因为，BigTable 提供的“最终一致性”，无法满足一些需要强事务一致性的应用的需求。同时，BigTable 还是 NoSQL，而大量的应用场景需要关系模型（现在大量的互联网企业都使用 MySQL 而不愿意使用 HBase）。因此，Google 才设计了可扩展数据库 F1，支持关系模型，而 Spanner 就是 F1 的至关重要的底层存储技术。

12.1.2 Colossus（GFS II）

Colossus 也是一个不得不提起的技术，它是第二代 GFS，对应于开源世界的新 HDFS。

GFS 是著名的分布式文件系统。第一代 GFS 是为批处理而设计的，对于大文件很友好，吞吐量很大，但是延迟较高。所以，使用它的系统不得不对 GFS 做各种优化，才能获得良好的性能。那为什么 Google 没有考虑到这些问题，设计出更完美的 GFS 呢？这是因为，那个时候是 2001 年，Hadoop 出生是在 2007 年。如果 Hadoop 是世界领先水平的话，GFS 比世界领先水平还领先了 6 年。同样地，Spanner 面世时间大概是 2009 年，等到我们看到论文

的时候（注：Google Spanner 英文论文于 2012 年 9 月发布），估计 Spanner 在 Google 已经很完善了，同时可以预见到，Google 内部应该已经有更先进的替代技术在酝酿了。最早在 2015 年才会出现 Spanner 和 F1 的开源产品。

Colossus 是第二代 GFS。Colossus 是 Google 重要的基础设施，因为，它可以满足主流应用对 GFS 的要求。Colossus 的重要改进有：

- 优雅 Master 容错处理 (不再有 2s 的停止服务时间)；
- Chunk 大小只有 1MB (对小文件很友好)；
- Master 可以存储更多的 Metadata(当 Chunk 从 64MB 变为 1MB 后，Metadata 会扩大 64 倍，但是 Google 也解决了)。
- Colossus 可以自动分区 Metadata。使用 Reed-Solomon 算法来复制，可以将原先的 3 份减小到 1.5 份，提高写的性能，降低延迟。

12.2与 BigTable、Megastore 的对比

Spanner 主要致力于跨数据中心的数据复制上，同时也能提供数据库功能。在 Google 类似的系统还有 BigTable 和 Megastore。和这两者相比，Spanner 又有什么优势呢？

BigTable 在 Google 得到了广泛的使用，但是，它不能提供较为复杂的 Schema，也无法提供在跨数据中心环境下的强一致性。Megastore 有类似于 RDBMS 的数据模型，同时也支持同步复制，但是，它的吞吐量太差，不能适应应用要求。Spanner 不再是类似 BigTable 的版本化 key-value 存储，而是一个“临时多版本”的数据库。所谓的“临时多版本”是指，数据是存储在一个版本化的关系表里面，存储的数据会根据其提交的时间打上时间戳，应用可以访问到较老的版本，另外，老的版本也会被垃圾回收掉。

Google 官方认为 Spanner 是下一代 BigTable，也是 Megastore 的继任者。

12.3 Spanner 的功能

从高层看，Spanner 是通过 Paxos 状态机将分区好的数据分布在全球的。数据复制是全球化的，用户可以指定数据复制的份数和存储的地点。 Spanner 可以在集群或者数据发生变化的时候将数据迁移到合适的地点，做负载均衡。用户可以指定将数据分布在多个数据中心，不过更多的数据中心将造成更多的延迟。用户需要在可靠性和延迟之间做权衡，一般来说，复制 1、2 个数据中心足以保证可靠性。

作为一个全球化分布式系统，Spanner 提供一些有趣的特性，主要包括：

（1）应用可以细粒度地指定数据分布的位置，精确地指定数据离用户有多远，可以有效地控制读延迟(读延迟取决于最近的拷贝)；可以指定数据拷贝之间有多远，可以控制写的延迟(写延迟取决于最远的拷贝)；还可以指定数据的复制份数，控制数据的可靠性和读性能(多写几份数据，可以抵御更大的事故)；

（2）Spanner 还有两个一般分布式数据库不具备的特性：读写的外部一致性和基于时间戳的全局的读一致。这两个特性可以让 Spanner 支持一致的备份，一致的 MapReduce，还有原子的 Schema 修改。

这些特性都得益于 Spanner 有一个全球时间同步机制，可以在数据提交的时候给出一个时间戳。因为时间是序列化的，所以才有外部一致性。这个很容易理解，如果有两个提交，一个在时间 T1，一个在时间 T2，那么，更晚的时间戳那个提交是正确的。

