**Google Bigtable读后感**

**1前言**

随着社会的进步和互联网信息技术行业的高速发展，海量数据出现在我们的生活中，大数据作为科技时代的发展产物应运而生。然而传统的关系型数据库面对海量数据的存储问题时，处理大量数据的读写性能较差且扩展性较低。为解决这一问题，Google研发了Bigtable，并在2006年的OSDI大会上发表了Google Bigtable的论文，该篇论文也与Google File System、Google MapReduce共同称之为Google的三架马车。在本学期的非关系型数据库课程上，为进一步了解和学习相关的知识，我阅读了Bigtable这篇论文并查阅了相关资料，现将个人学习所得的一些认识整理如下。

**2简介**

（1）与数据库比较：

a.相似：使用了很多数据库的实现策略。

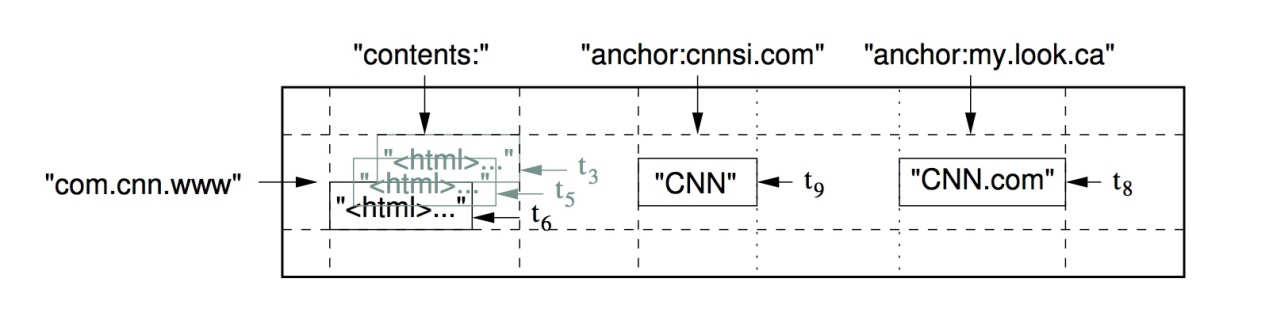
b.不同：不支持完整的关系数据模型，与之相反，它为客户提供了简单的数据模型，客户利用这个模型可以动态控制数据的分布和格式，也可以自己推测底层存储数据的位置相关性。

（2）数据下标是行名和列名（名字可以是任意字符串）

（3）Bigtable将存储的数据都视为字符串；客户通过仔细选择数据的模式（schema），可以控制数据位置的相关性；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。

**3数据模型**

Bigtable是一个分布式的结构化数据存储系统，可以用来处理大量的数据，尤其是能够可靠的处理PB级的数据，并且能够部署到上千台服务器上。它具有适用性广泛、可扩展、高性能、高可用性的特点。从本质上来说，Bigtable是一个稀疏的、分布式的、持久化存储的多维度排序的Map，其中Map的key是行关键字(Row key，一级索引)、列关键字(Column key，二级索引)、时间戳(timestamp，三级索引)，Map中的每一个value都是一个未经解析的byte数组，其数据模式如下:（row:string,column:string,time:int64）->string



3.1行关键字(Row key)

由于Bigtable会按照Row key进行字典排列，所以接近的两行会紧密相邻，这也是Bigtable能高效读写的原因之一。一定范围内相邻的row可组成一个集合，称为tablet，它是分布式存储和资源调度的最小单元。

3.2列关键字(Column key)

列关键字的命名语法为family:qualifier（列族：限定词），用户在使用前必须首先声明表中有哪些列族，然后才能在列族中任何的列关键字下存放数据。由于同一个列族中存储的数据通常属于同一类型，Bigtable还会对属于同一列族的数据进行合并压缩。一张表的列族不能有太多（最多几百个），但可以有无限多个列。

3.3时间戳 (TimeStamp)

