KIP-353: Improve Kafka Streams Timestamp Synchronization

Status

Current state: *Accepted*

Discussion thread: *link*

JIRA: **KAFKA-3514** - Getting issue details... STATUS

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

Motivation

Today Kafka Streams stream time inference and synchronization is tricky to understand and also introduces much non-detereminism into the processing ordering when a task is fetching from multiple input streams (a.k.a. topic-partitions from Kafka).

More specifically, a stream task may contain multiple input topic-partitions. And hence we need to decide 1) which topic-partition to pick the next record to process for this task, 2) how stream time will be advanced when records are being processed. Today this logic is determined in the following way:

a) Each topic-partition maintains a monotonically increasing timestamp that depends on the buffered records' timestamps.

b) When processing a task, we pick the head record from the partition that has the smallest time (picking head record is to guarantee offset ordering within a partition) AND has non-empty buffered records.

c) The task's stream time is defined as the smallest value across all of its partitions' timestamp, and hence is also monotonically increasing.

The above behavior is based on the motivations that for operations such as joins, we want to synchronize the timestamps across multiple input streams so that we can avoid "out-of-ordering" data across multiple topic-partitions in a best effort (note that within a single topic-partition we always process data following the offset ordering).

We have observed the above mechanism has the following issue:

- Users have no controls on out-of-ordering / non-detereminism and latency trade-offs: we always try to process the task as long as at least one of its partitions have buffered data; it means that when other partitions eventually have enqueued more data, we may find their timestamps to be actually smaller than what we have already processed, i.e. out-of-ordering. During re-process, since partitions data will be available immediately, we can then pick records that following the timestamp ordering, and its result will not be the same as the first time we execute this task, i.e. non-deterministic behavior when we execute the task multiple times.

In this KIP we propose to improve on this situation, to allow users tune the behavior of picking-next-record-to-process based on the stream time.

Public Interfaces

First, we will introduce the term of task "processability" , as the following:

- *A task is only processable when all of its input topic-partitions have buffered data from the consumer fetches, hence all of its topic-partitions current timestamp "position" is known. And hence it is safe to decide from which partition to pick the next record.*

To overcome the issue that a task may have high-variant incoming traffic from its partitions (consider: one partition has 10K records / sec, while another only have 1 record per sec). We will also allow users to "enforce" a non-processable task to be processed, via the following added config:

```
public static final String MAX_TASK_IDLE_MS_CONFIG = "max.task.idle.ms"     // the maximum wall-clock time a
task could stay to be not processed while still containing some buffered data in at least one of its partitions
```

When a task is enforced to be processed via this config, it could potentially introduce out-of-ordering (i.e. you may be processing a record with timestamp t0 while your stream time has been advanced to t1 > t0) and forcing a task for processing could also be non-deterministic and might result in a different output than the deterministic re-processing case. Users can use this config to control how much they can pay for latency (i.e. not processing the task) in order to reduce out-of-ordering possibilities: when it is set to Long.MAX, then we will wait indefinitely on the task before any data may be processed for this task; on the other extreme case, when it is set to 0 we will always process the task whenever it has some data, and bear in mind that such an enforced processing may cause out-of-ordering. And in order to let users be notified qualitatively such potential "out-of-ordering" events, we will also introduce a task-level metric:

By default, we will set its value to 0 to be consistent with the current behavior.

METRIC /ATTRIBUTE NAME	**DESCRIPTION**	**MBEAN NAME**	
enforced-processing-rate / total	The average number of enforced process calls on this task per second / total number of enforced process calls so far.	`kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.\w]+)`	

Note that with this change, if users choose to wait indefinitely by setting this config to Long.MAX, then we can eliminate out-of-ordering due to timestamp synchronization completely; but out-of-ordering can still happen from the input topics themselves (think of a single partition whose record's timestamps are not monotonically increasing as their offsets). One of the direct implications of out-of-ordering is the Join operator semantics:

Stream-Stream:

- inner join: computes correct result
- left join: can contain "incorrect" leftRecord-null in the result
- outer join: can contain "incorrect" leftRecord-null or null-rightRecord in the result
- all three variants handle out-of-order records correctly

Stream-Table (inner and left):

- time synchronization will be guaranteed via setting the above config value
- out-of-order records are not handled (ie, we don't check for out-of-order record and just process all records in offset order): produces unpredictable results

Table-Table (inner/left/outer):

- time synchronization will be guaranteed via setting the above config value
- out-of-order records are not handled (ie, we don't check for out-of-order record and just process all records in offset order)
- however, Table-Table joins are eventual consistent

Proposed Changes

User's observable changes have been summarized above.

Compatibility, Deprecation, and Migration Plan

- *From user's perspective, this proposal only contains additional config and metrics, so it should be compatible with their existing code.*
- *Some processing behavior maybe altered once users override the default values of the config, leading to performance differences; but should not lead to any incompatibility issues.*

Rejected Alternatives

No rejected alternatives so far.