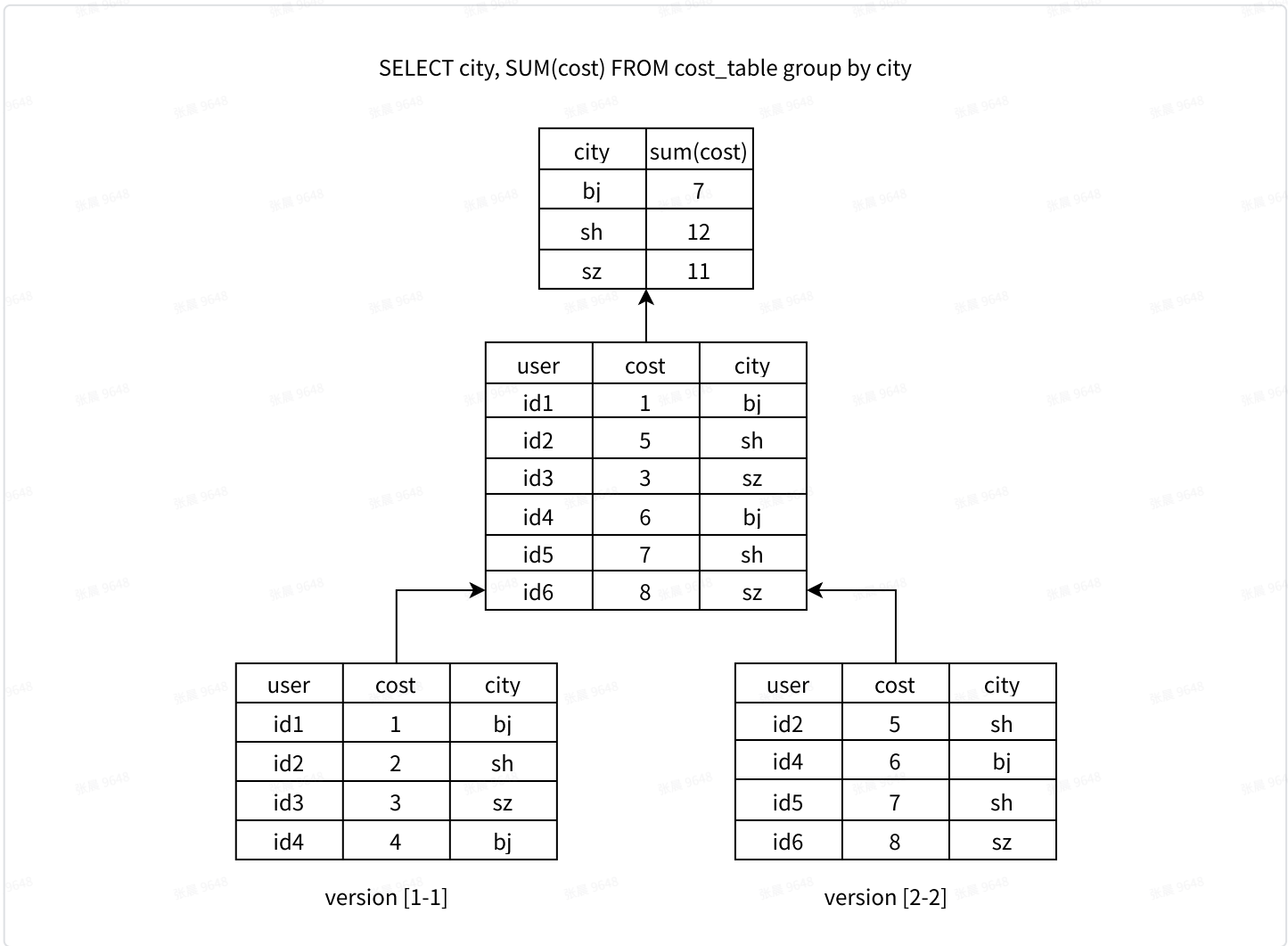
Unique-Key的主键模型支持

动机

Doris的Unique-Key模型被广泛使用在Flink-CDC同步、用户画像、电商订单等有更新需求的场景中，但Unique-Key目前使用的是Merge-On-Read的实现方式，由于以下几个原因，查询性能较差：

