|  |  |  |
| --- | --- | --- |
|  | |  |
|  | | |
| 学院标志 | | |
| 课程名称 :《非关系型数据库》 | | |
| **Bigtable读后感** | | |
|  | | |
| 姓名: | 赖伟江 | |
| 学号 | 18231642 | |
| 授课教师: | 李在 | |

目录

[1.前言 3](#_Toc58359701)

[2.Bigtable Introduction 3](#_Toc58359702)

[3.Data Model 3](#_Toc58359703)

[3.1Rows 3](#_Toc58359704)

[3.2Column Families 4](#_Toc58359705)

[3.3Timestamps 4](#_Toc58359706)

[4.API 4](#_Toc58359707)

[5.Building Blocks 5](#_Toc58359708)

[6.Implementation 5](#_Toc58359709)

[6.1Table Location 6](#_Toc58359710)

[6.2Table Assignment 6](#_Toc58359711)

[6.3Tablet Serving 7](#_Toc58359712)

[6.4Compactions 8](#_Toc58359713)

[7.Refinements 8](#_Toc58359714)

[7.1Locality groups 8](#_Toc58359715)

[7.2Compression 8](#_Toc58359716)

[7.3.Caching for read performance 9](#_Toc58359717)

[7.3.1Bloom filters 9](#_Toc58359718)

[7.3.2Commit-log implementation 9](#_Toc58359719)

[7.3.3Speeding up tablet recovery 10](#_Toc58359720)

[7.3.4Exploiting immutability 10](#_Toc58359721)

[8.Performance Evaluation 10](#_Toc58359722)

[8.1Single tablet-server performance 10](#_Toc58359723)

[8.2Scaling 11](#_Toc58359724)

[9.Real Applications 11](#_Toc58359725)

[9.1Google Analytics 11](#_Toc58359726)

[9.2Google Earth 11](#_Toc58359727)

[9.3Personalized Search 11](#_Toc58359728)

[10.Lessons 11](#_Toc58359729)

# 1.前言

刚好我们这个学期学了数据库和非关系性数据库这两门课，从我直观上Google的三驾马车中Bigtable与这两门课的相关性略大一些，理解起来相比另外两篇应该简单一些，所以我选择了对Bigtable写读后感，但与其说是读后感，我更愿意称这篇文章为学习笔记。

# 2.Bigtable Introduction

Bigtable是一个分布式的结构化数据存储系统，被用来处理海量数据：通常是分布在数千台服务器上的PB级的数据。

实现的目标：适用性广泛、可扩展、高性能和高可用性。

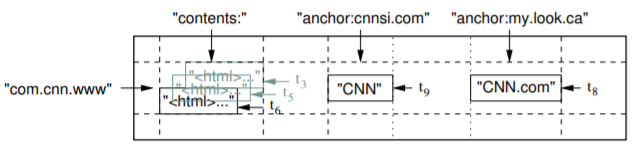
Bigtable将储存的数据都视为字符串

# 3.Data Model

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row key、column key、timestamp；Map中每个value都是一个未经解析的byte数组，其结构如下图所示：



例子如下：



row key：URL

column key：网页属性

timestamp：获取网页的时间

Bigtable就是一种非关系性数据库或许可以认为就是一个个的key，value结构，同时具有timestamp

3.1Rows

表中的row key可以是任意字符串（最大64KB）；

对同一个row key的读写操作是原子的；

Bigtable通过row key的字典顺序来组织数据；

每一行都可以动态分区，每一个分区叫做一个“tablet”，tablet是数据分布和负载均衡调整的最小单位。

发现网站储存的结构是反转URL，好处：可以把同一域名下的网页聚集起来组织成连续的行，把相同域中的网页储存在连续的区域可以让基于主机和域名的分析更加有效。

（对此我的理解是对于一个域名下的内容，这样会有更加快速的索引？网上查了一下后发现还可以大幅提高压缩率，说的比较简单，没有很能get到点）

3.2Column Families

column key组成的集合叫column families，是访问控制的基本单位，存放在同一column families下的所有数据通常都是同一个类型

column key的命名语法：family:qualifier

访问控制、磁盘和内存的使用统计都是在column families层面进行的

3.3Timestamps

在Bigtable中，表的每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timestamp来索引，类型为64位integers，且精确到毫秒的“实时”时间

