**bigtable-osdi06读后感**

18301030于沛尧

# 前言

BigTable作为谷歌的“三驾马车”之一，其地位之重与意义之大不言而喻。BigTable发展于2004年，在当时，飞速发展的互联网带来了数据的急剧膨胀，谷歌面临着以下问题：需要存储的数据种类繁多；海量的服务请求；商用数据库无法满足需求。

于是，为了顺应时代发展，谷歌的BigTable的主要方向是：广泛的适用性，满足一系列Google产品的存储要求；很强的可扩展性，随时可以加入或撤销服务器；高可用性，确保几乎所有的情况下系统都可用；简单性，减少系统出错的概率，并为上层应用的开发带来便利。

能力有限，我将分析一下SSTable & memtable架构的运作方式与意义。

# SSTable & memtable架构运作

SSTable是BigTable的物理最小存储单元（逻辑单元为子表，单个服务器上多个子表共用一个日志），存储在谷歌的文件系统GFS中。一个SSTable由多个64KB（可配置）大小的块和一个索引构成。数据就保存在块中。SSTable最主要的特点是不可变。也就是说，SSTable的值不会因写操作而改变。

当服务器收到一个合法的写请求，并将写请求提交到日志后，会将数据追加写入内存中的memtable。memtable相当于SSTable的缓存。追加写入的速度远高于在SSTable中随机读写。当memtable成长到一定规模会被冻结，BigTable随之创建一个新的memtable，并且将冻结的memtable转换为SSTable格式写入GFS，这个操作称为minor compaction。这样，写速度就得到了较大的提升。

不过，由于SSTable不会即时更改，读操作又该怎么办呢？当服务器收到读操作，它会在两个Table上进行合并查找，因为memtable和SSTable中对于键值的存储都是字典顺序的，所以整个读操作的执行会非常快。

至此，BigTable会在后台周期性地进行Major Compaction，将memtable中的数据和一部分的SSTable作为输入，将其中的键值进行归并排序，生成新的SSTable并移除原有的memtable和SSTable，新生成的SSTable中包含前两者的全部数据和信息，并且将其中一部分标记未删除的信息彻底清除。

# SSTable & memtable架构的意义

基于上述结构，BigTable对读敏感和写敏感的应用都提供了很好的性能。在高并发的实际应用中，基于行锁的数据库系统可能由于读写的排斥性而使性能受限于写速度，而BigTable的memtable支持同时被读操作和写操作访问。这是BigTable为了满足海量的服务请求而做出的创造。另一个这种架构更适合庞大查询的理由是（我认为），当查询量较大时，memtable处于较高的活跃状态，这种情况的合并操作，相较于直接使用行锁才更有意义。