随着时间的变化，同一数据项可能有多个不同的版本，我们就通过时间戳来区分。时间戳的类型是64位整型，Bigtable可以给时间戳赋值，也可以由用户程序给时间戳赋值。在数据项中，不同版本的时间戳以倒序排列，即最新的数据排在最前面，这样最新的数据可以先被读取。为减轻管理负担，可以设置只保存最后几个版本的数据。

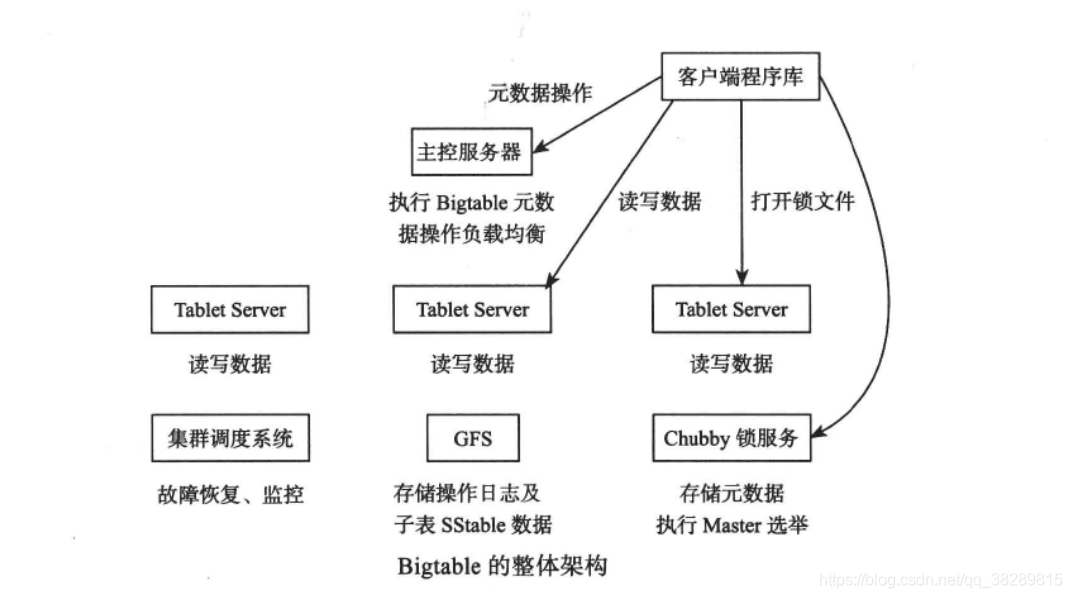
**4技术架构**

4.1Master和Tablet Server

Bigtable包括了三个主要的组件：连接到客户程序中的库、一个Master（主服务器）和多个Tablet Server（数据服务器）。

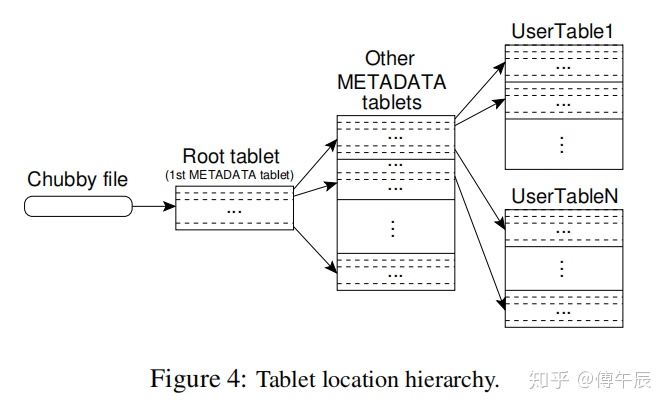
Master 服务器主要负责以下工作:为Tablet Server分配 tablets、检测新加入的或者过期失效的Tablet Server、对Tablet Server进行负载均衡、以及对保存在 GFS 上的文件进行垃圾收集。除此之外，它还还负责管理Schema（模式）修改操作，例如表和列族的创建和删除。