1. 由于是merge on read，所以在查询时需要把所有筛选过的数据读取出来，进行一次归并排序，才能确认哪行数据是最新的数据。这种归并排序带来了额外的cpu-cost
2. 由于无法确认某个segment中某一行的数据是否是最新的数据，因此UniqueKey不支持聚合函数谓词下推，加大了数据的读取量

下图是一个简化后的Unque Key模型执行过程表示：



相关调研

我们调研了其他主流数仓对于数据更新模型的实现方案，除了UniqueKey使用的Merge-On-Read以外，基本有以下几类

Merge-On-Write（Mark Delete and Insert)

- 方案概述：
  - 通过一个主键索引检查是否是更新请求，并定位到被覆盖的key是在哪个文件的哪一行
  - 通过一个bitmap记录标记删除key的rowid，读取时基于bitmap中记录的rowid过滤
- 优点：读取时避免了key的比较，同时支持各种谓词下推以及二级索引
- 缺点：写入时会有额外的查询索引和更新bitmap的代价
- 代表系统：Hologres, Aliyun ADB, ByteHouse

Copy-On-Write

- 方案概述：
  - 在写入/更新数据时，直接同步合并原文件，生成新版本的文件，即使只有一个字节的新数据被提交，也需要重写整个列数据文件。
- 优点：查询效率最高
- 缺点：写入数据的成本很高，适合低频大批量更新数据，高频读取的场景
- 代表系统：Delta Lake, Hudi

Delta-Store

- 方案概述：
  - 将数据区分为base data和delta data。每个key在base data中是唯一的
  - 在写入/更新数据时，先查询base data，如果有则是一条更新请求，将更新后的数据写入到delta store中，读取时将delta store中的数据与base data合并得到最新的数据；如果再base data中查询不到，则写入memtable，flush到base data中
- 优点：写入性能不错，能够高效的支持部分列更新
- 缺点：对二级索引支持的不好
- 代表系统：Kudu

方案选型

综合Doris的Unique Key使用场景（对导入的实时性要求高，导入频率高，希望查询尽可能的快），Merge-On-Write是比较适合的优化方案。

详细设计

关键设计点

想要实现Mark Delete and Insert方案，需要考虑以下几点：

1. 增加一个主键索引，能够在导入的时候快速的检查key是否存在，以及所在的文件位置和行号
2. 增加一个delete bitmap，在发生更新的时候，将旧的key标记为已删除的状态

