KIP-428: Add in-memory window store

Status

Current state: Accepted

Discussion thread: here

JIRA: KAFKA-4730

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Streams currently has both an in-memory and persistent implementation of its key-value store, but only a persistent windowed store. Extending the current window store with an in-memory implementation would allow for significant performance increase and has been requested by a number of users.

Public Interfaces

This KIP adds one public method to Stores

```
public static WindowBytesStoreSupplier inMemoryWindowStore(final String name,
                                                           final Duration retentionPeriod,
                                                           final Duration windowSize,
                                                           final boolean retainDuplicates) throws
IllegalArgumentException {
}
```

Proposed Changes

The in-memory window store will allow for single and range fetch, both for a range of timestamps and keys. Expired records will be removed as soon as possible in order to free up resources. Using a NavigableMap we can provide logN time for operations such as put and fetch. Users can expect that the overall memory footprint includes the space required for all live records plus some additional space proportional to the number of fetched records that have not yet had their iterator closed. It is therefore important that users read these results and close the iterator as soon as possible.

Compatibility, Deprecation, and Migration Plan

N/A

Rejected Alternatives

Several alternative designs for the in-memory window store were considered. One idea was to follow the segmented approach of the persistent window store, which groups records into larger time blocks for efficient batch deletion. Only after the last record in a segment has expired is the entire segment deleted, meaning some older records may exist for a while after they have technically expired, depending on the size of the segment. This tradeoff between time and space makes sense for RocksDB, but ultimately was rejected for the in-memory implementation as we would prefer to reclaim the resources as soon as possible after expiration.