**Bigtable读后感**

Bigtable实现了诸多目标，比如适用性广、可扩展性、高性能和高可用性，无论是对需要存储大量数据的应用，还是是对需要及时响应的应用，Bigtable都给出了很好的解决策略。Bigtable的数据下标为行名和列明，当我认真思考过后，觉得这很像redis那种键值的对应关系，只不过Bigtable是多个键对应一个值。也就是行、列、时间戳->值的形式。并且，在介绍当中也提到，Bigtable也可以控制模式参数来控制数据存放的位置，即内存或者硬盘。存放在内存的工作效率自然比存放到硬盘的工作效率高、速度快，但是内存不仅仅费用高，而且一旦机器故障数据就可能没了。所以存放到内存还是存放到硬盘各有千秋吧，也不能说哪个更好一点，我想，Bigtable既然兼容了这两种方式，也能让它适应不同需求的开发吧。

***数据模型部分：***

Bigtable作为一个多维度排序Map，它的索引是行关键字、列关键字和时间戳，而value是一个未经解析的byte数组。和平时的key-value类似，我理解的redis是一种string-string或string-hash或string-set等等形式，其不同的结构在value上；而Bigmap的形式则是固定的(string,string,int64)-string形式。其中，key的第一个参数为行关键字，第二个参数为列关键字，第三个参数为时间戳。

行关键字的字典顺序可以用来组织数据。每一个行范围叫做一个tanle，是数据分布和负载均衡的最小单位。根据论文中所举的Webtable的例子，可以把同一个域名下的网页聚集起来组织成连续的行。根据我的理解，这样做分区，可以让计算机在寻找对应的网页时效率更高，比如说，在1.2.3.4/index到1.2.3.4/login的跳转的效率更高，因为他们在Bigtable中存放的距离很近。这也就是说，当操作只读取很短的行范围数据时效率很高。

列关键字组成的集合叫做列族，一张表的列族数是有限的，在列族创建后，每一个列下面都能存放数据，存放在同一列族下的数据一般属于同一个类型。列关键字的命名语法为：“列族：限定词”。

时间戳可以用来区分数据的不同版本。Bigtable的时间戳类型为64位整型，可以精确到毫秒。例如在Webtable的例子里，一个网页的内容在不同的时间点，网络爬虫抓取的网页内容是不一样的，所以使用时间戳作为参数之一，可以让索引更有逻辑。

***API部分：***

Bigtable提供的API可以对Bigtable进行操作，比如写入或者删除Bigtable中的值、查找值、遍历数据子集等。API提供的操作简单，比如说插入列族:

//找到table位置

Table \*T = Open (string path);

//找到行关键字定位

RowMutation r1 (T , “com.cnn.www” );

//增加增加锚点，列族[www.c-span.org](http://www.c-span.org)下值CNN。

r1.Set(“anchor:www.c-span.org” , “CNN”)

//删除锚点

r1.Delete(“anchor:www.abc.com”);

//执行以上操作

Opertion op;

Apply (&op , &r1);

还有遍历的API，即使用Scanner类的对象指针遍历一行内的所有锚点。在遍历时，Bigtable也可以限定扫描的范围，因为毕竟范围越大，扫描所耗费的时间也越大。能够限定扫描范围也让Bigtable更加轻便。同时，Bigtable还有很多其他的特性，比如原子性的操作可以解决高并发的问题。

我在看Bigtable论文给出的使用API的例子时，发现Bigtable给出的使用示例并没有想象中的那么复杂，无论是怎么使用API，都是面向行、列、时间戳——值这一Bigtable数据模型的操作。并且Bigtable可以和MapReduce一起使用，使用一些Wrapper类可以让他们之间有很好的交互。不得不说Bigtable封装的确实很好，接口的使用简洁明了。

**Bigtable构件部分**

Bigtable的文件存储是依赖于另一篇papper——GFS的，它使用GFS存储日志文件和数据文件，内部存储数据文件格式为Google SSTable格式，即一个持久化的、排序的、不可更改的Map结构。Map结构，正如上面所说，为key-value形式存储数据的。SSTable的查询数据使用的为块索引，每次打开SSTable，块索引都会加载到内存中，只需要一次磁盘搜索即可查找到目标。数据块的位置可以在硬盘中，也可以位于内存当中，存放到内存当中的话就不用访问内存了，并且查找速度也会变快吧。同时，Bigtable还依赖一个高可用、序列化的分布式锁服务组件Chubby，它保障了Bigtable服务的正确性和一致性。

**Bigtable中Master服务器和Tablet服务器**

Master服务器可以为Tablet服务器分配管理的Tablet，检验Tablet服务器的状态和保持其管理的Table服务器的负载均衡。用户在使用Bigtable客户程序时，通常不需要和Master服务器进行通信，所以在实际的应用当中，Master主服务器的负载很轻，重量的任务都配分配给了Tablet子服务器，Master服务器保证负载均衡，让这分布式的结构更加稳定。