Bigtable将每个table分成多个tablets，每个Tablet Server都管理一个 由Master指定的tablet 集合，Tablet Server负责处理它所加载的tablet 的读写操作，以及在 tablet过大时对其进行分割。初始状态下，一个表只有一个 tablet。随着表中数据的增长，它被自动分割成多个 tablets。



4.2tablet的定位

我们使用一个三层的、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维护的是METADATA表的位置信息。每个METADATA表都包含了用户tablet集合的位置信息。Root tablet实际上是METADATA表的第一个tablet，只不过它比较特殊，因为它永远不会被分割，这就保证了tablet的位置信息存储结构不会超过三层。示例图如下所示：



4.3tablet的分配

BigTable依赖一个高可用的、序列化的分布式锁服务组件，叫做 Chubby。Chubby跟踪记录Tablet Server的状态。当一个Tablet Server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实时监控着这个目录，因此Master能够知道有新的Tablet Server加入了。如果Tablet Server丢失了Chubby上的独占锁，只要文件还在，他就会一直试图重新获得独占锁，除非文件不存在了，Tablet Server就会自行退出。

在明确集群中有哪些Tablet Server后，Master便将tablet分配给Tablet Server。任一时刻一个 tablet 只能被分配给一个Tablet Server。Master 会通过向 Tablet Server 发送装载请求进行分配。Master通过轮询Tablet Server文件锁的状态来检测何时Tablet Server不再为tablet 提供服务。当Master在检测到 Tablet Server 失效（互斥锁丢失）后，会尝试在 Chubby 上获取该Tablet Server对应的文件的互斥锁，并在成功获取后删除该文件，确保Tablet Server不再给tablet提供服务。之后，Master便将该Tablet Server的tablet放入未分配的tablet集合，以便之后分配给其他Tablet Server。

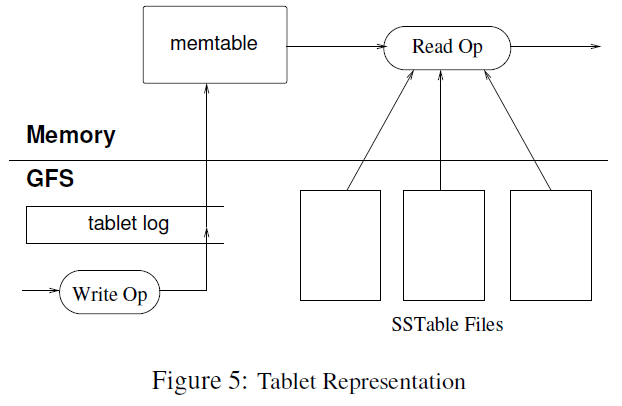
如果Master与Chubby之间的通信连接断开，那么Master 便会认为自己已经失效并自动关闭。当Master失效后，会有一个新Master。新Master首先要了解当前Tablet的分配状态，之后才能够进行恢复，修改分配状态。Master服务器在启动的时候执行以下步骤：

1. Master服务器从Chubby获取一个唯一的Master锁，确保不会有其它的Master服务器实例启动；
2. Master服务器扫描Chubby的服务器文件锁存储目录，获取当前有效的Tablet Server列表；
3. Master服务器与所有正在运行的Tablet表服务器通信，表明自己新 Master 的身份，获取每个Tablet Server上tablet的分配信息；
4. Master服务器扫描Root Tablet和METADATA表获取所有的tablet的集合，确保tablet已完成分配，并对未分配的tablet重新进行分配。

4.4tablet的读写与存储

4.4.1memtable、SSTable和log

Tablet的数据实际上存储在GFS中，由GFS提供数据的冗余备份。一个tablet由一个位于内存内的 memtable、若干个位于GFS上的SSTable文件、以及一个log文件组成。如图所示：



