Google—BigTable 读后感

18271074 施泽宇

Google 在 2006 年的 OSDI 发表了了两篇论文，分别是 BigTable 和 Chubby。BigTable 是用于管理结构化数据的分布式存储系统，被 Google 构建在 GFS、Chubby、SSTable 等服务上；同时，Google Earth，Google Analytics 等应用采用了这一技术。本篇读后感只对这一篇论文进行仔细的讨论。

1.BigTable 简要介绍

Bigtable 是一个分布式的结构化数据存储系统，它被设计用来处理海量分布在数千台普通服务器上的 PB 级的数据。Bigtable 和数据库有着类似的地方，并且使用了很多数据库的实现策略。和普通的并行、内存数据库不同，Bigtable 提供了完全不一样的接口。

Bigtable 不支持完整的关系数据模型；它为客户提供了简单的数据模型，利用这个模型，客户可以动态控制数据的分布和格式，用户也可以自己推测底层存储数据的位置相关性。比如树状结构的存储当中，具有相同前缀的数据的存放位置接近。因此在读取的时候，可以把这些数据一次读取出来。

Bigtable 将存储的数据都视为字符串，但是 Bigtable 本身不去解析这些字符串，客户程序通常会在把各种结构化或者半结构化的数据串行化到这些字符串里。通过仔细选择数据的模式，客户可以控制数据的位置相关性。

用户可以通过 BigTable 的模式参数来控制数据是存放在内存中、还是硬盘上。

2.BigTable 的数据模型

Bigtable 是一个稀疏的、分布式的、持久化存储的多维度排序 Map。Map 的索引是行关键字、列关键字以及时间戳；Map 中的每个 value 都是一个未经解析的 byte 数组。

$$(row:string,\ column:string,time:int64)->string$$

本质上说，Bigtable 是一个键值的映射，它所运用的很多术语例如 table，row 等和关系型数据库没有任何的关系。实际上，bigtable 是一个稀疏的、分布式的、持久化的、多为的排序映射。

2.1 行

行键通常有 10-100 字节。行的读写都是原子性的。Bigtable 按照行键的字典序存储数据。Bigtable 的表会根据行键自动划分为片，Bigtable 通过行关键字的字典顺序来组织数据。表中的每个行都可以动态分区。**最初表都只有一个片，但随着表不断增大，片会自动分裂。（片，即 tablet，在下文有详细的讨论）**

2.2 列

列族是访问控制的基本单位。存放在同一列族下的所有数据通常都属于同一个类型（我们可以把同一个列族下的数据压缩在一起）。列族在使用之前必须先创建，然后才能在列族中任何的列关键字下存放数据；列族创建后，

其中的任何一个列关键字下都可以存放数据。根据我们的设计意图，一张表中的列族不能太多（最多几百个），并且列族在运行期间很少改变。与之相对应的，一张表可以有无限多个列。

访问控制、磁盘和内存的使用统计都是在列族层面进行的。在我们的 Webtable 的例子中，上述的控制权限能帮助我们管理不同类型的应用：我们允许一些应用可以添加新的基本数据、一些应用可以读取基本数据并创建继承的列族、一些应用则只允许浏览数据（甚至可能因为隐私的原因不能浏览所有数据）。

2.3 时间戳

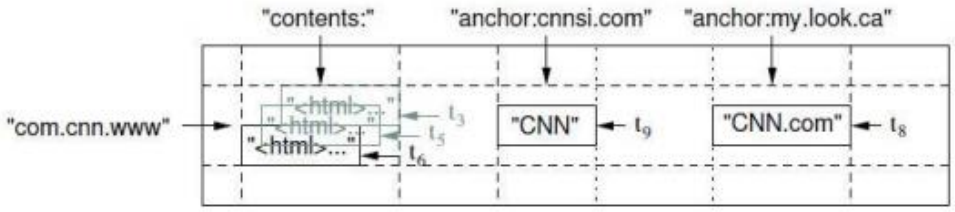
为了减轻多个版本数据的管理负担，每一个列族配有两个设置参数，Bigtable 通过这两个参数可以对废弃版本的数据自动进行垃圾收集。用户可以指定只保存最后 n 个版本的数据，或者只保存“足够新”的版本的数据。

