**Bigtable阅读报告**

1. **摘要。**

Bigtable是一种用于管理结构化数据的分布式储存系统，其设计目标是扩展到非常大的规模：跨越数千个商品服务的pb字节的数据。谷歌的许多项目都将数据储存在Bigtable中，包括web indexing、谷歌Earth和谷歌Finance。这些应用程序对Bigtable提出了非常不同的要求，包括数据大小（从url到web页面到卫星图像）和延迟要求（从后端批量处理到实时数据服务）。尽管有这些不同的需求，Bigtable已经成功地为所有这些谷歌产品提供了一个灵活的、高性能的解决方案。

1. **介绍。**

Bigtable兼具了广泛的可使用性、可拓展性、高性能和高可用性。Bigtable在实现策略上与传统数据库有很多相似的地方，但是Bigtable不支持完整的关系型数据模型；相反，Bigtable提供的是一种非常简单的数据模型，以及为客户端提供接口，可以灵活地操控数据层和格式，可以控制数据在物理层的存储方式。对于Bigtable来说，存储的数据都是未解释的字符串，bigtable对数据的行键和列名建立索引。最后，通过调整Bigtable的表结构参数，客户端还可以控制数据是存在内存还是硬盘上。

1. **数据模型。**

Bigtable是一个稀疏的、分布式的、持久化的多维有序Map。这张Map针对行键、列名和时间戳建立了索引；Map中的每个值都是一串未解释的字节数组。具体模型如下：

图示

描述已自动生成

**1、行**

对Bigtable同一行数据的读写操作都是原子的，在客户端需要并发操作同一行数据的时候，这个设计使服务器对读写请求的处理顺序更清晰明了。数据在Bigtable中按照行键的字母序进行排列，Bigtable对数据自动分片，每一份数据的行键范围是连续的。分片后，每一份数据称为一个tablet，tablet是数据分布和负载均衡的基本单位。

通过合理设计行键，用户可以将需要同时分析的数据存储在相邻位置，提高程序运行效率。

**2、列族**

Bigtable的列是分组的，每个组称为一个列族，是访问控制的基本单位。同一列族数据通常是同一类型的，列族的数量需要控制得小一些，最多不能超过几百。一个列key由两部分组成：family:qualifier。列族名必须是可打印，qualifier可以是任意字符串。访问控制以及硬盘内存使用，都是在列族界别上控制的。

**3、时间戳**

Bigtable中每个单元格都可以保存多个版本的数据，版本与版本之间用写入时的时间戳区分，时间戳的数据类型是int64。同一单元格的不同版本，按照时间戳降序排列，因此最近的数据总是最先被读到。Bigtable支持两种方式的数据自动删除（设置是在列族单位上）：  
 客户端可以指定Bigtable只保留最新的n个版本。

 客户端可以指定只保留多长时间以内新写入的数据。

1. **API。**

Bigtable提供的API支持一下几种操作：

1. 建删表、更改表结构等操作。
2. 数据写入删除等操作。
3. 支持单行的事务性操作，暂不支持跨行事务。
4. 支持将单元格当做整型计数器使用。
5. 支持在server端执行客户端自定义的脚本（Sawzall脚本），当前只支持数据处理，不支持数据会写到Bigtable。
6. 可与MapReduce计算框架结合使用（可用作输入或输出）。
7. **Building Blocks 。**

Bigtable建立在其他几个Google基础工具的基础上。Bigtable利用分布式的Google File System（GFS）来存储日志和数据文件。Bigtable通常和其它的分布式应用共用集群。Bigtable依赖于一个集群管理系统，来分发job，管理资源，处理机器宕机，以及管理机器状态。

Bigtable使用Google的SSTable文件格式，来存储数据。在SSTable中，数据是持久化不可更改的有序Map，数据可以是任意的字符串数组。SSTable由一系列的块组成，在文件末尾存储了块索引用于定位各个块，当文件打开时，块索引被优先载入到内存。在块索引帮助下，一次文件查找只需要一次磁盘寻道；当然，也可以将整个SSTable载入到内存，则可避免所有磁盘寻道操作。

Bigtable依赖一个高可用、持久化的分布式锁--Chubby。Chubby使用Paxos算法。Chubby提供了一个由目录和小文件组成的namespace，每个目录或者文件都可以充当锁，对一个文件的读写都是原子性的。Chubby客户端持有一个会话，在一个过期期限内，如果客户端无法成功刷新它的会话，客户端就会丢失所有当前持有的锁。Bigtable使用Chubby完成以下几项任务：