其中memtable是内存有序中的数据结构，而log、SSTable则会持久化到GFS中。在进行写操作时，Bigtable首先会先写日志文件，把此次变更记录到log中。然后再把数据插入到memtable中并保持其内部数据的有序。memtable的数据量到达一定情况下的时候就会以SSTable的形式持久化存储到GFS中。在进行读操作时，Tablet Server会作类似的完整性和权限检查。然后先从memtable中尝试获取目标数据，如果没有就再从SSTable中进行查找。由于SSTable和memtable是按字典排序的数据结构，因此可以高效生成合并视图。当进行tablet的合并和分割时，正在进行的读写操作能够继续进行。

4.4.2Bigtable压缩

（1）minor compaction

Memtable与SSTable本身都采取了数据不可变的设计思路。执行写操作时，当memtable大小达到阈值时，创建一个新的memtable，将之前的memtable转换为SSTable并写入GFS。这就是minor compaction过程，它有两个目标：减少Tablet Server的内存使用，并且减少了在服务器挂掉后的恢复期间必须从提交日志里读取的数据量。正在进行的读和写操作可以在压缩发生时继续进行。

1. merging compaction

每一个minor compaction都会创建一个新的SSTable。如此往复会产生大量的SSTable，读操作可能需要合并来自大量SSTable的更新。因此我们通过在后台周期性地执行合并压缩（merging compaction）来限制此类文件的数量，这样不仅每次合并需要压缩的数据量少，而且运行时间也相对较短。merging compaction读取几个最近的SSTable和memtable的内容合并成一个新的SSTable。merging compaction完成后原来的SSTables和memtable即可被清除。

（3）major compaction

将所有SSTable重写为一个SSTable的合并压缩称为主要压缩（major compaction）。由non-major compaction生成的SSTable可能包含特殊的删除条目，但是major compaction产生的SSTable不包含删除信息或删除数据。 bigtable循环扫描它所有的tablet，并定期进行major compaction。major compaction允许Bigtable回收已被删除数据所使用的资源，并允许Bigtable确保被删除的数据及时从系统中消失，这对于存储敏感数据的服务非常重要。

**5性能优化**

5.1局部性群组（Locality Group）

用户程序可以将多个列族组合成一个局部性群族。对tablet中的每个局部性群组都会生成一个单独的SSTable。将通常不会一起访问的列族分割成不同的局部性群组可以提高读取操作的效率。

5.2压缩

对于每个SSTable的块（块的大小由局部性群组的优化参数指定）都使用用户指定的压缩格式来压缩。虽然这会让压缩的效率下降，但是我们在读取SSTable的一小部分数据的时候就可以只解压部分文件而不必解压整个文件了，而且压缩后占据的空间也变小了。例如Bentley-McIlroy算法可以从来自同一个主机的页面里找到大量的重复内容。

5.3缓存

为了提高读操作的性能，Tablet Server使用二级缓存的策略。扫描缓存（scan cache）是第一级缓存，主要缓存Tablet Server通过SSTable接口获取的Key-Value对；Block缓存是二级缓存，缓存的是从GFS读取的SSTable的Block。对于经常要重复读取相同数据的应用程序来说，扫描缓存非常有效；对于经常要读取刚刚读过的数据附近的数据的应用程序来说，Block缓存更有用。

5.4布隆过滤器（Bloom filters）

Bigtable采用Bloom Filter算法，通过消耗一定量的内存对特定局部性群组的SSTable构建Bloom Filter，可以在客户端检索记录时利用Bloom Filter快速地排除某些不包含该记录的 SSTable来减少硬盘访问的次数。

5.5commit log

Bigtable使用了Write-Ahead Log的做法来确保数据高可用，但这样会涉及大量对Commit Log的写入，会有大量的磁盘Seek操作。为了避免这一问题，我们设置Tablet Server把其接收到的所有tablet以追加的方式写入到同一个Commit Log文件中。因此一个实际的日志文件中混合了对多个tablet修改的日志记录。当恢复一个tablet的状态的时候，新的Tablet Server要从原来的Tablet Server写的日志中提取修改操作的信息并重新执行。然而，这些操作的日志记录都混合在同一个日志文件中了。为了避免多次读取日志文件，我们首先把日志按照关键字（table，row name，log sequence number）排序。排序后对同一个tablet的修改操作的日志记录就会被连续存放在一起，因此我们只需要做一次磁盘Seek操作、之后顺序读取就可以了。