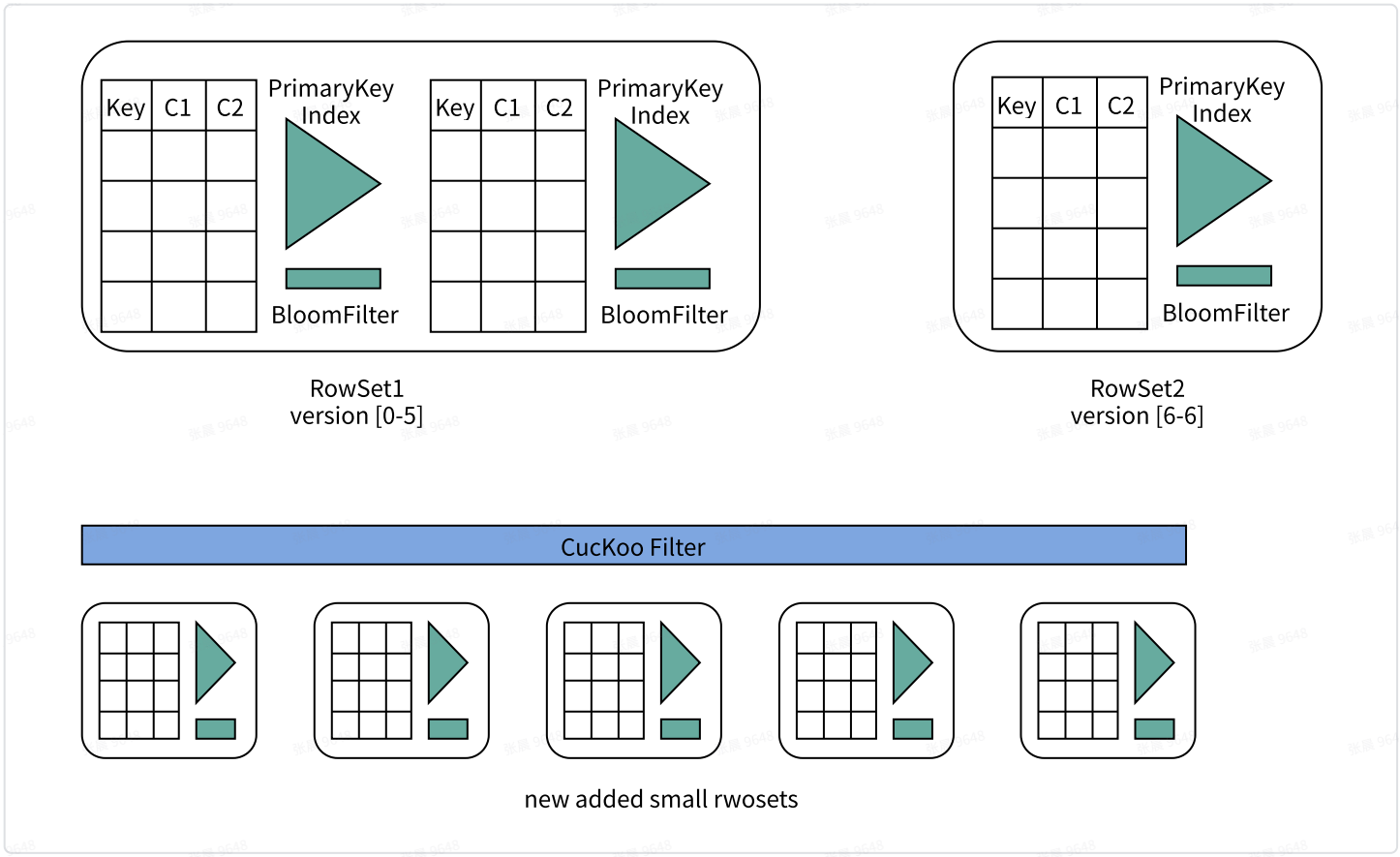
3. delete bitmap的更新需要考虑事务的支持和多版本支持
4. 查询适配delete bitmap
5. Compaction与数据导入的冲突处理。Compaction可能在读取了某个rowset的数据后，尚未提交之前，该rowset的某一行被正在进行的导入任务给覆盖掉了。需要在新生成的rowset中将该行重新标记删除