数据项中不同版本数据项按照timestamp排序，最新的数据排在最前面

每个column families配两个设置参数对废弃版本的数据自动进行垃圾收集，可以只能只保存最后n个版本，或者只保存“足够新”的版本数据

（我认为就是保存最后n个版本或最后n天/小时的版本）

# 4.API

提供对Bigtable的操作

基本的：

增删改查、遍历表中一个数据子集

其他的特性用以支持复杂处理：

支持单行上的事物处理：对储存在一个row key下的数据进行原子性的读、写、更新

允许把数据项用作整数计数器

允许用户在服务器的地址空间内执行脚本程序（Google开发的Sawzall数据处理语言）

Bigtable可以和MapReduce一起使用:Bigtable作为MapReduce框架的输入和输出

# 5.Building Blocks

Bigtable建立在其他几个Google基础构建上

Bigtable使用Google的分布式文件系统（GFS）储存日志文件和数据文件

Bigtable集群通常运行子啊一个共享的机器池中，Bigtable的进程经常和其他应用的进程共享机器

Bigtable内部储存数据的文件是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——Chubby

Bigtable使用Chubby完成以下任务：

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.储存Bigtable数据的自引导指令的位置

3.查找tablet server，以及在tablet server失效时进行善后

4.储存Bigtable的模式信息

5.储存访问控制列表

# 6.Implementation

Bigtable包括三个主要组件：client library、一个master server、多个tablet server

master server负责工作：

为tablet server分配tablets

监测新加入的或者过期失效的tablet server

对tablet server进行负载均衡

对保存在GFS上的文件进行垃圾收集

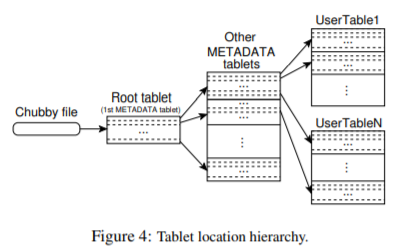
处理对模式的相关修改操作，例如建立table和column family

tablet server负责管理一个tablet的集合。tablet server负责处理它所加载的tablet的读写操作，以及在tables过大时对其进行分割

一个Bigtable集群储存了很多表，每个表包含了一个Tablet的集合，而每个tablet包含了某个范围内的行的所有相关数据

6.1Table Location

使用一个三层的、类似B+树的结构储存Tablet的位置信息，如下图：



root tablet的位置储存在Chubby

root tablet储存METADATA tablet的所有tablet位置

每个METADATA table储存一组user table位置

root table是METADATA table的第一个table，不会分裂

METADATA table的每个row key储存一个tablet的位置

client缓存tablet的位置，过期时从头开始遍历，最多需要读六次

二级信息（log，用于调试和性能分析）也储存在METADATA tablet中

6.2Table Assignment

在任何时刻，一个tablet只能分配给一个tablet server

master server记录了当前有哪些活跃的tablet server、哪些tablet分配给了哪些tablet server、哪些tablet还没有被分配，并分配tablet给有足够空间的tablet server

Bigtable使用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。如果tablet server丢失了Chubby上的独占锁，就会停止对tablet提供服务

master server负责检查一个tablet server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 server在启动时执行以下步骤

1.master server从Chubby获取一个唯一的Master锁，用来阻止创建其他的master server实例

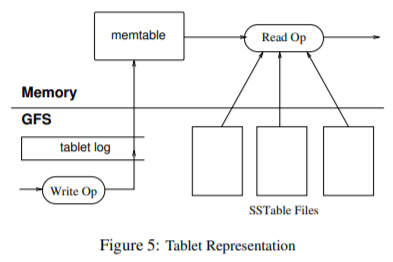
2.master server扫描Chubby的服务器文件锁储存目录，获取当前正在运行的服务器列表

3.master server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 server扫描METADATA表获取所有的tablet的集合

6.3Tablet Serving

示意图如下：



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，这个缓存称为——memtable

较早的更新存放在一系列SSTable中

恢复一个tablet：

tablet server从METADARA表中读取元数据

table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 server把SSTable的索引读进内存，之后通过redo point之后提交的更新来重建memtable

对tablet server进行写操作：

tablet server检查操作格式是否正确、操作发起者的权限是否足够（通过Chubby验证）

成功的修改操作会记录在提交日志里

当一个写操作提交后，写的内容插入到memtable里

对tablet server进行读操作：

tablet server进行完整性和权限检查

一个有效的读操作在一个由一系列SSTable和memtable合并的视图里执行，且可以高效生成合并视图

6.4Compactions

Minor Compaction：当memtable尺寸达到门限值的时候会被冻结，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

Minor Compaction过程目的：

shrink tablet server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

Minor Compaction过程中，正在进行的读写操作仍能继续

Merging Compaction：读取一些SSTable和memtable的内容，合并成一个新的SSTable

Major Compaction：合并所有的SSTable并生成一个新的SSTable

# 7.Refinements

7.1Locality groups

client可以将多个column families组合成一个locality group，对tablet中的每个locality group都会生成一个单独的SSTable。将通常不会一起访问的column families分割成不同的locality groups可以提高读取操作的效率

