SSTable 是 Google 为 Bigtable 设计的内部数据存储格式。所有 SSTable 文件都是存储在 GFS 上的，用户可以通过键来查询相应的值。SSTable 中的数据被划分成一个个的块（block），每个块的大小是可以设置的，一般来说设置为 64KB。在 SSTable 的结尾有一个索引（Index），这个索引保存了 SSTable 中块的位置信息，在 SSTable 打开时这个索引会被加载进内存，这样用户在查找某个块时首先在内存中查找块的位置信息，然后在硬盘上直接找到这个块，这种查找方法速度非常快。由于每个 SSTable 一般都不是很大，用户还可以选择将其整体加载进内存，这样查找起来会更快。

从概念上来讲子表是表中一系列行的集合。每个子表都是由多个 SSTable 以及日志（log）文件构成的。有一点需要注意，那就是不同子表的 SSTable 可以共享，也就是说某些 SSTable 会参与多个子表的构成，而由子表构成的表则不存在子表重叠的现象。

Bigtable 中的日志文件是一种共享日志，也就是说系统并不是对子表服务器上每个子表都单独地建立一个日志文件，每个子表服务器上仅保存一个日志文件，某个子表日志只是这个共享日志的一个片段。这样会节省大量的空间，但在恢复时却有一定的难度，因为不同的子表可能会被分配到不同的子表服务器上，一般情况下每个子表服务器都需要读取整个共享日志来获取其对应的子表日志。Google 为了避免这种情况出现，对日志做了一些改进。Bigtable 规定将日志的内容按照键值进行排序，这样不同的子表服务器都可以连续读取日志文件了。一般来说每个子表的大小在 100MB 到 200MB 之间。每个子表服务器上保存的子表数量可以从几十到上千不等，通常情况下是 100 个左右。

为了减少访问开销，提高客户访问效率，Bigtable 使用了缓存（Cache）和预取（prefetch）技术，这两种技术手段在体系结构设计中是很常用的。子表的地址信息被缓存在客户端，客户在寻址时直接根据缓存信息进行查找。

在数据的存储方面 Bigtable 做出了一个非常重要的选择，那就是将数据存储划分成两块。较新的数据存储在内存中一个称为内存表（memtable）的有序缓冲里，较早的数据则以 SSTable 格式保存在 GFS 中。这种技术在数据库中不是很常用，但 Google 还是做出了这种选择，实际运行的效果也证明 Google 的选择虽然大胆却是正确的。在数据存储中还有一个重要问题，就是数据压缩的问题。内存表的空间毕竟是很有限的，当其容量达到一个阈值时，旧的内存表就会被停止使用并压缩成 SSTable 格式的文件。在 Bigtable 中有三种形式的数据压缩，分别是次压缩（Minor Compaction）、合并压缩（Merging Compaction）和主压缩（Major Compaction）。三者之间的关系如图 1 所示。

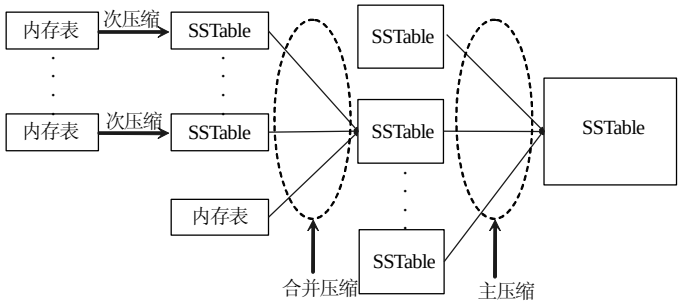

图 1　三种形式压缩之间的关系

每一次旧的内存表停止使用时都会进行一个次压缩操作，这会产生一个 SSTable。但如果系统中只有这种压缩的话，SSTable 的数量就会无限制地增加下去。由于读操作要使用 SSTable，数量过多的 SSTable 显然会影响读的速度。而在 Bigtable 中，读操作实际上比写操作更重要，因此 Bigtable 会定期地执行一次合并压缩的操作，将一些已有的 SSTable 和现有的内存表一并进行一次压缩。主压缩其实是合并压缩的一种，只不过它将所有的 SSTable 一次性压缩成一个大的 SSTable 文件。主压缩也是定期执行的，执行一次主压缩之后可以保证将所有的被压缩数据彻底删除，如此一来，既回收了空间又能保证敏感数据的安全性（因为这些敏感数据被彻底删除了）。