主键索引

主键索引设计和选型的几个关键点：

1. 高QPS（单个BE目标能支持10w的QPS）
   a. Doris使用列式存储，当主键列包含多列时，一次点查需要逐列读取每列的值，涉及多次磁盘IO。为了提高主键索引的性能，我们需要在主键索引中存储编码后的主键，当主键列包含多列时，这些列会编码到一起存储到主键索引中。
2. 不过度消耗内内存
   a. 主键索引选择使用类似于RocksDB Partitioned Index的方案[参考 https://github.com/facebook/rocksdb/wiki/Partitioned-Index-Filters]，在MemTable flush的时候，在Segment中创建。选用该方案的主要考虑如下：
   b. RocksDB已经证明了它的索引方案可以支撑非常高的QPS，配合BlockCache的合理配置，在保证index page 90+%的Cache 命中率的情况下，查询性能是可以保证的
   c. 由于是基于文件的索引，可以按需加载，对内存的消耗可控
   d. 该方案对主键数据进行了前缀压缩，可以进一步节省空间，减少内存的cache的占用以及磁盘IO的数据量
3. 高频导入下过多小文件对点查性能的影响
   a. 在默认配置下，一个tablet可能会有500个rowset，逐个去进行seek的性能可能会比较差
   b. 针对这种情况，每个tablet可以考虑维护一个额外的CuckooFilter用于小文件点查加速，CuckooFilter相比BloomFilter，具备FP rate低（同等大小），可删除，可存储value（如存储行号）等优势。(该方案受SlimDB启发：https://www.vldb.org/pvldb/vol10/p2037-ren.pdf，使用类似于SlimDB中的Multi Cuckoo Filter的方案能够用较低的内存消耗和IO消耗快速确定一条key是否存在）

