Doris存储层设计介绍3——读取流程、Compaction分析

1、整体介绍

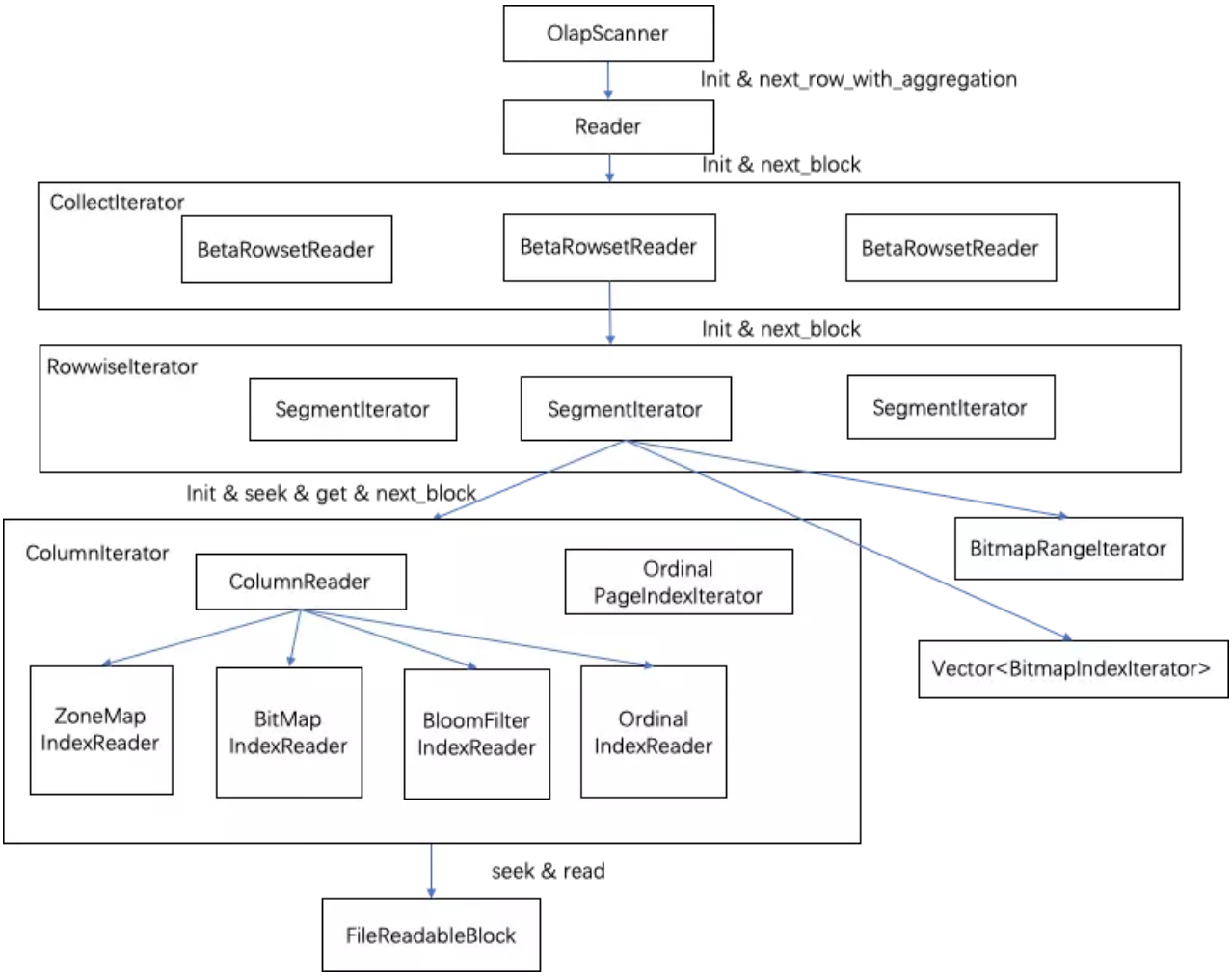
Doris是基于MPP架构的交互式SQL数据仓库，主要用于解决了近实时的报表和多维分析。Doris高效的导入、查询离不开其存储结构精巧的设计。本文主要通过阅读Doris BE模块代码，详细分析了Doris BE模块存储层的实现原理，阐述和解密Doris高效的写入、查询能力背后的核心技术。其中包括Doris列存的设计、索引设计、数据读写流程、Compaction流程等功能。这里会通过三篇文章来逐步进行介绍，分别为《Doris存储层设计介绍1——存储结构设计解析》、《Doris存储层设计介绍2——写入流程、删除流程分析》、《Doris存储层设计介绍3——读取、Compaction流程分析》。