这个全球时间同步机制是由一个具有 GPS 和原子钟的 TrueTime API 提供的。这个 TrueTime API 能够将不同数据中心的时间偏差缩短到 10ms 以内。这个 API 可以提供一个精确的时间，同时给出误差范围。Google 已经有了一个 TrueTime API 的实现。这个 TrueTimeAPI 非常有意义，如果能单独开源实现这部分机制的话，很多数据库（如 MongoDB）都可以从中受益。

12.4 体系结构

Spanner 由于是全球化的，所以有两个其它分布式数据库所没有的概念：

- **Universe**：一个 Spanner 部署实例，称为一个 Universe。目前全世界有 3 个，一个开发，一个测试，一个线上。因为一个 Universe 就能覆盖全球，因此，不需要多个。
- **Zone**：每个 Zone 相当于一个数据中心，一个 Zone 内部物理上必须在一起。而一个数据中心可能有多个 Zone。可以在运行时添加或移除 Zone。一个 Zone 可以理解为一个 BigTable 部署实例。

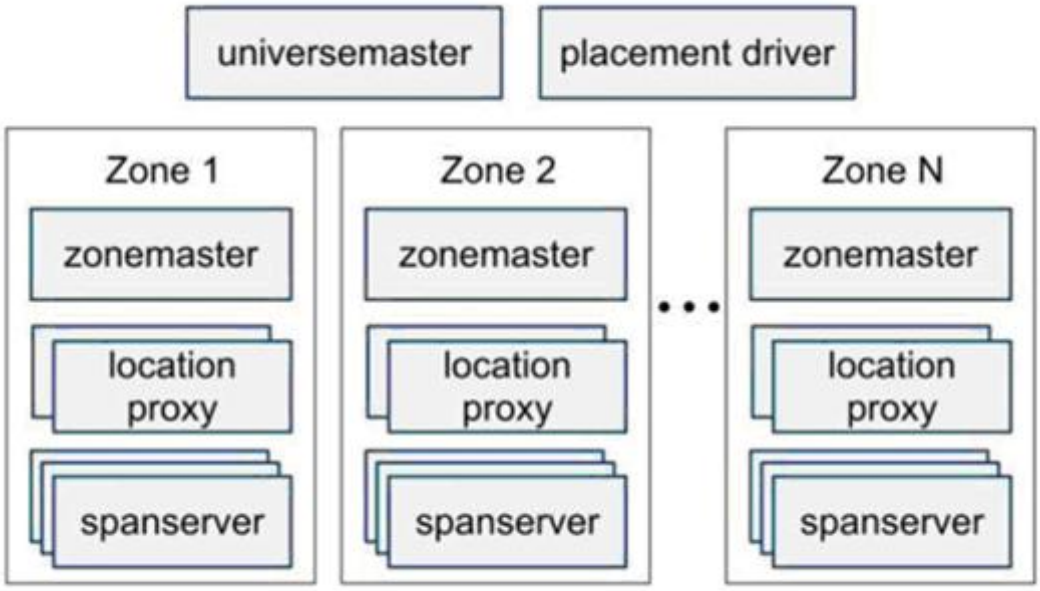

图 12-2 Spanner 体系结构

图 12-2 显示了在一个 Spanner 的 universe 中的服务器架构。一个 zone 包括一个 zonemaster 和一百至几千个 spanserver。Zonemaster 把数据分配给 spanserver，spanserver 把数据提供给客户端。客户端使用每个 zone 上面的 location proxy 来定位可以为自己提供数据的 spanserver。Universe master 和 placement driver，当前都只有一个。Universe master 主要是一个控制台，它显示了关于 zone 的各种状态信息，可以用于相互之间的调试。Placement driver 会周期性地与 spanserver 进行交互，来发现那些需要被转移的数据，或者是为了满足新的副本约束条件，或者是为了进行负载均衡。

简单总结一下，一个 Spanner 主要包括以下组件：

- Universemaster：监控一个 universe 里 zone 级别的状态信息；
- Placement driver：提供跨区数据迁移时的管理功能；
- Zonemaster：相当于 BigTable 的 Master，管理 Spanserver 上的数据；
- Location proxy：存储数据的 Location 信息，客户端要先访问它才能知道数据在哪个 Spanserver 上；
- Spanserver：相当于 BigTable 的 ThunkServer，用于存储数据。