此外，可以以locality group为单位设定一些有用的调试参数。

tablet server可以将频繁访问的locality group放入内润，提高METADATA表中具有位置相关性的column families的访问速度

7.2Compression

client可以控制一个locality group的SSTable是否需要压缩，如果需要压缩，以什么格式来压缩

client通常使用两遍、可定制的压缩方式：

第一遍采用Bentley and McIlroy‘s方式：这种方式在一个很大的扫描窗口里对常见的长字符串进行压缩

第二遍采用快速压缩算法：即在一个16KB的小扫描窗口中寻找重复数据

这种两遍式的压缩方式压缩和解压的速率很快且在空间上的压缩率也很令人满意

7.3.Caching for read performance

为了提高读操作的性能，tablet server使用二级缓存的策略：

扫描缓存是第一级缓存，主要缓存tablet server通过SSTable接口获取的key-value对

（对于经常要重复读取的相同数据的应用程序来说，扫描缓存非常有效）

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）

7.3.1Bloom filters

通过允许client对特定locality group的SSTable指定Bloom过滤器，以此来减少硬盘访问的次数

可以使用Bloom过滤器查询一个SSTable是否包含特定row和column的数据

7.3.2Commit-log implementation

如果把对每个tablet的操作的commit日志都存在一个单独的文件的话，那么就会产生大量的文件，并且这些文件会并行的写入GFS

设置每个tablet server一个commit日志文件，把修改操作的日志以追加方式写入到用一个日志文件，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

为了避免多次读取日志文件，我们首先把日志文件按照关键字（table，row name，log sequence number）排序。之后只要一次磁盘seek操作、之后顺序读取就可以了

为了并行排序，先将日志分割成64MB的段，之火在不同的tablet server对段进行并行排序

在恢复的时候，table server能够检测出并忽略掉那些由于线程切换而导致的重复的记录

7.3.3Speeding up tablet recovery

master server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

在卸载tablet之前，源tablet server还会再做一次Minor compaction

7.3.4Exploiting immutability

使用Bigtable时，处理SSTable缓存之外的其他部分产生的SSTable都是不变的，可以利用这一点对系统进行简化

当从SStable读取数据的时候，不必对文件系统访问操作进行同步

可以把永久删除被标记为“删除”的数据的问题，转换为对废弃的SSTable进行垃圾收集的问题

SSTable的不变性使得分割Tablet的操作非常快捷

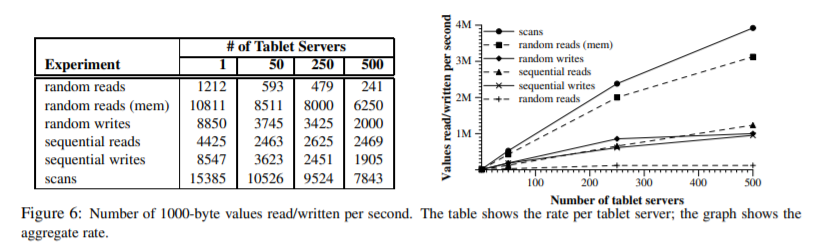
# 8.Performance Evaluation

测试性能和可拓展性，建立包括N台tablet server的Bigtable集群

tablet server，master，test client，GFS运行在同一组机器

测试随机读、从内存随机读、随机写、序列读、序列写、扫描

表现如下图所示：



8.1Single tablet-server performance

随机读的性能比其他操作慢一个数量级

内存中随机读快很多

随机和序列写操作的性能比随机读要好些

序列读的性能好于随机读

扫描的性能更高

8.2Scaling

tablet server从1台增加到500台，吞吐量增长100倍

多台server负载不均衡会造成性能下降

随机读的性能随tablet server增加，提升的幅度最小

# 9.Real Applications

9.1Google Analytics

Google Analytics使用两个表：

Raw click table（200TB），每行一个用户的会话，按时间顺序存储

Summary table（20TB），每个站点各种预定义信息的汇总，周期运行MapReduce任务生成

9.2Google Earth

这个系统使用一个表存储与处理数据，使用另外一个存储用户数据：

存储原始图像的表（70TB），每一行代表一个独立的地理区域

索引GFS数据的表（500GB），每秒几万个查询请求，上百个tablet server，in-memory，column family

9.3Personalized Search

每一个用户id和一个column name绑定，一个单独的column family被用来存储各种类型的行为

# 10.Lessons

发现很多类型的错误会导致大型分布式系统受损：网络中断、很多分布式协议中设想的fail-stop、内存数据损坏、时钟偏差、机器挂起、拓展的和非对称的网络分区、其他系统的Bug、GFS配额溢出、计划内和计划外的硬件维护

系统级的监控对Bigtable非常重要

拓展RPC，详细记录RPC调用的重要操作

每个Bigtable集群都在Chubby中注册，跟踪集群状态、监控大小