本文为第三篇《Doris存储层设计介绍3——读取、Compaction流程分析》，文章详细介绍了Doris存储层的读取数据、Compaction流程的实现。

2、读取流程

2.1 整体读取流程

读取流程为写入的逆过程，但读取流程相对复杂些，主要因为进行大量的读取优化。整个读取流程分为两个阶段，一个是init流程，一个是获取next_block数据块的过程。具体过程如下图所示：

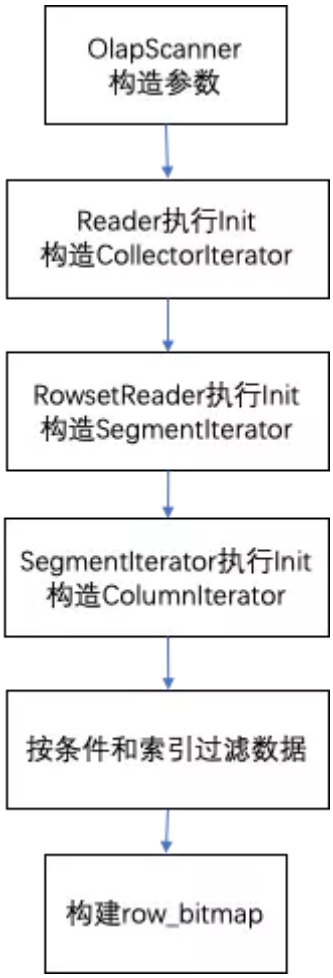


层级关系如下：

1. OlapScanner对一个tablet数据读取操作整体的封装。
2. Reader对读取的参数进行处理，并提供了按三种不同模型读取的差异化处理。
3. CollectIterator包含了tablet中多个RowsetReader，这些RowsetReader有版本顺序，CollectIterator将这些RowsetReader归并Merge成统一的Iterator功能，提供了归并的比较器。
4. RowsetReader则负责了对一个Rowset的读取。
5. RowwiseIterator提供了一个Rowset中所有Segment的统一访问的Iterator功能。这里的归并策略可以根据数据排序的情况采用Merge或Union。
6. SegmentIterator对应了一个Segment的数据读取，Segment的读取会根据查询条件与索引进行计算找到读取的对应行号信息，seek到对应的page，对数据进行读取。其中，经过过滤条件后会对可访问的行信息生成bitmap来记录，BitmapRangeIterator为单独实现的可以按照范围访问这个bitmap的迭代器。
7. ColumnIterator提供了对列的相关数据和索引统一访问的迭代器。ColumnReader、各个IndexReader等对应了具体的数据和索引信息的读取。

2.2 读取init阶段的主要流程

init阶段的执行流程如下：



2.2.1 OlapScanner查询参数构造

1. 根据查询指定的version版本查找出需要读取的RowsetReader（依赖于版本管理的rowset_graph版本路径图，取得查询version范围的最短路径）。
2. 设置查询信息，包括_tablet、读取类型reader_type=READER_QUERY、是否进行聚合、_version（从0到指定版本）。
3. 设置查询条件信息，包括filter过滤字段、is_nulls字段。
4. 设置返回列信息。
5. 设置查询的key_ranges范围（key的范围数组，可以通过short key index进行过滤）。
6. 初始化Reader对象。