数据项中，不同版本的数据按照时间戳倒序排序，即最新的数据排在最前面。

```
table{
  // ...
  "aaaaa" : { //一行
    "A:foo" : { //一列
        15 : "y", //一个版本
        4 : "m"
      },
    "A:bar" : { //一列
        15 : "d",
      },
    "B:" : { //一列
        6 : "w"
        3 : "o"
        1 : "w"
      }
  },
  // ...
}
```

以此图为例，行是第一级别索引，时间戳是第三级别索引。查询时，如果只给出行列，那么返回的是最新版本的数据；如果给出了行列时间戳，那么返回的是时间小于或等于时间戳的数据。

论文当中给出了 webtable 的一个例子：



在 webtable 当中，行用来存储网页，用它的反转的 url 作为行键。以列族“archor”为例，保存了网页的引用站点，“qualifier”是站点的名称，数据是链接的文本。而"contents:"列下保存了网页的三个版本，比如我们可以用("com.cnn.www", "contents:", t5)来找到 CNN 主页在 t5 时刻的内容。

3.API 接口

Bigtable 提供了建立和删除表以及列族的 API 函数。Bigtable 还提供了修改集群、表和列族的元数据的 API，比如修改访问权限。论文当中给出了写入的操作：

```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");
// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

客户程序可以进行写入或者删除 bigtable 中的值，从每个行中查找值，或者遍历表中的一个数据子集等操作。

同时，通过 API 还可以进行更加复杂的操作：

Bigtable 支持单行上的事务处理，利用这个功能，用户可以对存储在一个行关键字下的数据进行原子性的读-更新-写操作。

Bigtable 允许把数据项用做整数计数器。

Bigtable 允许用户在服务器的地址空间内执行脚本程序。

4.BigTable 构件

BigTable 是建立在其他的几个 Google 基础构件之上的。

4.1GFS

BigTable 使用 Google 的分布式文件系统(GFS)存储日志文件和数据文件。

BigTable 的进程经常要和其它应用的进程共享机器。BigTable 依赖集理

管理系统来调度任务、管理共享的机器上的资源、处理机器的故障、以及监视机器的状态。

4.2SSTable

BigTable 内部存储数据的文件是 Google SSTable 格式的。SSTable 是一个持久化的、排序的、不可更改的 Map 结构，而 Map 是一个 key-value 映射的数据结构，key 和 value 的值都是任意的 Byte 串。可以对 SSTable 进行如下的操作：查询与一个 key 值相关的 value，或者遍历某个 key 值范围内的所有的 key-value 对。

