KIP-540: Implement per key stream time tracking

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA: *here*

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

Motivation

Currently, Streams uses a concept of "stream time", which is computed as the highest timestamp observed by stateful operators, per partition. This concept of time backs grace period, retention time, and suppression.

For use cases in which data is produced to topics in roughly chronological order (as in db change capture), this reckoning is fine.

Some use cases have a different pattern, though. For example, in IOT applications, it's common for sensors to save up quite a bit of data and then dump it all at once into the topic. See https://stackoverflow.com/questions/57082923/kafka-windowed-stream-make-grace-and-suppress-key-aware for a concrete example of the use case.

There have been cases where each sensor dumps 24 hours' worth of data at a time into the topic. This results in a pattern in which, when reading a single partition, the operators observe a lot of consecutive records for one key that increase in timestamp for 24 hours, then a bunch of consecutive records for another key that are also increasing in timestamp over the same 24 hour period. With our current stream-time definition, this means that the partition's stream time increases while reading the first key's data, but then stays paused while reading the second key's data, since the second batch of records all have timestamps in the "past".

E.g:

```
A@t0 (stream time: 0)
A@t1 (stream time: 1)
A@t2 (stream time: 2)
A@t3 (stream time: 3)
B@t0 (stream time: 3)
B@t1 (stream time: 3)
B@t2 (stream time: 3)
B@t3 (stream time: 3)
```

This pattern results in an unfortunate compromise in which folks are required to set the grace period to the max expected time skew, for example 24 hours, or Streams will just drop the second key's data (since it is late). But, this means that if they want to use Suppression for "final results", they have to wait 24 hours for the result.

This tradeoff is not strictly necessary, though, because each key represents a logically independent sequence of events. Tracking by partition is simply convenient, but typically not logically meaningful. That is, the partitions are just physically independent sequences of events, so it's convenient to track stream time at this granularity. It would be just as correct, and more useful for IOT-like use cases, to track time independently for each key.

Public Interfaces

For per key stream time tracking to be scalable, a cache-backed store will need to be created. Consequently, this would require user approval before something like this occurs. In which case, a configuration will be added – `allow.stream.time.key.tracking` , a boolean config which allows the creation of stores.

Proposed Changes

Fundamentally speaking, there will be a few configurations added, but public API wise, nothing really needs to be modified. What we are aiming for is a behavioral change *internally*.

Here are the few core steps in implementing this per key stream time tracking system. When considering the number of keys that we will have to track, it is not uncommon to have millions of unique keys that we might have to process. In effect, storing them all in memory is too costly. The best solution for this case would be a cache backed by a state store, in which case, Kafka's current `CachingKeyValueStore` fulfills that role perfectly. Given that the user allows us to create these stores, how we track the timestamps should be relatively simple:

1. When a topology is initialized, we will register a (or multiple) `CachingKeyValueStore`(s) in StreamThread's `ProcessorContext`.
2. After which, during processing, if it is necessary for finer granularity of stream time tracking, we will get a store from the ProcessorContext (which exists perhaps under the name "stream-time-table").
3. We will use that store as a field which can be easily accessed for stream time updates and queries.
    a. Suppressors and any other processors involving the calculation of grace periods would probably require a store should the user choose it. (whether each processor has an independent store or a shared one is still up for discussion.

The behavior the user should expect in this instance is that instead of records being evicted on a per partition basis, it will be per key. To illustrate, imagine we have the same records list as the example provided in the Motivation section. We could see that we do not have to wait for 24 hours to get the records, nor are keys dropped in partition B because they are not considered late (as stream time now is *per key,* and thus, the stream time upon which a record is determined if it is evicted is determined solely by records of the *same key*).

The metrics that are reported for this design has yet to be decided upon.

Compatibility, Deprecation, and Migration Plan

If a Kafka Streams instance is migrated/restarted, we should check for the whether or not a previous store was constructed under the same location. If that is the case, we should try to at the very least partially restore the contents of that table to retain accuracy.

Rejected Alternatives

TBD after discussion. N/A for the moment.