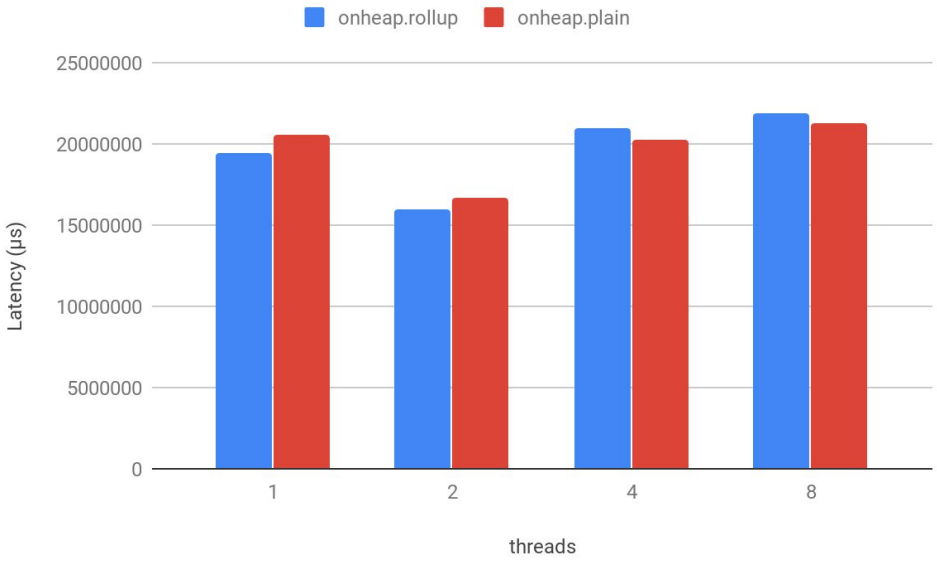
Incremental Index Scaling

Our work on Oak (see PR #7676), shows that there are significant performance gains with multi-threaded indexing (even when not using Oak). In our benchmarks we noticed that ingestion was not scaling as expected with multiple threads.

We added a multi-threaded ingestion benchmark (see `IndexIngestionMultithreadedBenchmark.Java)` and tested scaling with 1, 2, 4, and 8 threads. The figure below shows the average latency, in microseconds, of ingesting 1M rows using the `basic` schema:



We traced the threads' blocking states to two causes:

1. A monitor in IncrementalIndex that synchronized access to `dimensionDescs`
2. A Read-Write lock in `StringDimensionIndexer`

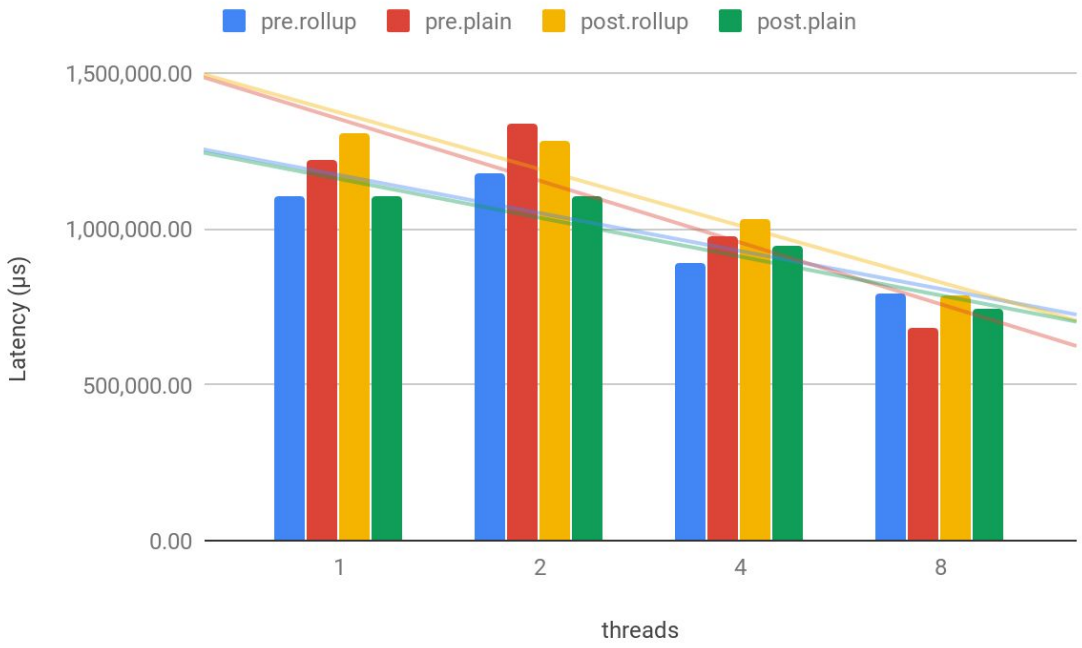
This PR proposes a solution to the first issue. The proposed solution is based on the observation that dimension data is updated infrequently and so ongoing exclusive locking is wasteful.

Summary of changes:

1. Shared state is encapsulated in a new class - `DimensionData`. This includes `dimensionDescs`, `dimensionDescsList` and `columnCapabilities`
2. Concurrent threads share an atomic reference to an instance of `DimensionData`
3. CoW: Only when a thread needs to update the shared state, it will copy the instance, update the copy, and eventually swap the reference atomically.
4. Consistency is maintained when the reference is updated. This simplifies row processing, removes the need for keeping an "overflow" array, and allows fast failure when a row contains duplicate dimensions.

To avoid the 2nd synchronization issue with string indexer, we use the `simpleFloat` schema.

Latency measurements show performance gains when using multiple threads, which scale with the number of threads, i.e., as thread number increases, latency strictly decreases. Where performance is lower, it is not by much and falls within measurement variance. We believe once the string indexer contention issue is remediated, performance gain in the general case will be even more noticeable.



threads	pre.rollup	pre.plain	post.rollup	post.plain
1	1,101,815.63	1,223,339.74	1,307,184.28	1,104,051.11
2	1,175,196.61	1,338,503.15	1,281,036.41	1,102,151.39
4	891,858.43	974,796.39	1,028,691.17	946,309.16
8	790,267.95	684,077.97	785,634.55	740,663.35