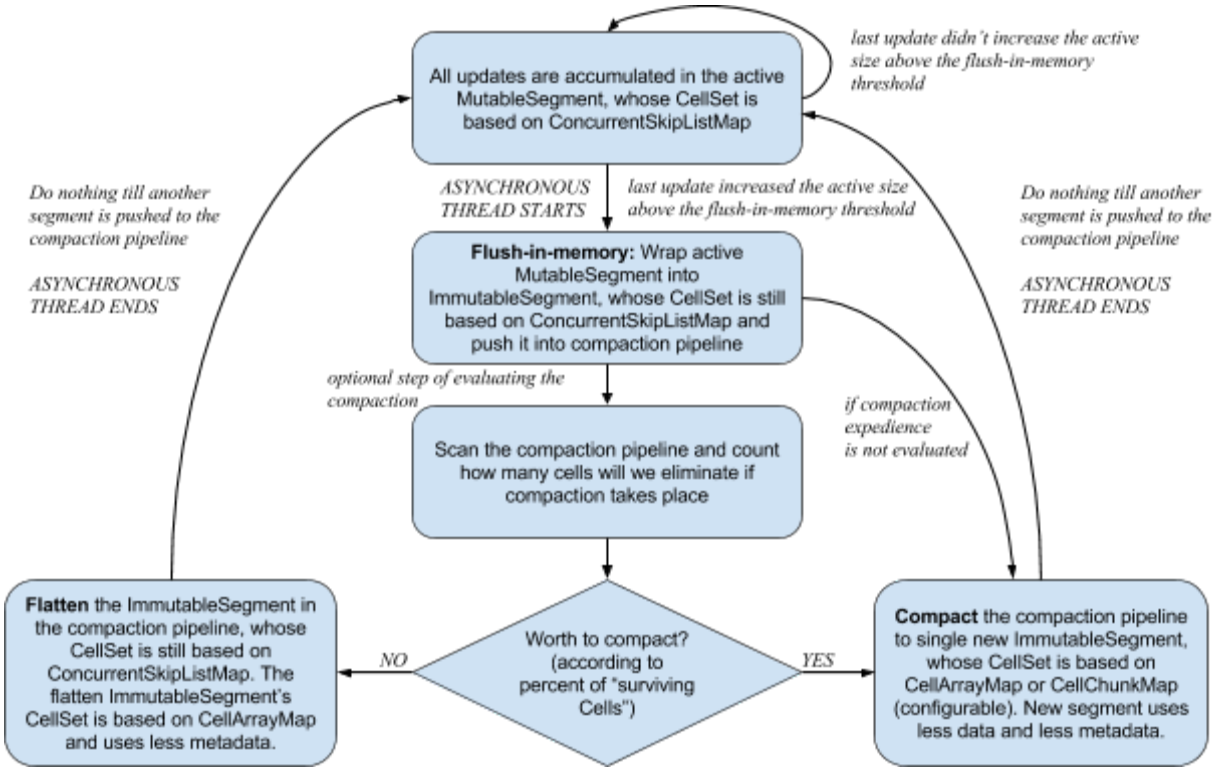
Introduction to New, Flat and Compact MemStore