1. 确保任何时间至多有一个活跃的master
2. 存储Bigtable数据的bootstrap位置
3. 发现新的tablet server，关闭宕机的tablet server
4. 存储Bigtable的表结构信息
5. 存储访问控制列表

当Chubby服务不可用时，也会导致Bigtable不可用。

1. **具体实现。**

Bigtable的实现主要有三个模块组成：一个链接给所有客户端的库，一个master server，和若干个tablet server。

master节点的作用是：

1. 分配tablet到各个tablet server
2. 发现新增或者被移除的tablet server
3. 均衡tablet server的负载
4. 回收GFS上的文件
5. 管理表结构更改（包括表或者列族的创建）

tablet server的职责是，管理一个tablet集合。tablet server直接处理读写请求，当tablet数据过多，需要发起tablet分裂操作。tablet server可以动态地添加或者移除，以适应集群负载。

与许多单master分布式存储系统相似，客户端数据不经过master，而是直接向tablet server发起请求。由于连tablet的位置信息也不需要通过master节点，master节点的负载是很轻的。

### 1. tablet位置信息

Bigtable使用一个三层结构的B+树来存储tablet位置信息。第一层是一个Chubby文件，存储了root tablet的位置，而root tablet存储了METADATA表所有分片的位置，METADATA表中则存储了用户表所有分片的位置。

图示

描述已自动生成

客户端中缓存了tablet的位置信息，当客户端发现tablet位置信息错误，它会递归地查找B+树，以取得新的位置信息。当客户端首次建立链接时，至多需要三个网络往返时间取得tablet位置信息；但如果是发现METADATA表信息错误，至多需要六次网络往返。为了节省时间，METADATA表存储在内存中；为了进一步节省时间，客户端每次回读取超过一个数据分片，当它读取METADATA表的信息时，这称为预读取。

### 2. tablet分配

master节点会追踪所有正在使用的tablet位置。

1. 当一个tablet server进程启动，首先会在chubby的对应目录下创建一个唯一命名的文件，并通过这个文件获取一个排它锁。当一个tablet server丢失了它的排它锁，那么分配到这台server的所有tablet就停止服务了。当然，如果这个文件仍然存在，tablet server会尝试重新获取锁；而如果这个文件已被删除，那么tablet server会选择杀死自己。
2. master通过监控chubby对应的目录，获取所有的tablet server。master节点会定期轮询这些tablet server，一旦发现某台tablet server丢失了它的chubby锁，或者无响应，master节点会尝试获取对应文件的互斥锁，并删除对应文件，然后master就可以重新分配对应的tablet。
3. master一旦发现自己的chubby会话过期失效，就会选择杀死自己。
4. 当master节点被集群管理系统启动时，master几点首先需要扫描得到所有的tablet分配情况，有以下四步：  
   ● 获取一个唯一的master chubby锁，确保master进程的唯一。  
   ● 扫描chubby对应目录，获取所有服务中的server  
   ● 与所有server通信，获取所有已分配的tablet  
   ● 扫描METADATA表，将不在已分配名单上的tablet加入待分配名单  
   ● 向server发送tablet load请求，分配对应的tablet
5. tablet拆分操作由server发起，当server将新的tablet信息记录到METADATA表，这个操作就被提交了。如果拆分被提交，即使未能成功通知到master，当master需要重新分配旧的tablet时，这个变化也会被发现。

### 3. tablet服务详述

tablet数据被记录在GFS中。所有的更新首先会被写入一份日志，之后再写入内存中一个有序的缓存--memtable，而旧的更新记录则被永久化成一些列的SSTable文件。当需要恢复一个tablet时，server除了需要读取对应的SSTable文件索引之外，还需要重放日志中保存的更新记录，以恢复memtable中的内容。

图示

描述已自动生成

当接收到读写请求时，server需要验证请求的合法性，另外也需要检查对应权限。读请求的处理是基于SSTable和memtable的合并视图进行的，因为SSTable和memtable都是按字母序排列的，合并视图是相当方便高效的。

### 4. compactions

**minor compaction**：随着更新操作进行，memtable中的数据膨胀，当触发阈值时，memtable中的数据被刷写到GFS，形成一个新的SSTable文件。好处有二：可以减少server的内存使用；当tablet需要被恢复，可以减少commit log中需要重放的操作记录。