一个Tablet只能分配给一个Table服务器，当一个服务器有足够的空闲空间状态Tablet时，Master服务器就会给这个服务器发送一个装在请求，并且分配Tablet。如何知道Tablet服务器的状态呢？这就是上面提到的Chubby的功能了，每当一个Tablet服务器启动时，就会在Chubby的一个指定目录下建立一个唯一的文件，并且获取文件的独占锁。这些功能为Chubby的功能，这也让Master服务器能够监视子Table服务器了。

那么，如果子服务器丢失了对文件的独占锁，或者Master服务器尝试几次和子服务器联系都没有响应，那么Master服务器就会去尝试收回该服务器的文件独占锁。一旦收回了独占锁，就说明Chubby时正常运行的，出问题的是Tablet子服务器，所以主服务器，也就是Master服务器只用收回对改子服务器的服务即可。

在一个Master服务器开启后，会取尝试获取Chubby的Master锁，若是获取到，就会先确定正在运行的Tablet服务器和Tablet的分配情况，如果有未分配的Tablet就添加到待分配列表中。

**空间压缩**

在Bigtable写操作的不断进行，Mem table的大小肯定会不断增加，当大小达到一个确定的阈值是就会创建一个新的Mem table，并将冻结的，也就是达到阈值的table转换成一个SSTable并写入GFS。可以合并所有的SSTable并生成一个新的SSTable以达成压缩的目的，这个过程叫做Major Compaction。

**Bigtable的优化**

看完Bigtable实现的内部原理之后，就是对Bigtable的优化，以让它能够更好的达到用户的高性能、高可用性和高可靠性的性能。优化工作主要有：局部性群组、压缩、通过缓存提高读操作的性能、Bloom过滤器、Commit日志的实现、Tablet恢复提速和利用不变性。

为了提高Bigtable的效率，引入了局部性群组的概念，即客户程序可以将多个列族组合成一个局部性群组，也就是分类。比如一个Webtable，网页的元数据和网页的内容之间并没有过多联系，当我们访问网页元数据，需要的是网页内容，没有必要读取所有的页面内容，所以局部性群组也让读取数据更加快捷、效率更高。

优化还有压缩的优化。客户程序可以控制一个局部性群组的SSTable是否需要压缩和压缩的方式。当我们在读取被压缩的数据时，不需要解压整个文件。这大大的提高了Bigtable对于空间的利用率。

通过缓存也可以提高操作的性能。我们知道，从缓存中读取信息的速度会更快。所以，当一个客户程序经常读取近期度过的数据时，Block缓存更加有用。

Bloom过滤器让用户在读操作中减少对硬盘的访问次数。即使用bloom来查询一个SSTable是否包含所需要的行和列，这样减少了对硬盘的访问，即可提高访问的速度。Bloom过滤器在用于检索一个元素是否在一个集合中时很有效，但是也有一定的误识别率，并且删除困难。

如果每个tablet的操作的commit日志都存放在一个单独的文件，会产生大量的文件，并且会并行地写入gfs，有大量的磁盘seek操作。所以，如果使用单个日志的话即可提高普通操作的性能。同时，在写入GFS日志文件时，会有两个线程，并且只有其中一个线程会在工作。当一个线程在写入的效率很低的时候会切换到另外的线程。这样也在一定程度上解决了写入GFS效率低的问题。

**性能**

当增加系统中的Tablet服务器中的数量时，系统的整体吞吐量会增长，进而提高整体的性能。但是，我们会发现与此同时，单个服务器的吞吐量其实是会降低的。在论文中的解释是，负载均衡算法不能完美的工作，论文中是这么说的，“如果Tablet被移动了，那么在短时间内——一般是1秒内——这个Tablet是不可用的”，还有就是我们的基准程序产生的负载会有波动。并且，没1000B的读操作都会导致64KB大的Block在网络上传输，进而小号了网络中共享的链路，导致我们增加服务器数量时，每个服务器上的吞吐量会降低。当然，从实际情况上考虑，和每一台服务器上的负载考虑，增加服务器的数量是不可避免地，如果解决了这个问题，或者降低读操作时在网络上传输的Block大小，可能会让系统的工作效率大幅增加。

以上的服务器效率降低的问题，可能也是提高Bigtable效率的一个切入点吧，或许未来这种问题能降低或者被解决呢。

**Bigtable的应用**

Bigtable在十几年前就在Google内部有了很多的实践应用了，比如Google Analytics、Google Earth等，可以看到Bigtable的内部设计的灵活性让很多不同领域的应用都能够灵活地使用，它应用的广泛也在一定程度上说明了其较高的兼容性。在2006年的这篇论文中，很有意思地考虑了新用户对Bigtable接口的陌生感，是否能够完美的应用提供的接口。实际上，在十几年前Bigtable已经展示了很高的应用价值，在Google内部有了很多的应用。如今Bigtable的应用也更多了，Bigtable的适用性广、可扩展性、高性能和高可用性的性能在实践过程中的应用价值是不可估量的。