对于主键索引，还要添加对应的BloomFilter。整体的设计如下图所示



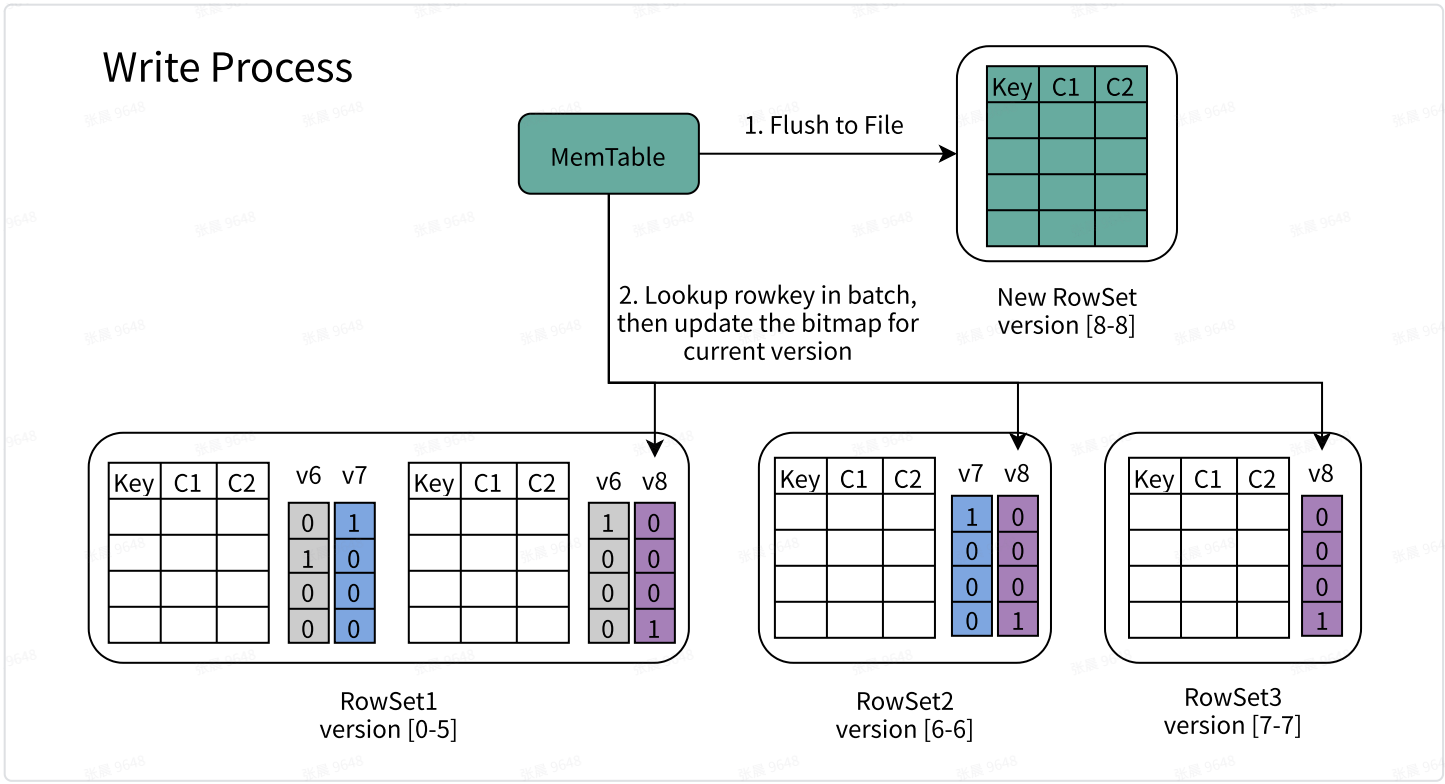
Delete Bitmap

Delete Bitmap使用Roaring Bitmap进行记录，和tablet meta一起存储在RocksDB中，每个segment对应一个bitmap。为了保持Unique Key原有的version语义，Delete Bitmap也需要支持多版本。**考虑到每次导入产生一个版本的bitmap，可能会造成比较大的内存消耗，因此每个版本都产生一个增量修改比较合适。**

写入流程

Delete Bitmap的写入流程如下图所示：

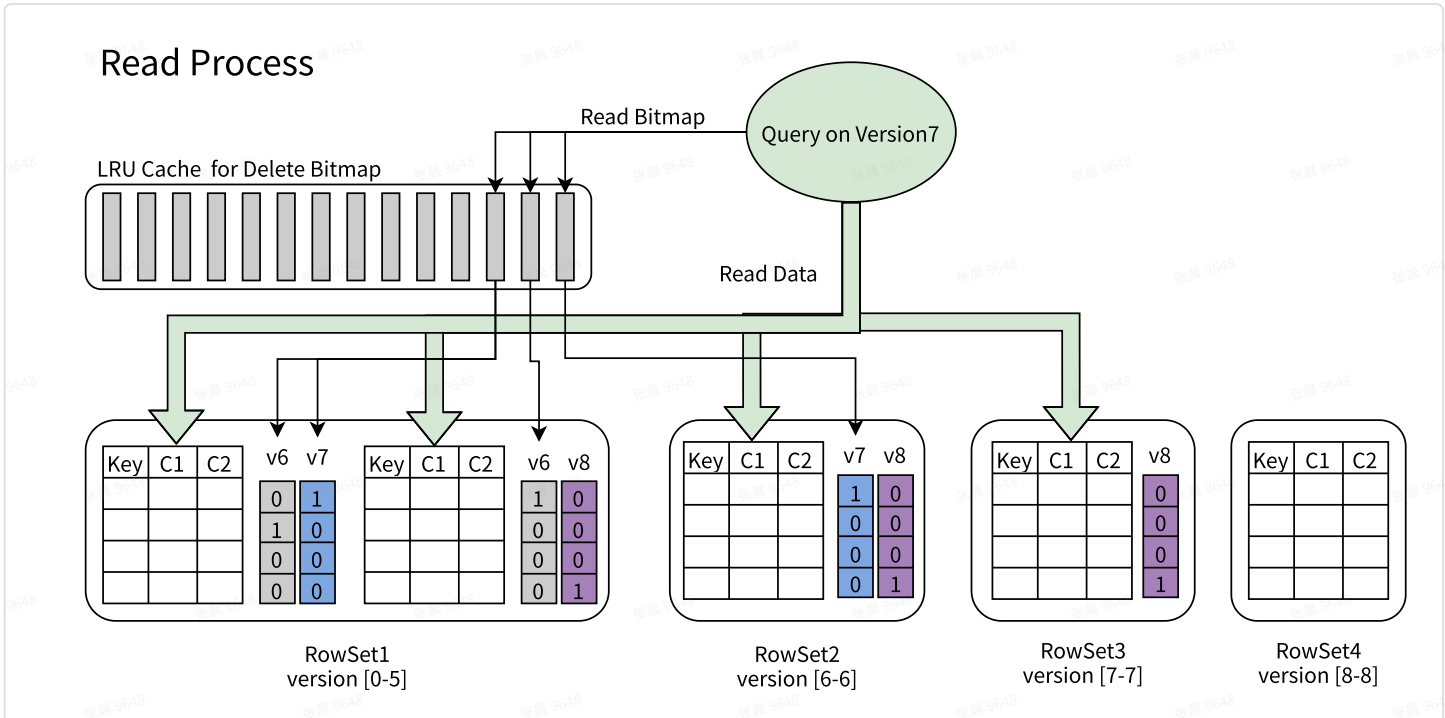
1. DeltaWriter会先将数据flush到磁盘
2. 在publish阶段去批量的点查所有的key，并且更新被覆盖的key对应的bitmap。在下图中，新写入的rowset版本是8，它修改了3个rowset中的数据，因此会产生3个bitmap的修改记录
3. 在publish阶段去更新bitmap，保证了批量点查key和更新bitmap期间不会有新增可见的rowset，保证了bitmap更新的正确性
4. 如果某个segment没有被修改,则不会有对应版本的bitmap记录，比如rowset1的segment1就没有version 8对应的bitmap