12.5 Spanserver

本节详细介绍 Spanserver 的设计实现。Spanserver 的设计和 BigTable 非常地相似。

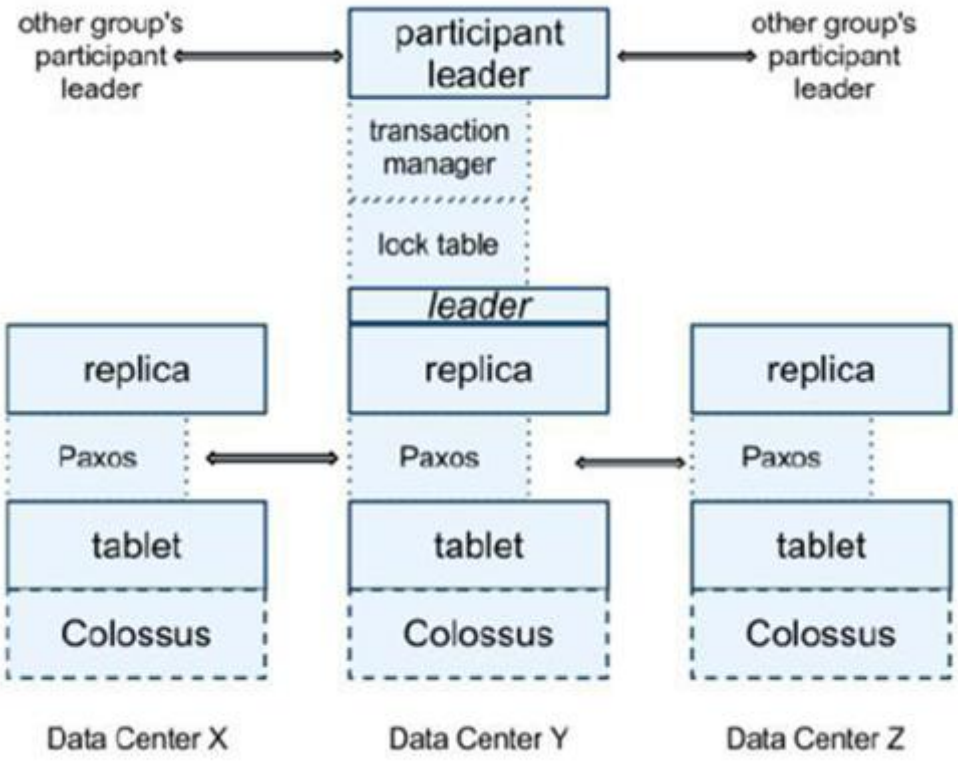

图 12-3 Spanserver 软件栈

如图 12-3 所示，从下往上看，每个数据中心会运行一套 Colossus (GFS II) ，每个机器有 100-1000 个 tablet。Tablet 在概念上将相当于数据库一张表里的一些行，物理上是数据文件。打个比方，一张 1000 行的表，有 10 个 tablet，第 1-100 行是一个 tablet，第 101-200 是一个 tablet。一个 tablet 就类似于 BigTable 中的 tablet，也实现了下面的映射：

(key:string, timestamp:int64)->string。

与 BigTable 不同的是，Spanner 会把时间戳分配给数据，这种非常重要的方式，使得 Spanner 更像一个多版本数据库，而不是一个键值存储。一个 tablet 的状态是存储在类似于 B-树的文件集合和写前(write-ahead)日志中的，所有这些都会被保存到一个分布式的文件系统中，这个分布式文件系统被称为 Colossus，它继承自 Google File System。

在 Spanner 中，每个 Tablet 上会有一个 Paxos 状态机（Paxos 是一个分布式一致性协议），Table 的元数据和日志都存储在上面。Paxos 会选出一个副本（replica）做 leader，这个 leader 的寿命默认是 10s，10s 后重选。Leader 就相当于复制数据的 master，其他副本的数据都是从它那里复制的。读请求可以走任意的部分，但是，写请求只有去找 leader。这些副本统称为一个 paxos group。

每个 leader replica 在 spanserver 上会实现一个锁表来管理并发。锁表记录了两阶段提交需要的锁信息。但是，不论是在 Spanner 还是在 BigTable 上，当遇到冲突的时候，长事务的

