[HBASE-14918] CellBlocksSegment and CellBlocksSegmentScanner Classes Design

January 27, 2016

Eshcar Hillel (eshcar@yahoo-inc.com)

Overview and Motivation

In previous issues we presented the design of a compacted memstore. In a nutshell, compacted memstore consists of an active segment (absorbing most recent updates), a snapshot segment (being flushed to disk), and a pipeline of segments. The segments in the pipeline are compacted in the background to reduce the memory consumption of the memstore. Compaction means de-duplicating cells by discarding obsolete entries. By reducing memory consumption, the memstore allows data that was not discarded to continue reside in-memory without being flushed to disk, thereby improving data retrieval, as both scans and gets are served from memory.

In previous design, segments in the pipeline store their data in a skip-list data structure. This format bears unnecessary memory overhead, and is not optimal for reads as it is not cache friendly.

Following the design suggested in HBASE-10713, we present the design of a new segment, *CellBlocksSegment*, for pipeline segments. It stores data in a flat format similar to the way HFiles store their data. We also present the design of a *CellBlocksSegmentScanner* which allows to iterate over the cells in the segment. This is required both for supporting external scans done by the user, as well as internal scans done by the process that is in charge of compacting the pipeline segments.

CellBlocksSegment

CellBlocksSegment is an immutable segment that cannot be changed after creation.
In this segment, cells are stored in a list of blocks. Each *block* is a PositionedByteRange (essentially encapsulating byte array), and the list is manifested as an array of PositionedByteRange.

To facilitate searching keys in the segment, HBASE-10713 suggested using an index (tree-map data structure) on top of the blocks list. However, since blocks are stored in an array of fixed size, an alternative design choice is to replace the index with a binary search. In addition, since blocks are immutable, we can employ binary search within the block (byte array) itself.

Since the segment is immutable, the block is a SimplePositionedByteRange which is a read-only byte array.

Additional auxiliary attributes are:
MemStoreLAB allocator - used to allocate the byte arrays for the blocks.
cell counter - set during construction, immutable

CellBlocksSegment implements the following API:

SegmentScanner getScanner(**long** readPoint) - *returns an instance of a CellBlocksSegmentScanner*
KeyValueScanner getScannerForMemStoreSnapshot() - *Builds a special scanner for the MemStoreSnapshot object that may be different than the general segment scanner.*
boolean isEmpty() - *Returns whether the segment has any cells, cell counter > 0*
int getCellsCount() - *Returns number of cells in segment, content of cell counter*
***long** add(Cell cell)* - *Adds the given cell into the segment, should be called only during build perion*
Cell *getFirstAfter(***Cell** *cell)* - *Returns the first cell in the segment that has equal or greater key than the given cell. Depending on the final design, use the index or a binary search to find the relevant block and then search for the key within this block.*
***void** close()* - *Closing a segment before it is being discarded, closes the allocator*
Cell** maybeCloneWithAllocator(Cell** cell)* - *If the segment has a memory allocator the cell is being cloned to this space, and returned; otherwise the given cell is returned*

CellBlocksSegmentScanner

CellBlocksSegmentScanner extends SegmentScanner.
It utilizes the position in a PositionedByteRange block to traverse keys in the segment.

Auxiliary attributes are:
CellBlocksSegment segment - the scanned segment
long readPoint - the highest relevant MVCC version to retrieve
long currBlock - index of current block in block array

CellBlocksSegmentScanner implements the following API:

Cell peek() - *peeks at the next Cell in this scanner, doesn't change the scanner*
Cell next() - *Return the next Cell in this scanner, and advances the position of the scanner*
boolean seek(Cell cell) - *Seek the scanner at or after the specified Cell.*
boolean reseek(Cell cell) - *Reseek the scanner at or after the specified KeyValue.*
void close() - *decrements segment's scanner counter*
boolean backwardSeek(Cell cell) - *Seek the scanner at or before the row of specified cell*
boolean seekToPreviousRow(Cell cell) - *Seek the scanner at the first Cell of the row which is the previous row of specified cell*
boolean seekToLastRow() - *Seek the scanner at the first Cell of last row*
boolean shouldSeek(Scan scan, **long** oldestUnexpiredTS) - *returns true if scan spans current scanner*