**GFS及BigTable阅读整理及读后感**

姓名：胡锦浩 学号：18301068

## 一、GFS文件系统

### 一、概述

Google GFS文件系统，实现了一个面向大规模，数据密集型应用的，可伸缩的分布式文件系统，相比于传统的分布式文件系统，GFS不仅满足了性能，可伸缩性，可靠性，可用性，而且在设计上，更多的考虑了应用的负载情况和技术环境，比如：第一，针对组件失效问题，GFS提供了持续的监控，错误检测，灾难冗余以及自动恢复机制；第二，针对文件数量多，尺寸大的问题，实现了master+many chunk server 的机制；第三，针对海量文件的访问方式，GFS实现了数据的追加操作，而不是覆盖原有的数据，保证了性能优化和原子性；第四，应用程序和文件系统api的协同设计提高了系统的灵活性

### 二、GFS架构方式

GFS架构包括一个master节点和多个chunk服务器，并且同时被多个客户端访问。GFS存储的文件被分割为固定大小的chunk，在建立chunk时，master服务器会给chunk分配标示，GFS充分考虑了文件存储的可靠性，将同一个数据块复制到3个服务器上，即使当chunk服务器发生意外后，能保证数据不丢失。

### 三、客户端读取数据流程分析

1.客户端记录文件名和程序指定的字节偏移，根据固定的chunk大小，转换为文件的chunk索引，之后将chunk索引发送给master节点

2.master节点将相应的chunk标识和副本的位置发送给客户端

3.客户端用文件名和chunk索引作为key索引缓存这些信息

4.客户端发送一个请求到距离自己最近的一个副本，包含chunk标识和字节范围

5.副本返回相应的信息

### 四、master对副本的管理

master节点管理了所有文件的元数据，而且还管理着整个系统的所有chunk副本

第一，使用锁结构，保证了同一目录下的并行操作。这种锁结构与操作系统中的同步机制是一致的，均是在执行操作之前读取当前节点的锁的状态信息，包含读取锁和写入锁，执行操作之后释放锁的机制

第二，chunk副本的创建，复制与负载均衡。在我们创建Chunk及其副本时，我们应该充分考虑平衡服务器之间的硬盘使用率和访问频率。在创建chunk及副本时，如果我们只在一台服务器上存储chunk，直到该服务器满载之后再更改服务器进行存储，就会造成很多服务器利用率不高，部分服务器过载且访问次数较高，造成服务器成为热点。其次，当一台chunk服务器发生意外存储的副本被破坏或chunk副本的复制因数提高，master节点需要重新进行复制，而复制的优先级取决于chunk现有的数量和复制因数的差值或者chunk的活跃度，master会优先选择优先级较高的chunk并进行复制，在选择服务器时，遵循2+1的选择模式，保证了即使一个机架出现意外，数据也不会丢失

第三，垃圾回收和过期失效的副本检测。GFS提供了文件删除后的垃圾回收机制，GFS在文件删除后不会立刻回收可用的物理空间而是采用惰性的策略，master会以日志的方式将删除的操作记录下来，将文件名改为包含时间戳的隐藏的名字，当master节点对文件系统进行常规扫描时，它会删除所有三天前的隐藏文件，在文件还没有被真正删除时，可以用新的特殊的名字读取（包含时间戳的隐藏的名字），也可以通过隐藏文件改名为正常显示的文件的名字来恢复文件。在这种方式下，垃圾回收会更加可靠，为意外的，不可逆转的删除操作提供了安全保证，同时，master定期对数据批量删除，开销会被分散。与此同时，master与chunk之间会进行交互的心跳检测，即master向chunk server发送消息，chunk server收到后回复，如果没有回复，会令未回复的chunk周围的服务器给该服务器发送消息，如果仍然未收到回复，判定chunk不可用，master会根据副本恢复该chunk中的节点信息至新的chunk server中

### 五、总结

Google GFS文件系统支持大量并发读写操作，具有很大的吞吐量，通过master+chunk server的分布式存储机制，可以存储海量的数据并且易于管理，与此同时，GFS提供了监控，错误检测，灾难冗余以及自动恢复，保证了数据不会丢失。在我们日常写代码的过程中，我们能够找到与GFS 实现方式相类似的算法程序，如文件索引，副本备份等，我们应该学习他的分布式管理的思维，更多的考虑系统的错误检测与恢复机制，考虑程序的算法效率和耦合度，并将其应用到我们的程序中。

## 二、BigTable

### 一、概述

BigTable是一个分布式的结构化的存储系统，用于解决GFS无法对结构化数据进行访问与管理的结构化数据存储访问问题，相比较于其他数据库系统，BigTable提供了和并行数据库和内存数据库完全不同的接口，同时，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，用户也可以自己推测底层存储数据的位置相关性

