BigTable读后感

Bigtable是一个分布式的结构化数据存储系统，它的设计目的是能够可靠的处理PB级别的数据，并且能够部署到上千台机器上。到现在，bigTable已经实现了适用性广、可扩展、高性能和高可用性，Bigtable不支持完整的关系数据模型；与之相反，Bigtable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，也就是说，bigTable中数据是没有格式的，用户自己定义格式，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。最后，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。

**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。表中的行关键字可以是任意的字符串，对同一个行关键字的读或者写操作都是原子的，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。每个分区叫做一个”Tablet”，Tablet是数据分布和负载均衡调整的最小单位。列关键字组成的集合叫做“列族“，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，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。

在Bigtable中，表的每一个数据项都可以包含同一份数据的不同版本；不同版本的数据通过时间戳来索引。Bigtable和用户都可以给时间戳赋值，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。

**API**

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，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。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还允许把数据项用做整数计数器。最后，Bigtable允许用户在服务器的地址空间内执行脚本程序。

**BigTable构件**

Bigtable是建立在其它的几个Google基础构件上的。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集群通常运行在一个共享的机器池中，池中的机器还会运行其它的各种各样的分布式应用程序，BigTable的进程经常要和其它应用的进程共享机器。BigTable依赖集群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。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。BigTable还依赖一个高可用的、序列化的分布式锁服务组件，叫做Chubby。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。当有副本失效的时候，Chubby使用Paxos算法来保证副本的一致性。Bigtable使用Chubby完成以下的几个任务：确保在任何给定的时间内最多只有一个活动的Master副本；存储BigTable数据的自引导指令的位置；查找Tablet服务器，以及在Tablet服务器失效时进行善后；存储BigTable的模式信息；以及存储访问控制列表。

**介绍**

Bigtable包括了三个主要的组件：链接到客户程序中的库、一个Master服务器和多个Tablet服务器。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服务器主要负责以下工作：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。除此之外，它还处理对模式的相关修改操作，例如建立表和列族。 每个Tablet服务器都管理一个Tablet的集合（通常每个服务器有大约数十个至上千个Tablet）。

**Tablet的位置**

BigTable使用一个三层的、类似Ｂ+树的结构存储Tablet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。

**Tablet的分配**

在任何一个时刻，一个Tablet只能分配给一个Tablet服务器。Master服务器记录了当前有哪些活跃的Tablet服务器、哪些Tablet分配给了哪些Tablet服务器、哪些Tablet还没有被分配。当一个Tablet还没有被分配、并且刚好有一个Tablet服务器有足够的空闲空间装载该Tablet时，Master服务器会给这个Tablet服务器发送一个装载请求，把Tablet分配给这个服务器。

Master服务器负责检查一个Tablet服务器是否已经不再为它的Tablet提供服务了，并且要尽快重新分配它加载的Tablet。Master服务器通过轮询Tablet服务器文件锁的状态来检测何时Tablet服务器不再为Tablet提供服务。

Master服务器在启动的时候执行以下步骤：

1. Master服务器从Chubby获取一个唯一的Master锁，用来阻止创建其它的Master服务器实例；

2. Master服务器扫描Chubby的服务器文件锁存储目录，获取当前正在运行的服务器列表；

3. Master服务器和所有的正在运行的Tablet表服务器通信，获取每个Tablet服务器上Tablet的分配信息；

4. Master服务器扫描METADATA表获取所有的Tablet的集合。在扫描的过程中，当Master服务器发现了一个还没有分配的Tablet，Master服务器就将这个Tablet加入未分配的Tablet集合等待合适的时机分配。

**Tablet服务**

Tablet的持久化状态信息保存在GFS上。更新操作提交到REDO日志中。在这些更新操作中，最近提交的那些存放在一个排序的缓存中，我们称这个缓存为memtable；较早的更新存放在一系列SSTable中。为了恢复一个Tablet，Tablet服务器首先从METADATA表中读取它的元数据。Tablet的元数据包含了组成这个Tablet的SSTable的列表，以及一系列的Redo Point ，这些Redo Point指向可能含有该Tablet数据的已提交的日 志记录。Tablet服务器把SSTable的索引读进内存，之后通过重复Redo Point之后提交的更新来重建memtable。

Compactions

随着写操作的执行，memtable的大小不断增加。当memtable的尺寸到达一个门限值的时候，这个memtable就会被冻结，然后创建一个新的memtable；被冻结住memtable会被转换成SSTable，然后写入GFS。Minor Compaction过程有两个目的：shrink Tablet服务器使用的内存，以及在服务器灾难恢复过程中，减少必须从提交日志里读取的数据量。在Compaction过程中，正在进行的读写操作仍能继续。

**结论**

对于BigTable,其对行事务提供一致性，对于跨行跨表的操作，无法提供强一致性。但是bigtable的优点是线性扩展性好，当一个tablet server 宕机了，master节点会将数据立刻rebalance到其他的机器上。由于tablet server 宕机会暂时无法提供服务，因此tablet server并不适合在线交易的应用，只适合离线或者半离线的应用。