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。