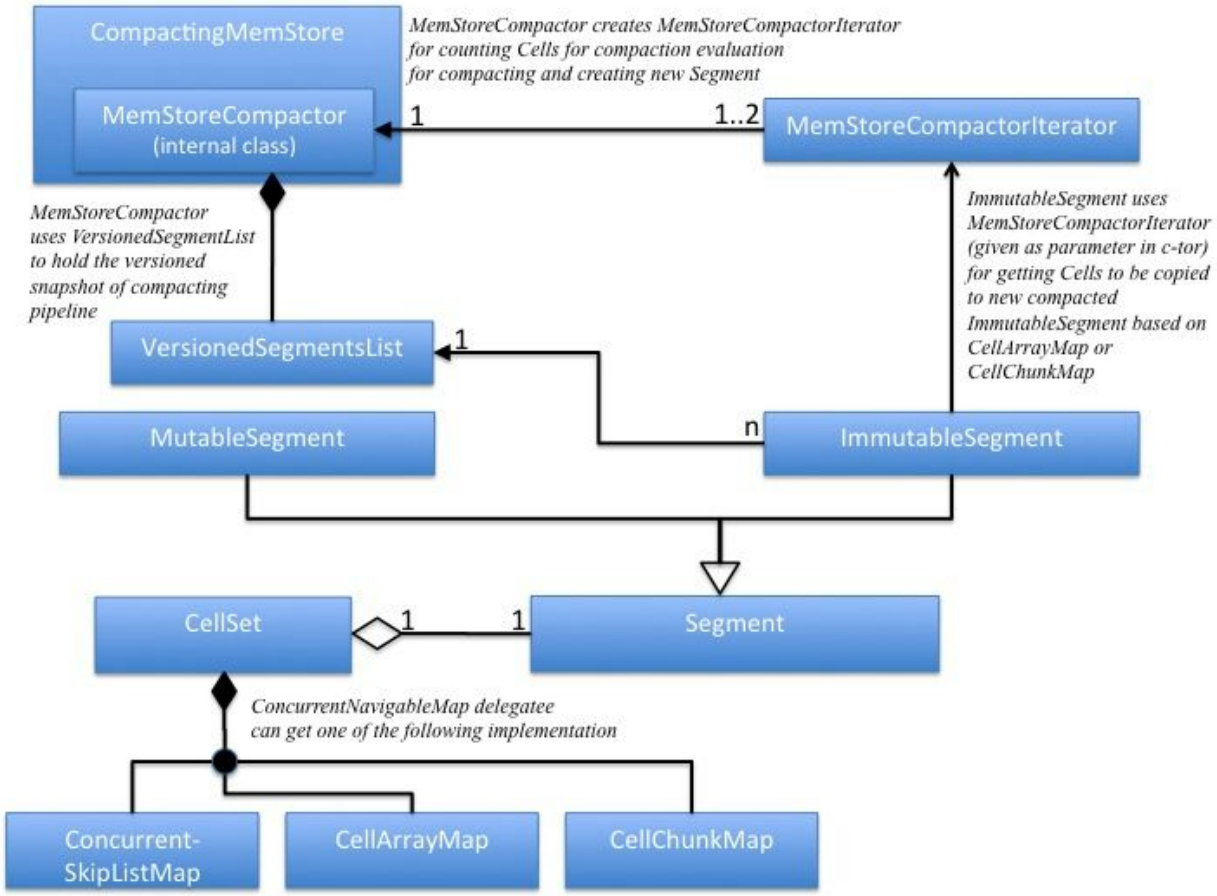
The suggested CompactingMemStore uses previously introduced CellArrayMap and CellChunkMap as an ImmutableSegment metadata. CellSet of ImmutableSegment can be CellArrayMap or CellChunkMap. In addition we present the Segment "*flattening*" option as an alternative to compaction when there is no need for compaction. Flattening is replacing of the ConcurrentSkipListMap index with CellArrayMap without actually touching the Cells.

We separate flush-in-memory and compaction. The former has its own merits, even if there is not enough redundancy to justify compaction. We flatten the index, thereby reducing the metadata memory overhead. If there's no compaction we don't spend a lot of CPU cycles on copy (although the speculative check does). When there is, we reduce the footprint here too. Either way, the I/O is delayed, and more data can be served off-memory. Most of the exploited memory is flat, which enables efficient off-heaping (we just need to address the problem of efficient flattening in the absence of compaction). All that would not have been possible without introducing multi-segment index structure.

Bird's Eye Flow Chart:



Partial Class Diagram:



Todo:

1. Take Chunk class out of HeapMemStoreLAB class (related to MemStoreChunkPool, CellChunkMap, MemStoreLAB, ImmutableSegment, etc.)
2. Arrange MemStoreChunkPool to always be not-null
3. Take care for the super-large-cells case, when Cells are bigger than Chunks (current workaround use only CellArrayMap in this case)
4. Arrange flattening also to CellChunkMap, currently impossible to get the chunk ID out of already created cell metadata
5. Create three subclasses of ImmutableSegment according to the type (software engineering)
6. Improve testing of the new sizing (changes due to new types of ImmutableSegment)
7. Improve testing of the CellChunkMap