读取流程

Bitmap的读取流程如下图所示：

1. 一个请求了版本7的Query，只会看到版本7对应的数据
2. 读取rowset5的数据时，会将v6和v7对它的修改产生的bitmap合并在一起，得到version7对应的完整DeleteBitmap，用来过滤数据
3. 在下图的示例中,版本8的导入覆盖了rowset1的segment2了一条数据，但请求版本7的Query仍然能读到该条数据
4. 在高频导入场景下，可能会存在大量版本的bitmap，合并这些bitmap本身可能也有较大的cpu计算消耗，因此我们引入了一个LRU cache，每个版本的bitmap只需要做一次合并操作。



事务支持

DeltaWriter在写入数据阶段不去进行点查和更新bitmap，而是在publish阶段再去做

查询层适配

Scan适配

1. TabletReader根据Query指定的版本在TabletMeta中获取指定版本的DeleteBitmap
2. 通过 `RowsetReaderContext` 将指定rowset的DeleteBitmap传递给 `BetaRowsetReader`
3. 再通过 `StorageReadOptions` 将Segment对应的DeleteBitmap传递给 `SegmentIterator`
4. SegmentIterator本身就维护了一个 `_row_bitmap` 用于快速的做一些条件过滤，用它减掉传入的DeleteBitmap即可

