BigTable读后感

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

Bigtable中的行关键字可以是任意的字符串，并且每行的读写操作都是原子的；Bigtable中的行关键字是按照字典顺序排序存储的，表中的行都可以进行动态分区，每个分区叫tablet，tablet是数据分布和负载均衡的最小单位。由于行键是按照字典序存储的，所以查询时以行关键字作为条件查询速度毫秒级。

列关键字组成的集合叫做“列族”，列族时访问控制的基本单位。列关键字的命名语法如下：列族：限定词。

每行数据都会有用来当作版本号的时间戳，可以系统自动赋值，也可以用户自己指定。最新的数据行排在最前面。还可以利用时间戳来进行垃圾收集。

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内部存储数据的问文件时Google SSTable格式的。SSTable是一个持久化的、排序的、不可更改的Map<key,value>结构，其值都是任意的byte串，因此使用key查询速度很快。Big Table还依赖一个高可用的、序列化的分布式锁服务组件——Chubby。BigTable使用Chubby完成以下几个任务：

确保在任何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模式信息
存储访问控制列表。
BigTable包括了三个主要的组件：链接到客户程序的库、一个Mater服务器和多个Tablet。针对系统工作负载的变化情况，BigTable可以动态的向集群添加或者删除Tablet服务器。
Master服务器主要为Tablet服务器分配Tablets、检测新加入的或者过期失效的Tablet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，还处理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，每个Tablet的服务器负责处理它所加载的Tablet的读写操作，以及在Tablets过大时，对其进行分割。

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。

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那些活跃的Tablet服务器、那些Tablet分配给了那些Tablet服务器、那些Tablet还没有被分配。

BigTable使用Chubby跟踪记录Tablet服务器的状态。当一个Tablet服务器启动时，它在Chubby的一个指定目录下建立一个有唯一性名字的文件，并且获取该文件的独占锁。Master服务器实时监控着这目录，因此Master服务能够知道有新的Tablet服务器加入了。只要文件存在Tablet服务器就会试图重新获得对该文件的独占锁，如果文件不存在了，那么
Tablet服务器就不能在提供服务了。

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；
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；
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；
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的持久化状态信息保存在GFS上。更新操作提交到REDO日志中。这些更新操作中，最近提交的那些放在一个排序的缓存中，我们称这个缓存为memtable；较早更新存放在一系列的SSTable中。

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；被冻结住的memtable会被转换成SSTable，然后写入GFS。

客户程序可以将多个列族组合成一个局部性群族。对Tablet中的每个局部性群族都生成一个单独的SSTable。将同城不会一起访问的列族分割成不同的局部性群族可以提高读取操作的效率

客户程序可以控制一个局部性群族的SSTable是否需要压缩，一般使用两遍的、可定制的压缩

为了提高读操作的性能，Tablet服务器使用二级缓存的策略，一级用来缓存Tablet服务器通过SSTable接口的Key-Value对；Block是二级缓存，用来缓存从GFS读取的SSTable的Block。

整个BigTable设计符合大部分大数据程序的需求，打破了关系型数据库的结构化存储，能够部署在成千上万台服务器上，可以存储PB级数据，对整个互联网行业的快速发展提供了坚实的理论基础与成功案例。

GFS读后感

Google文件系统GFS是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，但可以提供容错功能。它可以给大量的用户提供总体性能较高的服务。
GFS与过去的 分布式文件系统有很多相同的目标，但GFS的设计受到了当前及预期的应用方面的工作量及技术环境的驱动，这反映了 它与早期的文件系统明显不同的设想。这就需要对传统的选择进行重新检验并进行完全不同的设计观点的探索。
一个GFS集群文件系统可能是多层分布的。一般情况下是成千上万个文件块服务器分布于不同的机架上，而这些文件块服务器又被分布于不同机架上的客户来访问。因此，不同机架上的两台机器之间的通信可能通过一个或多个交换机。 数据块 冗余配置策略要达到连个目的：最大的数据可靠性和可用性，最大的网络带宽利用率。因此，如果仅仅把数据的拷贝置于不同的机器上很难满足这两个要求，必须在不同的机架上进行数据备份。这样即使整个机架被毁或是掉线，也能确保数据的正常使用。这也使数据传输，尤其是读数据，可以充分利用带宽，访问到多个机架，而写操作，则不得不涉及到更多的机架。
在一个文件被删除之后，GFS并不立即收回磁盘空间，而是等到垃圾收集程序在文件和 数据块级的的检查中收回。当一个文件被应用程序删除之后，MASTER会立即记录下这些变化，但文件所占用的资源却不会被立即收回，而是重新给文件命了一个隐藏的名字，并附上了删除的 时间戳。在MASTER定期检查名字空间时，它删除超过三天（可以设定）的隐藏的文件。在此之前，可以以一个新的名字来读文件，还可以以前的名字恢复。当隐藏的文件在名字空间中被删除以后，它在内存中的元数据即被擦除，这就有效地切断了他和所有 数据块的联系。在一个相似的定期的名字空间检查中，MASTER确认孤儿 数据块（不属于任何文件）并擦除他的元数据，在和MASTER的心跳信 息交换中，每个服务器报告他所拥有的数据块，MASTER返回元数据不在内存的数据块，服务器即可以删除这些数据块。

mapReduce读后感

1. MapReduce的思想源自函数式编程(lisp类语言)中的Map和Reduce函数，MapReduce将该思想应用在了 分布式环境中，将任务由单机进程变为了分布式系统环境下的Task，并解决任务调度、中间结果传输、错误 处理等关键问题;
2. Mapper数量M和Reducer数量R之间有没有什么关系?M>>R?M~R?R的数量不能太多，因为M产生的中 间结果会被存储为本地临时文件，且这些临时文件根据hash(key) % R被分别存储在R个文件中，如果R过多会 造成本地文件系统下的临时文件数量较多，影响性能;
3. 每个Worker既可以执行Map任务也可以执行Reduce任务吗?应该是的，保证资源的高效利用。Map和 Reduce任务作为一个个的Task被调度至Worker上，Hadoop的MapReduce框架实现了调度算法YARN。
4. Reducer和Mapper可以并行处理?应该可以，两者之间其实没什么耦合，只要数据准备好了，就可以运行。 Map任务完成后，Worker即可向Master节点汇报完成状态以及其产生的临时文件位置，Master即可通知 Reduce Task。
5. Mapper可以级联吗?Reducer可以级联吗?应该可以，甚至在很多时候是必要的，这也是该模型的强大之 处:提供了一个足够简单的框架，但是可以高度扩展
6. 数据源以及最终结果都会存储在磁盘上(GFS)，但Map阶段产生的中间结果存储在Map任务处理本地。每 个阶段的交互通过RPC请求经由网络传输，瓶颈在于网络传输
7. Mapper之间、Mapper-Reducer之间、Reducer之间没有任何通讯，非常的松耦合，由Master角色负责统一 调度，保持了架构清晰和较强的可扩展性;Master节点存储的状态数据较少，压力不大。根据论文描述，， 单Master集群可扩展至数千Worker节点;
8. Map阶段产生的中间结果存在本地磁盘上(为什么不直接使用GFS存储?效率的原因?同时也减少网络等资 源消耗)
9. 批处理而非流式处理:无法做到数据边产生边处理。Mapper产生的中间结果数据只能全部写入临时文件后才 能通知Reducer进行下一阶段处理，实时性较差