SSTable 使用块索引（通常存储在 SSTable 的最后）来定位数据块；在打开 SSTable 的时候，索引被加载到内存。每次查找都可以通过一次磁盘搜索完成：首先使用二分查找法在内存中的索引里找到数据块的位置，然后再从硬盘读取相应的数据块。也可以选择把整个 SSTable 都放在内存中，这样就不必访问硬盘了。

4.3Chubby

一个 Chubby 服务包括了 5 个活动的副本，其中的一个副本被选为 Master，并且处理请求。只有在大多数副本都是正常运行的，并且彼此之间能够互相通信的情况下，Chubby 服务才是可用的。

Bigtable 使用 Chubby 完成以下的几个任务：

1. 确保在任何给定的时间内最多只有一个活动的 Master 副本；
2. 存储 BigTable 数据的自引导指令的位置（参考 5.1 节）；
3. 查找 Tablet 服务器，以及在 Tablet 服务器失效时进行善后（5.2 节）；
4. 存储 BigTable 的模式信息（每张表的列族信息）；
5. 以及存储访问控制列表。

5.BigTable 的组件和讨论

Bigtable 包括了三个主要的组件：

1.链接到客户程序中的库，客户端需要读写数据时，直接与片服务器联系。因为客户端并不需要从主服务器获取片的位置信息，所以大多数客户端从来不需要访问主服务器，主服务器的负载一般很轻。

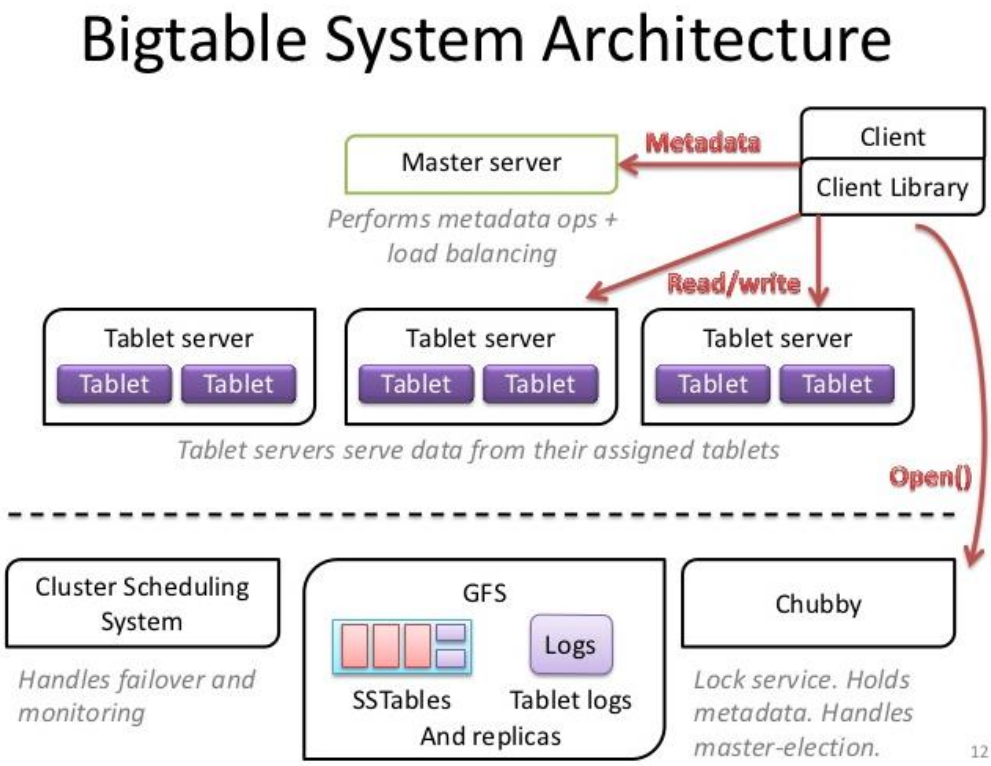
2.一个 Master 服务器，主服务器负责将片分配给片服务器，监控片服务器的添加和删除，平衡片服务器的负载，处理表和列族的创建等。注意，主服务器不存储任何片，不提供任何数据服务，也不提供片的定位信息。

3.多个 Tablet 服务器，每个片服务器负责一定量的片，处理对片的读写请求，以及片的分裂或合并。每个片实际由若干 SSTable 文件和 memtable 组成，而且这些 SSTable 和 memtable 都是已排序的。片服务器可以根据负载随时添加和删除。这里片服务器并不真实存储数据，而相当于一个连接 Bigtable 和 GFS 的代理，客户端的一些数据操作都通过片服务器代理间接访问 GFS。

针对系统工作负载的变化情况，BigTable 可以动态的向集群中添加（或者删除）Tablet 服务器。

Master 服务器负责为 Tablet 服务器分配 Tablets、检测新加入的或者过期失效的 Table 服务器、对 Tablet 服务器进行负载均衡、以及对保存在 GFS

上的文件进行垃圾收集等工作。



客户端读取的数据都不经过 Master 服务器：客户程序直接和 Tablet 服务器通信进行读写操作。由于 BigTable 的客户程序不必通过 Master 服务器来获取 Tablet 的位置信息，因此，大多数客户程序甚至完全不需要和 Master 服务器通信。在实际应用中，Master 服务器的负载是很轻的。