2.2.2 Reader的Init流程

1. 初始化conditions查询条件对象。
2. 初始化bloomFilter列集合（eq、in条件，添加了bloomFilter的列）。
3. 初始化delete_handler。包括了tablet中存在的所有删除信息，其中包括了版本和对应的删除条件数组。
4. 初始化传递给下层要读取返回的列，包括了返回值和条件对象中的列。
5. 初始化key_ranges的start key、end key对应的RowCusor行游标对象等。
6. 构建的信息设置RowsetReader、CollectIterator。Rowset对象进行初始化，将RowsetReader加入到CollectIterator中。

7. 调用CollectIterator获取当前行（这里其实为第一行），这里开启读取流程，第一次读取。

2.2.3 RowsetReader的Init流程

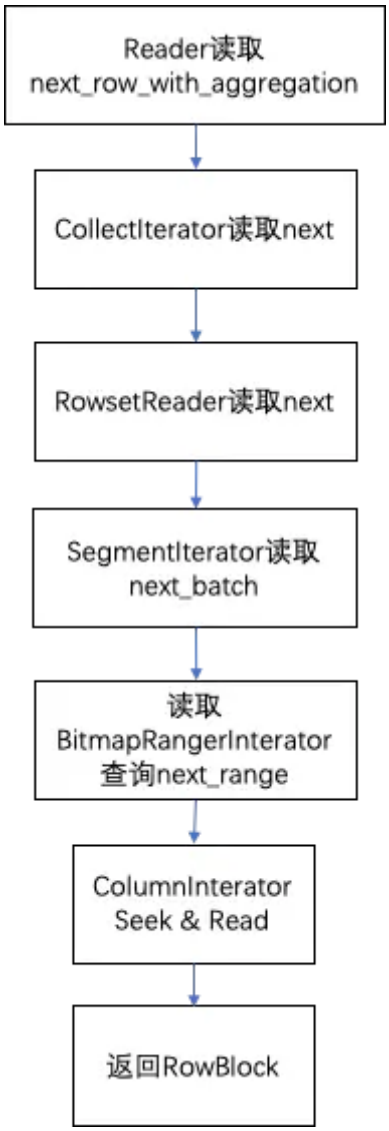
1. 构建SegmentIterator并过滤掉delete_handler中比当前Rowset版本小的删除条件。
2. 构建RowwiseIterator（对SegmentIterator的聚合iterator），将要读取的SegmentIterator加入到RowwiseIterator。当所有Segment为整体有序时采用union iterator顺序读取的方式，否则采用merge iterator归并读取的方式。

2.2.4 SegmentIterator的Init流程

1. 初始化ReadableBlock，用来读取当前的Segment文件的对象，实际读取文件。
2. 初始化_row_bitmap，用来存储通过索引过滤后的行号，使用bitmap结构。
3. 构建ColumnIterator，这里仅是需要读取列。
4. 如果Column有BitmapIndex索引，初始化每个Column的BitmapIndexIterator。
5. 通过SortkeyIndex索引过滤数据。当查询存在key_ranges时，通过key_range获取命中数据的行号范围。步骤如下：（1）根据每一个key_range的上、下key，通过Segment的SortkeyIndex索引找到对应行号upper_rowid，lower_rowid，然后将得到的RowRanges合并到row_bitmap中。
6. 通过各种索引按条件过滤数据。条件包括查询条件和删除条件过滤信息。（1）按查询条件，对条件中含有bitmap索引的列，使用bitmap索引进行过滤，查询出存在数据的行号列表与row_bitmap求交。因为是精确过滤，将过滤的条件从Condition对象中删除。（2）按查询条件中的等值（eq，in，is）条件，使用BloomFilter索引过滤数据。这里会判断当前条件能否命中Page，将这个Page的行号范围与row_bitmap求交。（3）按查询条件和删除条件，使用ZoneMapIndex过滤数据，与ZoneMap每个Page的索引求交，找到符合条件的Page。ZoneMapIndex索引匹配到的行号范围与row_bitmap求交。
7. 使用row_bitmap构造BitmapRangerInterator迭代器，用于后续读取数据。