### 二、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，通过三维（row 行关键字、column 列关键字、time 时间戳）方式定位数据，相比于我们所使用的关系型数库如oracle或mysq，BigTable使用了三维的数据存储方式，可以记录时间戳，进行版本更迭，同时value是一个未经过解析的byte数组

行关键词通过字典顺序来组织数据按照行关键字的范围进行数据划分，表中的每一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。由于字典序相接近的两个行关键字数据被存储在接近的位置，使得存取效率较高。在选择存储方式时，我们应该充分考虑数据的位置相关性。

列关键字即列族是访问控制的基本单位，BigTable将数据表中的列先划分为不同的列族，在列族中还可以再定义相应的列关键字，通过这种方式，使得BigTable将相同类型的列聚集为一个族来统一管理，甚至可以统一进行数据压缩，方便了数据管理，也提高了数据存储的灵活度，数据的访问和控制都是基于列族的内容进行操作的。

时间戳，表的每一项可以包含同一份数据的不同版本，不同版本通过时间戳索引。数据项中不同版本的数据按照时间戳倒序排列。

### 三、BigTable构件

BigTable 使用 Google 的分布式文件系统(GFS)存储日志文件和数据文件，采用Google SSTable格式文件存储内部数据，在打开SSTable文件时，块索引被加载到内存，每次查找可以通过一次磁盘搜索完成：首先二分查找在内存中的块索引找数据块的位置，然后再从硬盘读取相应的数据块。

BigTable 依赖一个高可用的、序列化的分布式锁服务组件，即Chubby，chubby完成的功能有：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； 3. 查找 Tablet 服务器，以及在 Tablet 服务器失效时进行善后； 4. 存储 BigTable 的模式信息； 5. 以及存储访问控制列表。

### 四、BigTable组件

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。Google采用了master+chunk server的模式，在主服务器上存储了各个tablet索引的元数据，然后在tablet中提供了对元数据索引，可根据这个索引找到相应的信息，返回给客户端

库函数主要负责服务器与客户端的通信

主服务器 (master) 主服务器上不存储子表，也不是用来提供表定位信息的，而是主要负责子表服务器的分配、负载均衡，监控子表服务器的状态，当子表服务器的租约到后仍然没有回应则要重新安排新的子表服务器来代替，同时当子表服务器的所存的子表过大的时候还要分配新的子表服务器进行负载均衡。同时，主服务器还要负责处理表模式更改、列族增加和GFS上垃圾回收等任务。

子表服务器 (tablet server) BigTable在存储表的时候会将表划分为一个个子表来进行存储。子表服务器上存储着子表信息，由于为了减小主服务器的负载，数据请求不会经过主服务器，子表服务器还需要直接响应客户机对字表服务器上存储的子表的读和写操作。在所存储的子表过大的时候需要对子表进行切分操作。需要注意的是，子表服务器也不是直接存放数据的，数据只是存放在GFS中，然后由子服务器来进行分片管理

### 五、存储结构

BigTable采用三层的存储结构存储table的位置信息

第一层：存储在 Chubby 中，它包含了 Root Tablet 的位置信息

第二层：存储了metadata table所有的tablet的位置信息，根子表随着大小的增长是不会被分割的。

第三层：保存其他用户数据表的子表信息

客户端想要定位某个 Tablet 时，便会递归地安装上述层次向下求得位置，并把中间获得的结果缓存在自己的内存中。如果某一时刻客户端发现缓存在内存中的地址已不再有效，它便会再次递归地沿着上述层次向上，最终再次向下求得所需 Tablet 的位置。

### 六、Tablet分配

当一个子表服务器启动的时候，它会在Chubby特定的目录下建立一个自己的文件并获得互斥锁，主服务器通过文件来监控存在哪些子表服务器，通过周期性尝试获取这些文件的互斥锁来确认这些子表服务器是否还在正常工作。当子表服务器失去与Chubby的连接后，就会失去这个互斥锁。但是只要子表服务器上的数据还存在并且Chubby相应文件还存在，它还会不断试图请求会这个互斥锁。一旦主服务器获得了这个互斥锁，它就会删除这个文件，导致子表服务器的最终停止。而子表服务器主动停止服务的时候，也会释放这个互斥锁，以便主服务器更快意识到这个子表服务器的退出。

当主服务器和Chubby连接被断开后，当前的主服务器会主动关闭自己，这时候系统就会重新选择一个新的主服务器出来。主服务器启动时，会首先获取一个Chubby上的master锁以防止其他服务器同时成为主服务器；然后新主服务器会扫描Chubby的特定目录尝试获取互斥锁来获得当前正在工作的子表服务器；之后再询问每个子表服务器被分配的子表；最后再统计METADATA子表中还未被分配的子表，准备将其分配

### 七、性能提升

BigTable提供了提升性能的相关方法，包括局部性群组，压缩，bloom过滤器，commit日志实现，tablet恢复提速和利用不变性