当片服务器启动时，它会在 Chubby 的某个特定目录下创建并获取一个锁文件（互斥锁），这个锁文件的名称是唯一表示该 tablet server 的。master server 通过监控这个目录获取当前存活着的 tablet server 的信息。

1.如果 tablet server 失去了锁，那么 tablet server 也就不再为对应的 tablet 服务了。

2.如果锁文件存在，那么 tablet server 会主动获取锁。

3.如果锁文件不存在，那么 tablet server 就永远不会再服务对应的 tablet 了，所以 tablet server 就会 kill 自己。

4.当 tablet server 要终止时，它会自己释放占有的锁，master server 就会把该 tablet server 上的 tablet 分配给其它的 tablet server。

master server 会定期轮询每个 tablet server 的锁状态。如果 tablet server 报告自己失去了已经失去了锁，或者 master server 不能获取 tablet server 的状态，那么 master server 就会尝试去获取 tablet server 对应的锁文件。如果 master server 获取到了锁文件，并且 Chubby 是处于正常工作的状态的，此时 master server 就确认 tablet server 已经无法再提供服务了，master server 删除相应的锁文件并把 tablet server 对应的 tablet 分配给新的 tablet server。

如果 master server 与 Chubby 之间出现了网络问题，那么 master server 也会 kill 自己。但是这并不会影响 tablet 与 tablet server 之间的分配关系。

master server 的启动需要经历一下几个阶段：

1.master server 需要从 Chubby 获取锁，这样可以确保在同一时刻只有一个 master server 在工作。

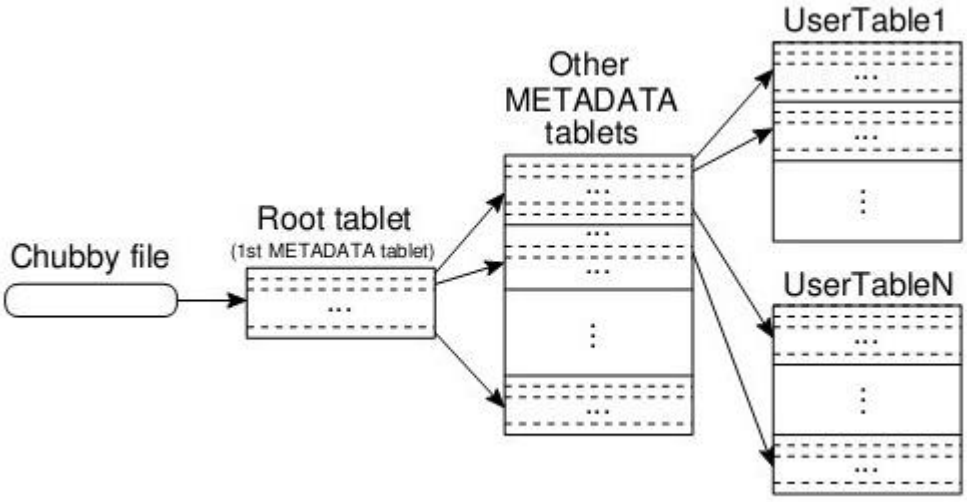
2.master server 扫描 Chubby 下特定的目录（即 tablet server 创建锁文件的目录），获取存活着的 tablet server 的信息。

3.master server 与存活着的 tablet server 通信，获取已被分配到 tablet server 的 tablet 信息。

4.master server 扫描 METADATA tablet，获取所有的 tablet 信息，然后把未分配的 tablet 分配给 tablet server。

5.1 tablet 的定位

