列族：列族数据库将数据存储在列族中，而列族里的行则把许多列数据与本行的“行键”关联起来。存放在同一列族下的所有数据通常都属于同一个类型（我们可以把同一个列族下的数据压缩在一起）。感觉一个列族对应的就是一张表。

时间戳：数据库中自动生成的唯一二进制数字，与时间和日期无关的，通常用作给表行加版本戳的机制。每次修改或插入包含column列的行时，就会在column列中插入增量数据库时间戳值。使用某一行中的column列可以很容易地确定该行中的任何值自上次读取以后是否发生了更改。如果对行进行了更改，就会更新该时间戳值。如果没有对行进行更改，则该时间戳值将与以前读取该行时的时间戳值一致。这将会在并发控制中有广泛应用。

原子操作：一个操作要不然就做完，要不然就不做，不会做一半就结束。

Tablet：表中的每个行都可以动态分区。每个分区叫做一个”Tablet”，Tablet是数据分布和负载均衡调整的最小单位。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。

数据块：SSTable是一系列的数据块组成的。SSTable使用块索引（通常存储在SSTable的最后）来定位数据块。在打开SSTable的时候，索引被加载到内存，每次查找都可以通过一次磁盘搜索完成；也可以选择把整个SSTable都放在内存中，这样就不必访问硬盘了。

Chubby:首先我认为Chubby是一个解决分布式系统一致性问题的组件。利用这个组件，可以实现一个分布式系统的一致性。Chubby类似于一个提供锁的服务器，里面有很多小的文件。假如有一个系统想要实现一致性，获取一个一致性的值，则就会所有的机器同时向Chubby申请创建一个文件，最终仅会有一个机器创建文件成功，那么这个创建成功的机器的值则将会被选为这个分布系统最终的值，并将这个确定的值分发给所有系统中的其他机器，从而实现系统的一致性。<https://blog.csdn.net/historyasamirror/article/details/3870168> （更多细致性的内容可以再看看这个网站）

SSTable与memtable、REDO：SSTable中是Tablet的已保存更新数据（存储在GFS中），memtable中存储的是Tablet较新的数据。当要恢复Tablet时，通过SSTable的数据以及其中的REDO Point，来一步步的重建Tablet的最新数据更新表memtable，从而获得Tablet出问题时到底存有哪些数据。

Bigtable是一个分布式的结构化数据存储系统，能够实现适用性广泛、可扩展、高性能和高可用性的目标。Bigtable面对不同产品的迥异需求，Bigtable成功的提供了一个灵活的、高性能的解决方案。

Bigtable中的数据没有Schema，用户自己去定义Schema。

**数据模型**：

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

(row:string, column:string,time:int64)->string

数据的下标是行和列的名字，名字可以是任意的字符串。Bigtable将存储的数据都视为字符串，但是Bigtable本身不去解析这些字符串，客户程序通常会在把各种结构化或者半结构化的数据序列化到这些字符串里。

**行：**

表中的行关键字可以是任意的字符串（目前支持最大64KB的字符串）。对同一个行关键字的读或者写操作都是原子的（不管读或者写这一行里多少个不同列）（这是通过支持单行上的事务处理实现的）。Bigtable目前还不支持通用的跨行事务处理。

Bigtable通过行关键字的字典顺序来组织数据。

用户可以通过选择合适的行关键字，在数据访问时有效利用数据的位置相关性，从而更好的利用这个特性。举例来说，在Webtable里，通过反转URL中主机名的方式，可以把同一个域名下的网页聚集起来组织成连续的行。具体来说，我们可以把maps.google.com/index.html的数据存放在关键字com.google.maps/index.html下。把相同的域中的网页存储在连续的区域可以让基于主机和域名的分析更加有效。相当于是大类放在一起，然后大类里面不同的小类放在一起；而不是小类放在一起，然后小类与小类之间并没有什么大的共同特性。

**列族：**

列关键字的命名语法如下：列族：限定词。

列族的名字必须是可打印的字符串，而限定词的名字可以是任意的字符串。

**时间戳：**

数据项中，不同版本的数据按照时间戳倒序排序，即最新的数据排在最前面。

**API:**

可以用现有的提供的API对Bigtable进行一些基本的新建、删除以及修改更新操作等。用Scanner可以扫描Bigtable并通过ScannerStream对获取的列族等进行扫描。

**BigTable构件：**

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内部存储数据的文件是Google SSTable格式的。

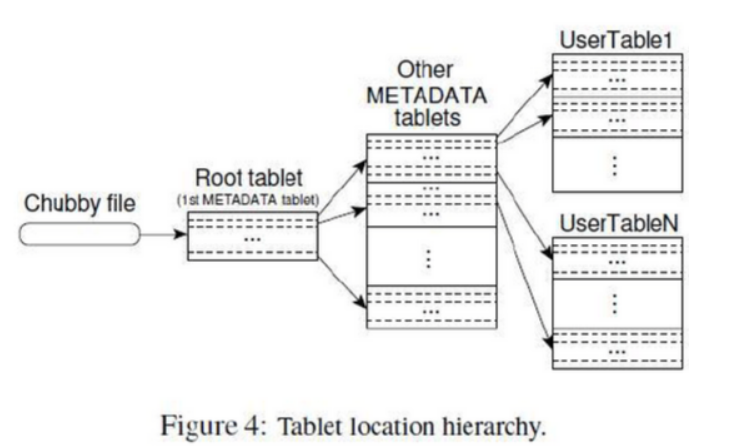
BigTable还依赖一个高可用的、序列化的分布式锁服务组件，叫做Chubby。

