Big table

Bigtable是一个稀疏的、分布式的、持久化存储的多维度排序Map 。也就是一个分布式的结构化数据存储系统，它被设计用来处理海量数据.很多方面，Bigtable很像一个数据库：它实现了很多数据库的策略。并行数据库和内存数据库已经实现了可扩展和高性能，但是Bigtable与这些系统相比提供了不同的接口。Bigtable不支持全关系型的数据模型，BigTable为客户端提供了一种简单的数据模型，客户端可以动态地控制数据的布局和格式，并且利用底层数据存储的局部性特征。Bigtable将数据统统看成无意义的字节串，客户端需要将结构化和非结构化数据串行化再存入Bigtable。

1. MAP

Bigtable的键有三维，分别是行键（row key）、列键（column key）和时间戳（timestamp），行键和列键都是字节串，时间戳是64位整型；而值是一个字节串。可以用 **(row:string, column:string, time:int64)→string**来表示一条键值对记录。

### 2.持久化

        Persistence（持久性）只是表示当你的程序结束或数据入口关闭后，你保存在map中的数据会被持久化，这个和其他的持久化存储方式没区别。

### 3. 分布式

       BigTable构建在分布式文件系统上，以便底层文件存储可以在独立机器阵列之间传播。BigTable可以运行在Google File System上。

### 4. 有序

       与大多数Map实现不同，在BigTable中，键值对以严格的字母顺序保存。BigTable中的value是不被排序的，只有key被排序。

### 5.多维度

 Bigtable不是关系型数据库，但是却沿用了很多关系型数据库的术语，像table（表）、row（行）、column（列）等。这容易让读者误入歧途，将其与关系型数据库的概念对应起来，从而难以理解论文。但如果跳脱出关系型数据库的思想，而把BigTable只是想成一个简单的三维表，那么就可以解释得通了。行是表的第一级索引，列是第二级索引，时间戳是第三级索引。

行 Row：行关键字为第一级索引 ,行关键字可以是任意的字符串，但是大小不能超过64KB ;表中数据按照行关键字的字典序进行排序;同一地址域的数据存放在表中连续的位置 ;倒排便于数据压缩，可以大幅提高数据的压缩率 Big table支持行级事务级别，支持row级别操作的原子性 ;同一行可以包含一个或多个列，不同行列的组成可以不同 ;是对于同一域名的内容，我们可以进行更加快速的索引.

列关键字 Family：Qualifier ：列关键字为第二级索引

是具有相同类型的列的集合，有两点优势：

相同类型的列放在一起，可以提高压缩效率

将相似的信息放在一个列族中的不同列，提高了读数据效率

列族是访问权限控制的最小单元

时间戳 Timestamp : 时间戳为第三级索引

每一个Column Key可能关联多次更新，因此，Bigtable使用Timestamp来标识不同的版本

同一个Column Key的多个版本按Timestamp倒序存放，这样查询时总是先读取到最新的版本

每一个Column Family允许用户配置最多保留的版本数量，超出的版本将会被清理掉

### **组成**

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。  
 Master服务器主要负责以下工作：为Tablet服务器分配Tablets、检测新加入的或者过期失效的Table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，它还处理对模 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（通常每个服务器有大约数十个至上千个Tablet）。每个 Tablet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。  
 和很多Single-Master类型的分布式存储系统类似，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。由于BigTable的客户程序不必通过Master服务器来 获取Tablet的位置信息，因此，大多数客户程序甚至完全不需要和Master服务器通信。在实际应用 中，Master服务器的负载是很轻的。  
 一个BigTable集群存储了很多表，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。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 Tablet。

**存储**

Bigtable使用一个类似B+树的数据结构存储片的位置信息。

首层将root tablet的位置信息以文件的形式存储在Chubby中，root tablet包含所有其他metadata

tablet的位置信息

第二层metadata tablet保存着其他所有tablet的位置

第三层即为数据的tablet

****优化****

**1、局部性群族**

\* 用户可将多个列族组合成一个局部性群族，Tablet中每个局部性群族都会生产一个SSTable，将通常不会一起访问的分割成不同局部性群族，可以提高读取操作的效率

\* 此外，可以局部性群族为单位专门设定一些调优参数，如是否存储于内存等

**2、压缩**

\* 用户可以控制一个局部性群族的SSTable是否压缩

\* 很多用户使用”两遍可定制“的压缩方式：第一遍采用Bentley and Mcllroy（大扫描窗口内常见长字符串压缩），第二遍采用快速压缩算法（小扫描窗口内重复数据），这种方式压缩速度达到100~200MB/s，解压速度达到400~1000MB/s，空间压缩比达到10:1

**3、缓存**

\* Tablet服务器使用二级缓存策略来提高读操作性能。两级的缓存针对性不同：

\* 第一级缓存为扫描缓存：缓存Tablet服务器通过SSTable接口获取的Key-Value对（时间局部性）

\* 第二季缓存为块缓存：缓存从GFS读取的SSTable块（空间局部性）

**4、布隆过滤器**

\* 一个读操作必须读取构成Tablet状态的所有SSTable数据，故如果这些SSTable不在内存便需多次访问磁盘

\* 我们允许用户使用一个Bloom过滤器来查询SStable是否包含指定的行和列数据，付出少量Bloom过滤器内存存储代价，换来显著减少访问磁盘次数

**5、Commit日志实现**

\* 如果每个Tablet操作的Commit日志单独写一个文件，会导致日志文件数过多，写入GFS会产生大量的磁盘Seek操作而产生负面影响

\* 优化：设置为每个Tablet服务器写一个公共的日志文件，里面混合了各个Tablet的修改日志。  
\* 这个优化显著提高普通操作性能，却让恢复工作复杂化。当一台Tablet服务器挂了，需要将其上面的tablet均匀恢复到其他Tablet服务器，则其他服务器都得读取完整的Commit日志。为了避免多次读Commit日志，我们将日志按关键字排序(table, row, log\_seq)，让同一个Tablet的操作日志连续存放

**6、Tablet恢复提速**

\* Master转移Tablet时，源Tablet服务器会对这个Tablet做一次Minor Compaction，减少Tablet服务器日志文件没有归并的记录，从而减少了恢复时间

**7、利用不变性**

\* 在使用Bigtable时，除了SSTable缓存外其他部分产生的SSTable都是不变的，可以利用这个不变性对系统简化

总结：

在BigTable中存储结构化数据

BigTable是一个大型的具有容错和自治特性的系统

BigTable不是一个关系型数据库,不支持sql查找，但可以通过键值来查询，并且构建在GFS之上。

BigTable有三个不同类型的服务器master,talbet,Chubby