**merging compaction**：随着minor compaction的进行，形成大量磁盘小文件，这样处理读请求时就需要处理大量SSTable文件，形成不便；merging compaction的作用就是将多个SSTable文件和memtable中的记录，重写成一个新的SSTable文件。

**major compaction**：major compaction是特殊的merging compaction操作，因为它会将所有的SSTable重写成一个新的SSTable；另外，旧的无效数据的删除也是在major compaction中进行；Bigtable会定期执行major compaction，回收资源。

1. **优化。**

### 1. 位置组

客户端可以将多个列族定义成一个位置组（locality group）。同一locality group的数据会被存在同一个SSTable文件中，因此最好将会同时访问，有相似特性的数据放在同一个locality group中，用户可以为每个locality group定义一些特性，包括是否优先存在内存中。

### 2. 压缩

客户端可以为每个locality group定义SSTable文件是否压缩存储，以及用哪一种压缩格式。bigtable的数据是每个block单独压缩的，比起整个文件一起压缩，分块压缩会浪费一些空间，但是在读取某个块数据的时候，bigtable就不必解压整个文件了。

### 3. 缓存以得到更好的读效率

bigtable有两重缓存：

* **scan cache**：high-level，缓存用户查找的key-value键值对，当用户重复读取相同的数据时，有更好表现。
* **block cache**：low-level，每次缓存从GFS上读到的SSTable块，当用户读取相邻数据时，有更好表现。

### 4. 布隆过滤器

为每个SSTable创建一个布隆过滤器，用于判断特定的行或者列是否在对应的SSTable中。借助布隆过滤器，我们需要花费少量的内存用于存储布隆过滤器，却可以大大减少查找不存在的行或列所需要的磁盘寻道时间。

### 5. commit-log实现

commit-log存储在GFS上，同一server上的所有tablet共享同一个commit-log，因为如果一个数据分片一个log的话，会导致大量的并发写GFS日志文件，耗费大量的时间在磁盘寻道上；并且，批量提交的批次大小也会变小，导致批量提交的优化效果变差。

然而，如果一台server挂掉了，分配到这台server上的所有tablet会被重新分配到其他服务器，而这些tablet的commit-log在同一文件里，那么，假如这些tablet被重新分配到其他100台服务器上，那么这份commit-log会被读取100次。为了避免这个问题，在读取commit-log之前，master会发起对log中的记录按照<table, row, name, log sequence number>排序，log会按照特定大小分片，散发给多台服务器并行排序。排序完成之后，需要读取log的server只需要一次磁盘寻道以及一次顺序读取，就可以读完自己载入的tablet的相关记录。

另外，为了避免偶尔可能出现的commit-log写入卡住问题，一个server上通常有两个写log线程及对应的两个log文件，一个写入线程卡住则切换到另一线程，为了避免同一记录重复记录log，每个操作会有一个序列号用于去重。

### 6. 加速tablet恢复

当master指令将一个tablet从一台server移动到另一台，原先的server会立刻对该tablet执行一次minor compaction，将内存中的数据先刷写到磁盘，这样当tablet移动到另一台server的时候，可以减少需要从log中恢复的未持久化的记录。第一次compact执行过后，原先的server停止接收新请求，并立即执行下一次minor compaction（这次用时会很短），然后卸载该tablet，tablet移动到新的server之后，新的server就不必读取log文件恢复内存中的记录了。

### 7. 利用不变性

SSTable文件的数据不变性为具体实施带来许多便利。例如，在并发读SSTable数据文件时，并发控制变得十分简单；删除旧数据的操作简化为从METADATA表中删除对应SSTable文件的记录；tablet分裂时，不需要立刻为子分片生成两份新的SSTable文件，子分片可以共享旧的SSTable文件。

memtable需要同时处理读写请求，为了简化并发操作，memtable中的行都是copy-on-write的，这样就能读写并发进行。

**八、经验教训**。

1. 分布式系统易受多种错误影响，不仅仅局限于网络分区和故障停止。我们曾遇见，内存和网络损坏、很大的时钟偏差、被挂起的机器、扩展和非对称的网络分区、我们依赖的模块系统出现bug、GFS资源溢出，以及预见和未预见的硬件维护等等。
2. 在确定确切的用户使用场景之前，不要急着添加新的功能特性。
3. 做好适当的系统级别监控非常重要。
4. 保持简单的设计。