Bigtable 使用一个类似 B+树的数据结构存储片的位置信息。

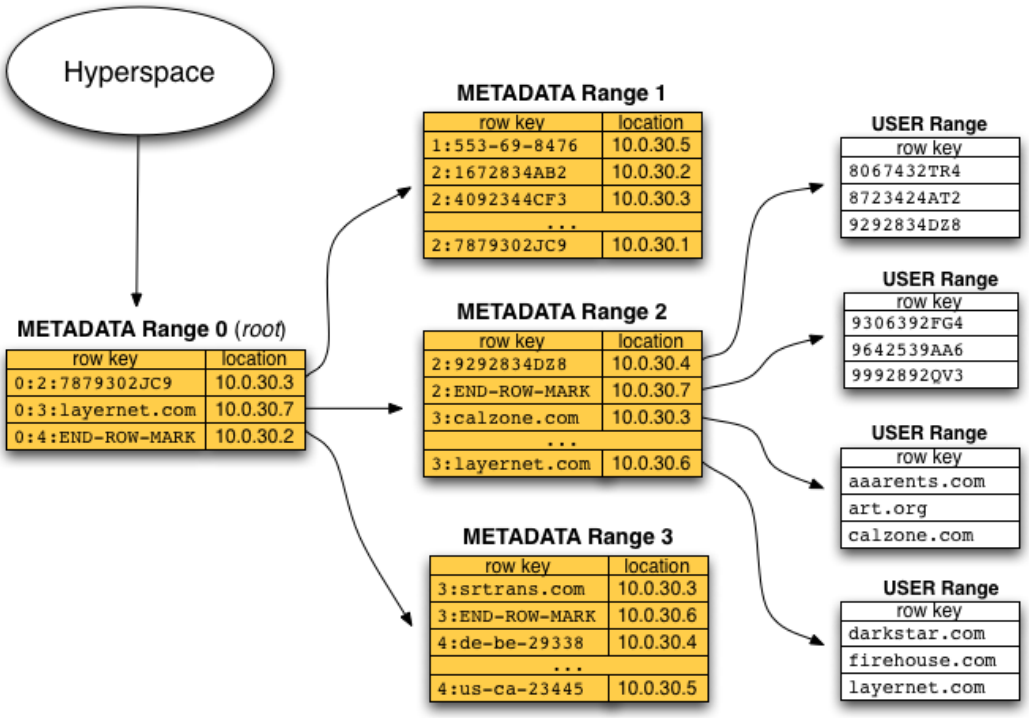


第一层是一个存储在 Chubby 中的文件，它包含了 Root Tablet 的位置信息。Root Tablet 包含了一个特殊的 METADATA 表里所有的 Tablet 的位置信息。METADATA 表的每个 Tablet 包含了一个用户 Tablet 的集合。

Root Tablet 实际上是 METADATA 表的第一个 Tablet，只不过对它的处理比较特殊：**Root Tablet 永远不会被分割**。这就保证了 Tablet 的位置信息存储结构不会超过三层。

5.2 元数据表

每个片都可能由多个 SSTable 文件组成，列族可以用来存储任意多个 SSTable 文件的位置。一个合理的假设就是每个 SSTable 文件的位置信息占据一列，列名为 location:filename。当然不一定非得用列键存储完整文件名，更大的可能性是把 SSTable 文件名存在值里。获取了文件名就可以向 GFS 索要数据了。

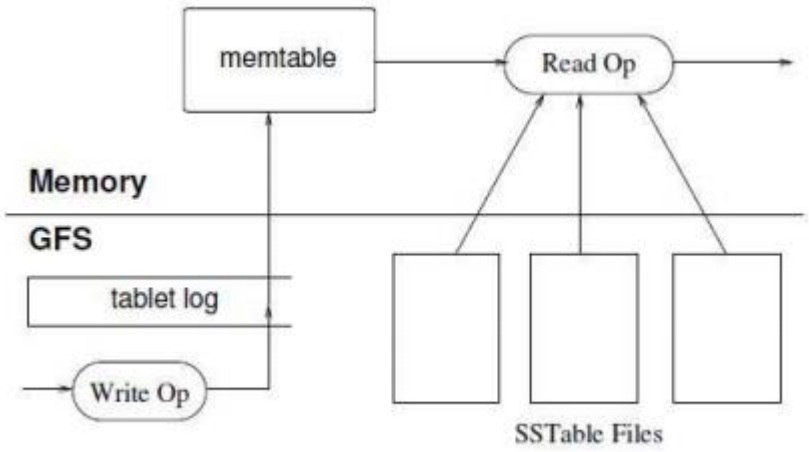


客户端会缓存 tablet 的位置信息，客户端在获取 tablet 的位置信息时，会涉及到两种情况：

1. 如果客户端没有缓存目标 tablet 的位置信息，那么就会沿着 root tablet 定位到最终的 tablet，整个过程需要 3 次网络往返。

2. 如果客户端缓存了目标 tablet 的位置信息，但是到了目标 tablet 后发现原来缓存的 tablet 位置信息过时了，那么会重新从 root tablet 开始定位 tablet，整个过程需要 6 个 network round-trips。

5.3tablet 的存储和读写



当片服务器收到一个写请求，**片服务器首先检查请求是否合法**。如果合法，先将写请求提交到日志去，然后将数据写入内存中的 memtable。memtable

相当于 SSTable 的缓存，当 memtable 成长到一定规模会被冻结，Bigtable 随之创建一个新的 memtable，并且将冻结的 memtable 转换为 SSTable 格式写入 GFS，这个操作称为 minor compaction。

当片服务器收到一个读请求，同样要检查请求是否合法。如果合法，这个读操作会查看所有 SSTable 文件和 memtable 的合并视图，因为 SSTable 和 memtable 本身都是已排序的，所以合并相当快。

每一次 minor compaction 都会产生一个新的 SSTable 文件，SSTable 文件太多读操作的效率就降低了，所以 Bigtable 定期执行 merging compaction 操作，将几个 SSTable 和 memtable 合并为一个新的 SSTable。BigTable 还有一个操作 major compaction，它将所有 SSTable 合并为一个新的 SSTable。