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。

Master服务器用来管理Tablet服务器，同时可以对模式进行相关的修改操作。

Tablet服务器都管理一个Tablet的集合，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。客户程序可以直接和Tablet服务器通信进行读写操作，以避免Master服务器负载过大。

**Tablet的位置：**



Chubby file存储Root tablet，Root tablet是METADATA表中的第一个tablet，剩余的tablet用来存储User tablet集合的位置信息。

**Tablet的分配：**

一个Tablet只能分配给一个Tablet服务器。

Master服务器负责管理Tablet以及Tablet服务器。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。

每个Tablet服务器会在Chubby上申请一个文件锁，Master服务器会监听Chubby中所有Tablet服务器的文件信息，从而得知Tablet服务器是否还持有锁，Tablet服务器也能够从此感知我是否还持有锁。

Master服务器通过轮询Tablet服务器文件锁的状态来检测何时Tablet服务器不再为Tablet提供服务，并且尽快将不再提供服务的Tablet服务器中的Tablet尽快分配给其他Tablet服务器。

**Tablet的服务：**

Tablet的持久化信息保存在GFS上，更新操作提交到REDO日志中。较新的数据存储在memtable中，较早的数据存储在SSTable中。

当要恢复一个Tablet时，首先读取Tablet的元数据，包括：构成Tablet的SSTable列表，以及REDO Point。

**Compactions：**

Minor Compaction ：Memtable用来保存Tablet的最新更新数据，当memtable的大小到达一个确定值的时候，会将其转化为SSTable，并写入GFS。

Major Compaction：合并所有的SSTable并生成一个新的SSTable的Merging Compaction过程叫作Major Compaction。（实际是回收资源）

**优化：**

**局部性群组：**

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。

**压缩：**

客户程序可以控制一个局部性群组的SSTable进行压缩。分块压缩可以在只读取SSTable的一小部分数据的时候就不必解压整个文件。

很多客户程序使用了“两遍”的、可定制的压缩方式。这种方式在速度以及空间上的压缩效率均是非常高的。

**通过缓存提高读操作的性能：**

**Bloom过滤器：**

允许客户程序对特定局部性群组的SSTable指定Bloom过滤器，来减少读操作访问硬盘的次数。

**Commit日志的实现：**

设置每个Tablet服务器一个Commit日志文件，把修改操作的日志以追加方式写入同一个日志文件，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。

使用单个日志显著提高了普通操作的性能，但是将恢复的工作复杂化了。

减少磁盘seek次数以及实现服务器并行恢复Tablet：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。排序之后，对同一个Tablet的修改操作的日志记录就连续存放在了一起，因此，我们只要一次磁盘Seek操作、之后顺序读取就可以了。为了并行排序，我们先将日志分割成64MB的段，之后在不同的Tablet服务器对段进行并行排序。这个排序工作由Master服务器来协同处理，并且在一个Tablet服务器表明自己需要从Commit日志文件恢复Tablet时开始执行。

**Tablet恢复提速：**

通过两次Minor Compaction提速。第一次将memtable转化为SSTable，从而减少未归并的数据；源Tablet服务器还会再做一次（通常会很快）Minor Compaction，以消除前面在一次压缩过程中又产生的未归并的记录。

**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。这不仅使得分割Tablet的操作非常快速，而且不必为每个分割出来的Tablet建立新的SSTable集合，而是共享原来的Tablet的SSTable集合。

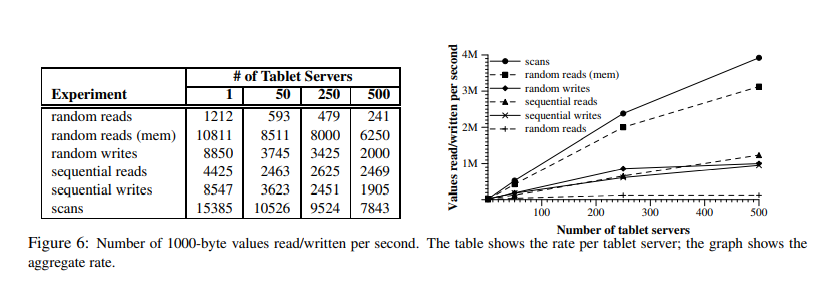
**性能评估：**

**单个Tablet服务器的性能：**

随机读的性能比其它操作慢一个数量级或以上，因为需要从GFS读取64KB的Block，如果在本地存储的话速度会快很多。

随机和序列写操作的性能比随机读要好些，因为通过把写入操作的内容追加到一个Commit日志文件的尾部，并且可以采用批量提交的方式。

**性能提升：**



当将系统中的Tablet服务器进行增加后，系统的整体吞吐量有显著增加。

个人感悟：

对于这样一个大型系统来说，感觉性能是一个方面，其中对容错的处理机制是更能体现这个系统的性能指标。这样一个大型系统，任何一个方面出错而导致微小或者短暂的系统问题都将会产生巨大的影响。这样的系统又不可能进行完整的测试从而保证这个系统就是完美无瑕的，因此在现有的基础上能够完善的运行，同时有多种容错机制已经能够从很大的程度上体现了这个系统的成功。