性能会表现得很差。所以，有一些操作（如事务读）可以走锁表，其它的操作可以绕开锁表。

每个 leader replica 的 spanserver 上还有一个事务管理器。如果事务在一个 paxos group 里面，可以绕过事务管理器。但是，一旦事务跨多个 paxos group，就需要事务管理器来协调。其中一个事务管理器被选为 leader，其它的事务管理器是 slave，听从 leader 的指挥。这样可以保证事务性。

12.6 Directory

之所以 Spanner 比 BigTable 有更强的扩展性，主要原因在于，Spanner 还有一层抽象的概念——directory。directory 是一些 key-value 的集合，一个 directory 里面的 key 有一样的前缀，更妥当的叫法是 bucketing。 Directory 是应用控制数据位置的最小单元，可以通过谨慎地选择 key 的前缀来控制。

Directory 作为数据放置的最小单元，可以在 paxos group 里面移来移去。Spanner 移动一个 directory 一般出于如下几个原因：

- 一个 paxos group 的负载太大，需要切分；
- 将数据移动到距离访问者更近的地方；
- 将经常同时访问的 directory 放到一个 paxos group 里面。

Directory 可以在不影响 client 的前提下，在后台移动。移动一个 50MB 的 directory 大概需要几秒钟时间。

那么 directory 和 tablet 又是什么关系呢？可以这么理解，Directory 是一个抽象的概念，管理数据的单元；而 tablet 是物理的东西，数据文件。由于一个 Paxos group 可能会有多个 directory，所以，Spanner 的 tablet 实现和 BigTable 的 tablet 实现有些不同。BigTable 的 tablet 是单个顺序文件。而 Spanner 的 tablet 可以理解为是一些基于行的分区的容器。这样就可以将一些经常同时访问的 directory 放在一个 tablet 里面，而不用太在意顺序关系。

在 paxos group 之间移动 directory 是后台任务，这个操作还被用来移动副本。移动操作在设计的时候并没有采用事务来实现，否则，会造成大量的读写阻塞（block）。操作的时候，需要先将实际数据移动到指定位置，然后再用一个原子的操作更新元数据，这样就完成了整个移动过程。

Directory 还是记录地理位置的最小单元。数据的地理位置是由应用决定的，配置的时候需要指定复制数目和类型，还有地理的位置(比如上海复制 2 份；南京复制 1 份) 。

12.7 数据模型

Spanner 的数据模型来自于 Google 内部的实践。在设计之初，Spanner 就决心有以下的特性：

- 支持类似关系数据库的 schema；
- Query 语句；
- 支持广义上的事务。

为何会这样决定呢？在 Google 内部还有一个 Megastore，尽管要忍受性能不够的折磨，但是在 Google 还有 300 多个应用在用它，因为，Megastore 支持一个类似关系数据库的 schema，而且支持同步复制 (BigTable 只支持最终一致的复制) 。使用 Megastore 的应用包括大名鼎鼎的 Gmail、Picasa、Calendar、Android Market 和 AppEngine 等等。 而必须对 Query 语句的支持，则来自于广受欢迎的 Dremel，它可以支持在 2-3 秒内实现对 PB 级别数据的查询。最后，对事务的支持是必不可少的了，BigTable 在 Google 内部被抱怨的最多的就是其只能支持行级事务，再大粒度的事务就无能为力了。Spanner 的开发者认为，过度使用事务造成的性能下降的恶果，应该由应用的开发者来承担；应用开发者在使用事务的时候，必须考虑到性能问题;而数据库必须提供事务机制,而不是因为性能问题就干脆不提供事务支持。

Spanner 的数据模型是建立在 directory 和 key-value 模型的抽象之上的。一个应用可以在一个 universe 中建立一个或多个 database，在每个 database 中建立任意的 table。Table 看起来就像关系型数据库的表，有行，有列，还有版本。Query 语句看起来是多了一些扩展的 SQL 语句。