2.3 读取next阶段的主要流程

next阶段的执行流程如下：



2.3.1 Reader读取next_row_with_aggregation

在reader读取时预先读取一行，记录为当前行。在被调用next返回结果时会返回当前行，然后再预先读取下一行作为新的当前行。

1. （reader的读取会根据模型的类型分为三种情况。
2. _dup_key_next_row读取（明细数据模型）下，返回当前行，再直接读取CollectorIterator读取next作为当前行。
3. _agg_key_next_row读取（聚合模型）下，会取CollectorIterator读取next之后，判断下一行是否与当前行的key相同，相同时则进行聚合计算，循环读取下一行；不相同则返回当前累计的聚合结果，更新当前行。
4. _unique_key_next_row读取（unique key模型）下，与_agg_key_next_row模型方式逻辑相同，但存在一些差异。由于支持了删除操作，会查看聚合后的当前行是否标记为删除行。如果为删除行舍弃数据，直到找到一个不为删除行的数据才进行返回。

2.3.2 CollectIterator读取next

CollectIterator中使用heap数据结构维护了要读取RowsetReader集合，比较规则如下：按照各个RowsetReader当前行的key的顺序，当key相同时比较Rowset的版本。

1. CollectIterator从heap中pop出上一个最大的RowsetReader。
2. 为刚pop出的RowsetReader再读取下一个新的row作为RowsetReader的当前行并再放入heap中进行比较。读取过程中调用RowsetReader的nextBlock按RowBlock读取。（如果当前取到的块是部分删除的page，还要对当前行按删除条件对行进行过滤。）
3. 取队列的top的RowsetReader的当前行，作为当前行返回

2.3.3 RowsetReader读取next

1. RowsetReader直接读取了RowwiseIterator的next_batch。
2. RowwiseIterator整合了SegmentIterator。当Rowset中的Segment整体有序时直接按Union方式迭代返回。当无序时按Merge归并方式返回。RowwiseIterator同样返回了当前最大的SegmentIterator的行数据，每次会调用SegmentIterator的next_batch获取数据。

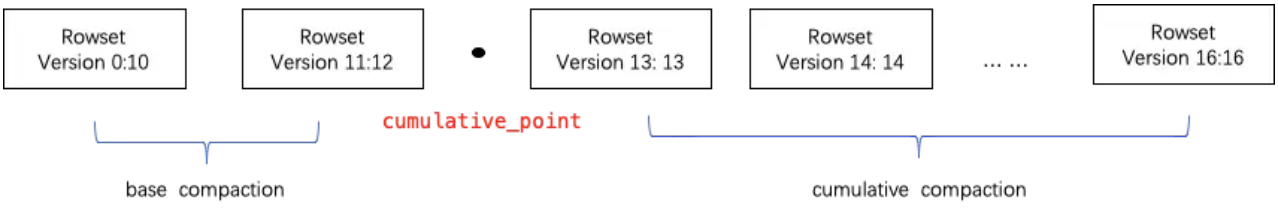
2.3.4 SegmentIterator读取next_batch

1. 根据init阶段构造的BitmapRangerInterator，使用next_range每次取出要读取的行号的一个范围range_from、range_to。
2. 先读取条件列从range_from到range_to行的数据。过程如下：
3. 调用有条件列各个columnIterator的seek_to_ordinal，各个列的读取位置current_rowid定位到SegmentIterator的cur_rowid。这里是通过二分查ordinal_index对齐到对应的data page。
4. 读出条件列的数据。按条件再进行一次过滤（这次是精确的过滤）。
5. 再读取无条件列的数据，放入到Rowblock中。返回Rowblock。

