需求：

1.公司很多地方把SSB作为持久化的存储设备，同时SSDs是Rocksdb的主要存储介质。

2.公司在各个数据中心内用大量的不同配置的的硬件， 并且在大多数简易节点上，配备了1-2个SSD。

3.要存储大量的数据。

4.在多数的使用场景下，读写比例大概是2:1，并且大范围的使用基于内存的缓存

解决：

3.1：Dynamic level size adaptation

如果每个level的大小是固定的，在实际中，最后一级level所存储的数据的大小很有可能并不是之前level的10倍， 在极端情况下，甚至最后一级level所储存的数据只是稍微比之前level大一点，这种情况下，写放大甚至会超过2,

如果我们能动态调整每一级的level的大小使其只有下一级level的1/10， 那么整体的写放大就会被限制在1.11111。

在Rocksdb中，level size multiplier 是可配置的参数，默认是10， multiplier越大， 空间放大和读放大就越小，但是写放大也会越大。因此，这里有一个tradeoff, 也就是空间放大、读放大、写放大三者不可得兼，一般情况下， Facebook 设置的multiplier是10， 某些实例设置为8。

其实还有个好玩的问题：是不是应该把level 和 level +1 直接的multiplier设置为一样，原始LSM-tree论文中证明了设置为相同的值最有利于减少写放大。 当然这是个开放性的问题，是否同样的设置也有利于减少空间放大，特别是当各个level都使用了各自的压缩方法，导致各个level有不同的压缩比率，这个问题大家可以继续思考下。

3.2 各种压缩策略

Key prefix encoding: 就是前缀压缩，restart point。只存储不一样的部分。（相邻key 共享 prefix）

Sequence ID garbage collection: 回收Sequence ID, 如果一个key的sequence ID小于最旧snapshot的Sequence,那就删除掉这些key的Sequence id（即使只有7 byte，感觉就是在极限压榨）. 这是由于这些Sequence ID不在使用了（指向他们的snapshot已经被删除了）， 这个优化方法在某些只有key没有value的场景特别有用。

data compression: rocksdb当前支持多个压缩方法，包括LZ、 Snappy、zlib以及标准的， 每一级level可以配置为这些中的任意一个。 压缩的对象是block(根据实验结果，强压缩可以压缩到25%， 弱压缩可以到压缩到40%)， 为了较少解压缩的频度， Block cache 缓存的是非压缩的数据（注意最近被访问过的压缩的file block是以压缩的方式被缓

存在page cache中， 这样压缩的sst file 就只占用更少的space, 这样page cache能缓存更多的数据）。

dictionary-base compression 数据词典可以用来提供更好的压缩， 这种压缩特别适合于比较的data block（正常情况下， 如果data block比较下，压缩率比较小），dictionary-base能够在data block比较小的时候， 依然得到好的压缩率。 在实现的时候， LSM-tree这种结构能够很容易的产生并维护该dictionary, 一个在所有data block产生的dictionary能够储存在文件中，因此当一个文件被删除时，该dictionary也就被删除了。

Tiered compression 压缩可以减少对存储空间的需求， 但是会增加CPU的负载，这是由于数据需要频繁的解压缩。 一般说来，压缩率越高， CPU的负载越高， 在我们实际中，last level使用的压缩率最高的算法，有以下几个原因： 虽然last level 包含了大部分的数据（%90），但只有少量的读写会命中这样，last level使用高压缩的数据，可以极大的节省存储空间。通常情况下， level0~level2不使用任何压缩，这样可以有更小的read延迟（虽然然增加了空间 / 写放大， 但是由于数据小， 可以忽视）， level3到last-level之前， 使用低压缩率压缩算法， 这个时候CPU的负载也可以接受， 由于L0~L2之间的数据被频繁访问， 因此这些file cache被缓存在page cache中，L3及其以上，读这些level中数据必须进行解压缩，

Bloom filter 虽然会有一定的CPU开销， 布隆过滤器可以有效减少IO。同时， 会增加了一定的内存开销，这是由于每一个key需要占用7bit. 在某些场景下， 我们并不给last level设置布隆过滤器， 虽然这会增加对last level的访问， 但在实际情况中，直接穿透到last level的读请求是极其少量的，而且，last level的过滤器所占的空间比较大， 过滤器本身所占用的内存会大量的其它的cache踢出cache。 因此在实际中， 我们根据经验，我们在last level不设置bloom filter.这其实就是种权衡。

Prefix Bloom filters bloom filter 并不对range query有所帮助， 因此我们开发了基于前缀的bloom filter。 这种优化在range query的时候可以极大减少读放大。