Spanner 的数据模型也不是纯正的关系模型，每一行都必须有一列或多列组件，看起来还是 Key-value，主键组成 Key，其他的列是 Value。但是，这样的设计对应用而言也是很有裨益的，应用可以通过主键来定位到某一行。

```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

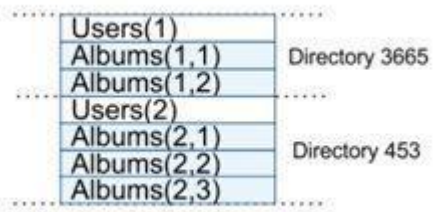

图 12-4 Spanner 模式实例

图 12-4 是一个 Spanner 模式的例子。对于一个典型的相册应用，需要存储其用户和相册，可以用上面的两个 SQL 来创建表。Spanner 的表是层次化的，最顶层的表是 directory table，其它的表在创建的时候，可以用 interleave in parent 来声明层次关系。这样的结构在实现的时候，Spanner 可以将嵌套的数据放在一起，这样在分区的时候性能会提升很多。否则，Spanner 无法获知最重要的表之间的关系。

12.8 TrueTime

Method	Returns
TT.now()	*TTinterval*: [*earliest*, *latest*]
TT.after(*t*)	true if *t* has definitely passed
TT.before(*t*)	true if *t* has definitely not arrived

表 12-1 TrueTime API

TrueTime API（如表 12-1 所示）是一个非常有创意的东西，可以同步全球的时间。TT.now()可以获得一个绝对时间 TTinterval，这个值和 UnixTime 是相同的，同时，还能够得到一个误差 e。TT.after(t)和 TT.before(t)是基于 TT.now()实现的。

在底层，TrueTime 使用的时间是 GPS 和原子钟。TrueTime 使用两种类型的时间，是因为它们有不同的失败模式。GPS 参考时间的弱点是天线和接收器失效、局部电磁干扰和相关失败（比如设计上的缺陷导致无法正确处理闰秒和电子欺骗），以及 GPS 系统运行中断。原子钟也会失效，不过失效的方式和 GPS 无关，不同原子钟之间的失效也不存在彼此相关

性。由于存在频率误差，在经过很长的时间以后，原子钟也会产生明显误差。

TrueTime 是由每个数据中心上面的许多 time master 机器和每台机器上的一个 timeslave daemon 来共同实现的。大多数 master 都有具备专用天线的 GPS 接收器，这些 master 在物理上是相互隔离的，这样可以减少天线失效、电磁干扰和电子欺骗的影响。剩余的 master（我们称为 Armageddon master）则配备了原子钟。一个原子钟并不是很昂贵：一个 Armageddon master 的花费和一个 GPS master 的花费是同一个数量级的。所有 master 的时间参考值都会进行彼此校对。每个 master 也会交叉检查时间参考值和本地时间的比值，如果二者差别太大，就会把自己驱逐出去。在同步期间，Armageddon master 会表现出一个逐渐增加的时间不确定性，这是由保守应用的最差时钟漂移引起的。GPS master 表现出的时间不确定性几乎接近于 0。

每个 daemon 会从许多 master 中收集投票，获得时间参考值，从而减少误差。被选中的 master 中，有些 master 是 GPS master，是从附近的数据中心获得的，剩余的 GPS master 是从远处的数据中心获得的；还有一些是 Armageddon master。Daemon 会使用一个 Marzullo 算法的变种，来探测和拒绝欺骗，并且把本地时钟同步到非撒谎 master 的时间参考值。为了免受较差的本地时钟的影响，Spanner 会根据组件规范和运行环境确定一个界限，如果机器的本地时钟误差频繁超出这个界限，这个机器就会被驱逐出去。

在同步期间，一个 daemon 会表现出逐渐增加的时间不确定性。ε 是从保守应用的最差时钟漂移中得到的。ε 也取决于 time master 的不确定性，以及与 time master 之间的通讯延迟。在 Google 的线上应用环境中，ε 通常是一个关于时间的锯齿形函数。在每个投票间隔中，ε 会在 1 到 7ms 之间变化。因此，在大多数情况下，ε 的平均值是 4ms。Daemon 的投票间隔，在当前是 30 秒，当前使用的时钟漂移比率是 200 微秒/秒，二者一起意味着 0 到 6ms 的锯齿形边界。剩余的 1ms 主要来自到 time master 的通讯延迟。在失败的时候，超过这个锯齿形边界也是有可能的。例如，偶尔的 time master 不确定性，可能会引起整个数据中心范围内的ε 值的增加。类似地，过载的机器或者网络连接，都会导致ε 值偶尔地局部增大。

12.9 Spanner 的并发控制

Spanner 使用 TrueTime 来控制并发，实现外部一致性，主要支持以下几种事务：

- 读写事务；
- 只读事务；

- 快照读，客户端提供时间戳；
- 快照读，客户端提供时间范围。

例如，一个读写事务发生在时间 t，那么，在全世界任何一个地方，指定 t 快照读都可以读到写入的值。