5.6恢复提速

当Master服务器将一个Tablet从一个Tablet Server移到另外一个Tablet Server时，源Tablet Server会对这个tablet做一次Minor Compaction。这个Compaction操作减少了Tablet Server的日志文件中没有压缩的记录的数量，从而减少了恢复的时间。Compaction完成之后，该服务器就停止为该tablet提供服务。在卸载tablet之前，源tablet Server还会再做一次Minor Compaction以消除前面一次压缩过程中又产生的未归并的记录。第二次Minor Compaction完成以后，tablet就可以被装载到新的Tablet Server上了，并且不需要从日志中进行恢复。

5.7利用不变性

我们在使用Bigtable时，除了SSTable缓存之外的其它部分产生的SSTable都是不变的，我们可以利用这一点对系统进行简化。例如，当从SSTable读取数据的时候，我们不必对文件系统访问操作进行同步。这样就可以非常高效地实现对行的并行操作。

**6实例分析——Google搜索引擎**

随着信息时代的进步，大量的数据涌入人们的生活，Google的搜索引擎因此也要存储着极其庞大的信息量，包括应用数据、图片数据、网页数据、配置信息等。而Bigtable这一分布式数据存储系统，既能存储大量的关系型数据库不能存储和处理的海量数据，又支持多种数据类型（如URL、图片数据、网页数据、用户个性化设置等数据），还有很高的适用性（目前已经在Google的60多个产品和项目上得到了应用）、扩展性（三层结构存储数据信息，具有线性扩展性，可以简单地增加机器）、分布性（每个表都可以分成多个tablet，每个tablet又可以分配到一个独立的Tablet Server）、操作简单容易部署（底层架构的简单提高了便利性增加了容错性）。

**7结语**

在阅读完Bigtable这篇论文后，我对于大数据的基本模式有了更详细深入的认识，也对Google开发人员的设计理念感到叹服。Bigtable是为了解决Google自身应用需求多样的问题而研发的，同时它很好地解决了大数据时代海量数据存储处理的问题，推动了科技的进步和发展，在各个领域都发挥着至关重要的作用。作为一个高扩展性、高度可靠的数据存储系统，它使用简单的底层架构处理海量数据，既能够高效的处理平面文件，又允许使用者对系统的重要行为进行调整。同时该模型提供的组件比简单的key-value pair更加丰富，它支持稀疏的、半结构化的数据。虽然Bigtable相对于传统的数据库有很多优点，但是它还是有一些不足之处，比如不支持触发器和存储过程、不支持联结或者SQL类型查询等。二者各有所长，我们在应用时也要根据实际需求进行选择和使用。

作为软件行业的学生，我们应该学习这种在处理和解决问题时不满足于固有的方法和思路的开拓创新精神，做到在原有的思维模式基础上进行不断地创新和改进，正是这种大胆创新才有了Bigtable这一开创性颠覆性的NoSQL方案。以及像论文中说的一样“对我们来说，最宝贵的经验是简单设计的价值。”，起初他们设计的协议过于复杂了，并且依赖一些Chubby很少被用到的特性，浪费了大量的时间在调试一些古怪的问题：有些是Bigtable代码的问题，有些是Chubby代码的问题。最后，设计人员重新制订了一个新的、更简单、只使用Chubby最广泛使用的特性的协议。这也启发了我，有些时候往往基础和简单的操作才是解决问题的根本，就像Bigtable依赖简单的底层架构一样，就足以解决大量数据的问题。最后，我还认识到，我的知识积累和涉猎的领域还远远不够，对于有些专业知识只是了解了很浅层的东西，还有待日后深入地学习、挖掘、积累和实践。