3、Compaction流程

3.1、Compaction整体介绍

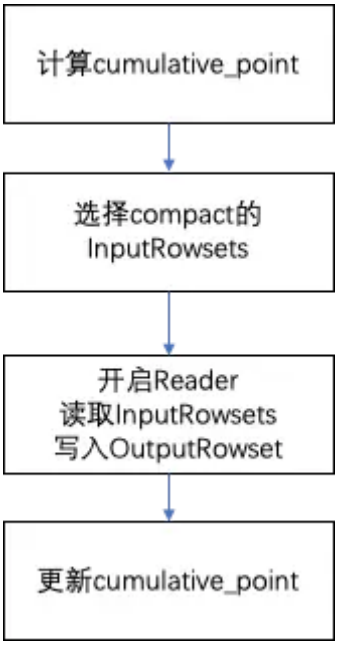
Doris通过Compaction将增量聚合Rowset文件提升性能，Rowset的版本信息中设计了有两个字段first、second来表示Rowset合并后的版本范围。当未合并的cumulative rowset的版本first和second相等。Compaction时相邻的Rowset会进行合并，生成一个新的Rowset，版本信息的first，second也会进行合并，变成一个更大范围的版本。另一方面，compaction流程大大减少rowset文件数量，提升查询效率。



如上图所示，Compaction任务分为两种，base compaction和cumulative compaction。cumulative_point是分割两种策略关键。可以这样理解理解，cumulative_point右边是从未合并过的增量Rowset，其每个Rowset的first与second版本相等；cumulative_point左边是合并过的Rowset，first版本与second版本不等。base compaction和cumulative compaction任务流程基本一致，差异仅在选取要合并的InputRowset逻辑有所不同。

3.2、Compaction详细流程

Compaction合并整体流程如下图所示：



（1）计算cumulative_point。

（2）选择compaction的需要合并的InputRowsets集合：

base compaction选取条件：

- 当存在大于5个的非cumulative的rowset，将所有非cumulative的rowset进行合并。
- 版本first为0的base rowset与其他非cumulative的磁盘比例小于10:3时，合并所有非cumulative的rowset进行合并。
- 其他情况，不进行合并。

cumulative compaction选取条件：

- 选出Rowset集合的segment数量需要大于等于5并且小于等于1000（可配置），进行合并。
- 当输出Rowset数量小于5时，但存在删除条件版本大于Rowset second版本时，进行合并（让删除的Rowset快速合并进来）。
- 当累计的base compaction和cumulative compaction都时间大于1天时，进行合并。
- 其他情况不合并

（3）执行compaction

Compaction执行基本可以理解为读取流程加写入流程。这里会将待合并的inputRowsets开启Reader，然后通过next_row_with_aggregation读取记录。写入到输出的RowsetWriter中，生产新的OutputRowset，这个Rowset的版本为InputRowsets版本全集范围。

（4）更新cumulative_point

更新cumulative_point，将cumulative compaction的产出的OutputRowset交给后续的base compaction流程。

Compaction后对于aggregation key模型和unique key模型分散在不同Rowset但相同key的数据进行合并，达到了预计算的效果。同时减少了Rowset文件数量，提升了查询效率。

4、总结

本文详细介绍了Doris系统底层存储层的读取相关流程。读取流程依赖于完全的列存实现，对于olap的宽表场景（读取大量行，少量列）能够快速扫描，基于多种索引功能进行过滤（包括short key、bloom filter、zoon map、bitmap等），能够跳过大量的数据扫描，还进行了延迟物化等优化，可以对应多种场景的数据分析；Compaction执行流程同样做了分场的优化。能够保证数据量接近的Rowset结合进行compact，减少IO操作提升效率。