表 12-2 Spanner 支持的事务类型

Operation	Concurrency Control	Replica Required
Read-Write Transaction	pessimistic	leader
Read-Only Transaction	lock-free	leader for timestamp; any for read
Snapshot Read, client-provided timestamp	lock-free	any
Snapshot Read, client-provided bound	lock-free	any

表 12-2 是 Spanner 现在支持的事务。单独的写操作都被实现为读写事务；单独的非快照被实现为只读事务。事务总有失败的时候，如果失败，对于这两种操作会自己重试，无需应用自己实现重试循环。

时间戳的设计大大提高了只读事务的性能。事务开始的时候，要声明这个事务里没有写操作。只读事务可不是一个简单的没有写操作的读写事务，它会用一个系统时间戳去读，所以，对于同时的其他的写操作是没有阻塞的。而且，只读事务可以在任意一台已经更新过的副本上面读。

对于快照读操作，可以读取以前的数据，需要客户端指定一个时间戳或者一个时间范围。Spanner 会找到一个已经充分更新好的副本上读取。

还有一个有趣的特性的是，对于只读事务，如果执行到一半，该副本出现了错误，客户端没有必要在本地缓存刚刚读过的数据，因为，数据是根据时间戳读取的，只要再用刚刚的时间戳去读取，就可以获得一样的结果。

- **读写事务**

正如 BigTable 一样，Spanner 的事务是会将所有的写操作先缓存起来，在 Commit 的时候一次提交。这样的话，就读不出在同一个事务中写的数据了。不过这没有关系，因为 Spanner 的数据都是有版本的。

Spanner 在读写事务中使用 wound-wait 算法来避免死锁。当客户端发起一个读写事务的时候，首先是读操作，它先找到相关数据的 leader replica，然后加上读锁，读取最近的数据。在客户端事务存活的时候会不断地向 leader 发心跳，防止超时。当客户端完成了所有的读操作，并且缓存了所有的写操作，就开始了两阶段提交。客户端选择一个 coordinator，并给每一个 leader 发送 coordinator 的 id 和缓存的写数据。

leader 首先会上一个写锁，它要找一个比现有事务晚的时间戳。通过 Paxos 记录，每一个相关的 leader 都要给 coordinator 发送它自己准备的那个时间戳。

Coordinator 一开始也会上个写锁，当大家发送时间戳给它之后，它就选择一个提交时间戳。这个提交的时间戳，必须比刚刚的所有时间戳晚，而且还要比 TT.now()+误差时间还要晚。这个 Coordinator 将这个信息记录到 Paxos。

在让副本写入数据生效之前，coordinator 还要再等一会儿，需要等两倍时间误差。这段时间也刚好让 Paxos 来同步。因为等待之后，在任意机器上发起的下一个事务的开始时间，都不会比这个事务的结束时间早了。然后，coordinator 将提交时间戳发送给客户端还有其它的副本，它们记录日志，写入生效，释放锁。

- **只读事务**

对于只读事务，Spanner 首先要指定一个读事务时间戳，还需要了解在这个读操作中，需要访问的所有的读的 Key。Spanner 可以自动确定 Key 的范围。

如果 Key 的范围在一个 Paxos group 内。客户端可以发起一个只读请求给 group leader。leader 选一个时间戳，这个时间戳要比上一个事务的结束时间要大，然后读取相应的数据。这个事务可以满足外部一致性，读出的结果是最后一次写的结果，并且不会有不一致的数据。

如果 Key 的范围在多个 Paxos group 内，就相对复杂一些。其中一个比较复杂的例子是，可以遍历所有的 group leaders，寻找最近的事务发生的时间并读取。客户端只要时间戳在 TT.now().latest 之后就可以满足要求了。

本章小结

本章首先介绍了 Spanner 的诞生背景，并把 Spanner 与 BigTable、Megastore 进行了对比；接下来简单介绍了 Spanner 的功能和体系结构；然后，介绍了 Spanner 的具体设计方法，包括 Spanserver、Directory、数据模型、TrueTime 等，最后，介绍了 Spanner 的并发控制机制。

参考文献

[1] 林子雨翻译. Google Spanner(中文版). http://dblab.xmu.edu.cn/node/230

[2] 颜开. 全球级的分布式数据库 Google Spanner 原理. http://www.oschina.net/question/12_70811

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.

主讲课程：《大数据技术基础》

办公地点：厦门大学海韵园科研 2 号楼

E-mail: ziyulin@xmu.edu.cn

个人网页：http://www.cs.xmu.edu.cn/linziyu