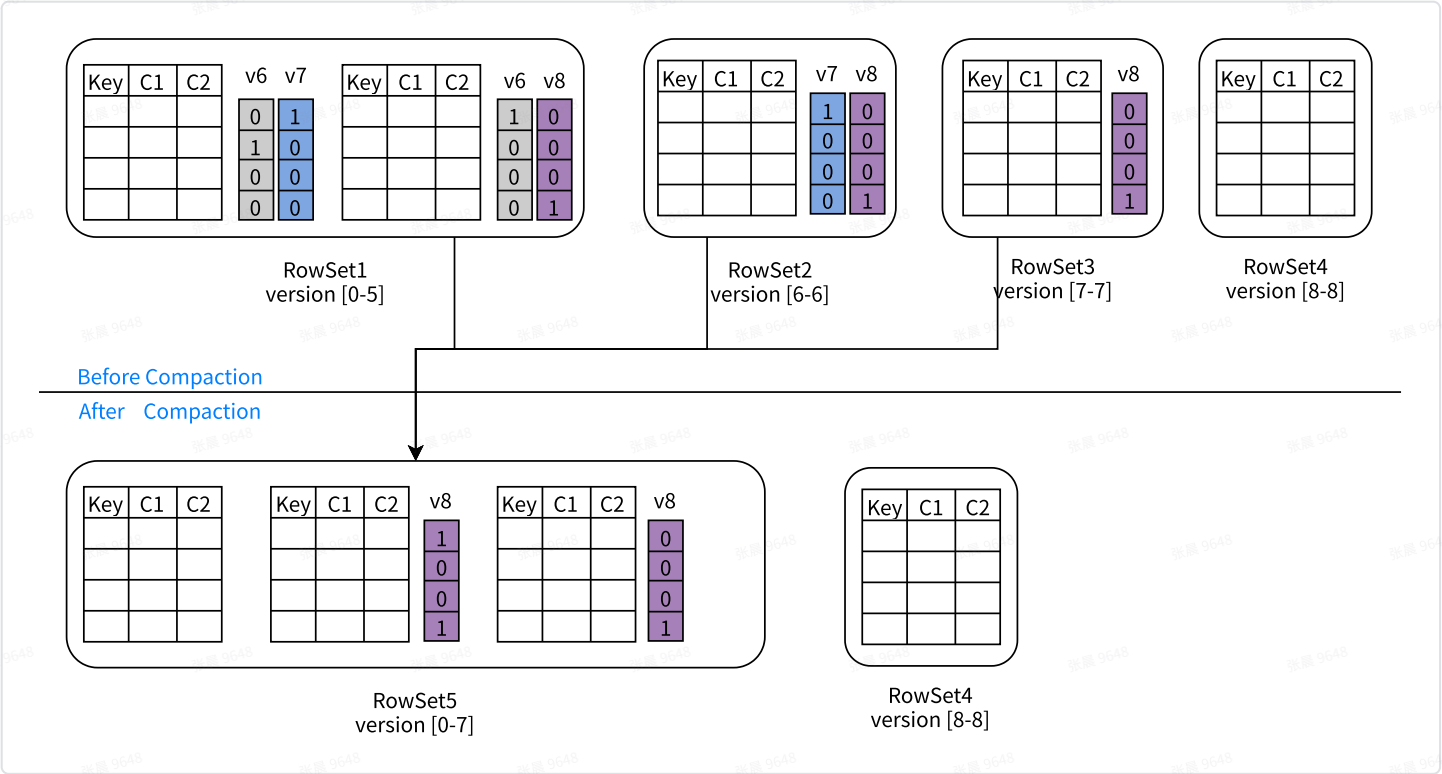
点查适配

1. 目前Doris的数据读取都是针对Scan设计的，没有对点查能力的支持，因此需要增加以下的接口
2. 为每个Segment的meta增加minkey/maxkey的记录，并在内存中通过线段树结构将所有的segment组织起来
3. 通过线段树能够快速的定位到某个encoded_key可能会在哪个segment中
4. 给SegmentIterator增加一个lookup_row_key接口，用于点查

Compaction与写冲突的处理

1. Compaction正常流程
   a. compaction在读取数据时，获取当前处理的rowset的版本Vx，会自动通过delete bitmap过滤掉被标记删除的行（见前面的查询层适配部分）
   b. compaction结束后即可清理源rowset上所有小于等于版本Vx的DeleteBitmap
2. Compaction与写冲突的处理
   a. 在compaction的执行过程中，可能会有新的导入任务提交，假设对应版本为Vy。如果Vy对应的写入有对compaction源rowset中的修改，则会更新到该rowset的DeleteBitmap的Vy中
   b. 在compaction结束后，检查该rowset上所有大于Vx的DeleteBitmap，将它们中的行号更新为新生成的rowset中的segment行号

如下图所示，compaction选择了[0-5], [6-6], [7-7]三个rowset， 在compaction过程中，version8的导入执行成功，在compaction commit阶段，需要处理由version8的数据导入所生成的新bitmap



计划

引用

1. https://github.com/facebook/rocksdb/wiki/Partitioned-Index-Filters
2. https://www.vldb.org/pvldb/vol10/p2037-ren.pdf