6.BigTable 的优化

6.1 局部性群组

Bigtable 定期执行 merging compaction 操作，将几个 SSTable 和 memtable 合并为一个新的 SSTable。

以以局部性群组为单位设定一些有用的调试参数。这个特性对于需要频繁访问的小块数据特别有用：在 Bigtable 内部，我们利用这个特性提高 METADATA 表中具有位置相关性的列族的访问速度。

6.2 压缩

每个 SSTable 的块（块的大小由局部性群组的优化参数指定）都使用用户指定的压缩格式来压缩。

第一遍采用 Bentley and McIlroy’s 方式，这种方式在一个很大的扫描窗口里对常见的长字符串进行压缩；

第二遍是采用快速压缩算法，即在一个 16KB 的小扫描窗口中寻找重复数据。

需要注意到压缩算法的时候重点考虑的是速度而不是压缩的空间

6.3 通过缓存提高读操作的性能

6.3.1 Bloom 过滤器

我们通过允许客户程序对特定局部性群组的 SSTable 指定 Bloom 过滤器来减少硬盘访问的次数。我们可以使用 Bloom 过滤器查询一个 SSTable 是否包含了特定行和列的数据。对于某些特定应用程序，我们只付出了少量的、用于存储 Bloom 过滤器的内存的代价，就换来了读操作显著减少的磁盘访问的次数。使用 Bloom 过滤器也隐式的达到了当应用程序访问不存在的行或列时，大多数时候我们都不需要访问硬盘的目的。

6.3.2 Commit 日志的实现

为了避免多次读取日志文件，我们首先把日志按照关键字（table，row name，log sequence number）排序。排序之后，对同一个 Tablet 的修改操作的日志记录就连续存放在了一起，因此，我们只要一次磁盘 Seek 操作、之后顺序读取就可以了。为了并行排序，我们先将日志分割成 64MB 的段，之后在不同的 Tablet 服务器对段进行并行排序。这个排序工作由 Master 服务器来协同处理，并且在一个 Tablet 服务器表明自己需要从 Commit 日志文件恢复 Tablet 时开始执行。

6.3.3 Tablet 恢复提速

当 Master 服务器将一个 Tablet 从一个 Tablet 服务器移到另外一个 Tablet 服务器时，源 Tablet 服务器会对这个 Tablet 做一次 Minor Compaction。这个 Compaction 操作减少了 Tablet 服务器的日志文件中没有归并的记录，从而减少了恢复的时间。

6.3.4 利用不变性

我们在使用 Bigtable 时，除了 SSTable 缓存之外的其它部分产生的 SSTable 都是不变的，我们可以利用这一点对系统进行简化。

SSTable 是不变的，因此，我们可以把永久删除被标记为“删除”的数据的问题，转换成对废弃的 SSTable 进行垃圾收集的问题了。