KIP-259: Improve Streams DSL Timestamp Propagation Semantics

Status

Current state: *"Under Discussion"*

Discussion thread: *TODO*

JIRA: *KAFKA-6455*

Released: 1.2

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

Motivation

Kafka Streams DSL inherits the timestamp propagation "contract" from the Processor API. In this "contract", the output record gets the timestamp of the current input record assigned. For many operators like filter, map, etc this contract is reasonable. However, for more complex operators like aggregation or joins, this contract in not optimal, as it does not provide strong guarantees about the timestamp of the result records. This make reasoning about the expected result hard and non-deterministic with regard to the record timestamp. However, the timestamp in event-time stream processing is as important as the actual data. Hence, we should define a DSL level contract that matches the semantics of the corresponding DSL operator.

Public Interfaces

No public interfaces changes. This KIP suggests a semantical change only.

Proposed Changes

For the following operators nothing changes:

- KStream: filter, filterNot, map, mapValues, flatMap, flatMapValues, selectKey, merge, branch, groupByKey, groupBy
- KTable: filter, filterNot, mapValue, groupByKey
- KGroupedStream: windowedBy
- KStream-Table-join (inner and left)

For this operations, all output record (note, that flatMap and flatMapValues might have multiple output records for a single input record) get the same timestamp as the input record. Also note, that for KStream-KTable join, only the KStream input can trigger an output record.

We suggest to change the semantics for timestamp inheritance for the following operators:

- KGroupedStream/KGroupedTable: count/reduce/aggregate
  - output records will have timestamps "max over all processed records"
  - example (only showing the timestamps of input and output records):
    ```
    input:       1, 2, 5, 6, 4, 3, 7, 9
    output-old:  1, 2, 5, 6, 4, 3, 7, 9
    output-new:  1, 2, 5, 6, 6, 6, 7, 9
    ```
    the new semantics ensure, that time does not go "backwards" for aggregations what is undesired behavior atm
- TimeWindowedStream/SessionWindowedStream: count/reduce/aggregate
  - output records will have timestamps "max over all records in a window"
  - example with tumbling window of size 10 (only showing the timestamps of input and output records)
    ```
    input:          1, 2, 5, 6, 4, 3, 7, 9

      output-old-w0:  1, 2,       4, 3
      output-old-w1:        5, 6,       7, 9

      output-new-w0:  1, 2,       4, 4
      output-new-w1:        5, 6,       7, 9
    ```
    the new semantics ensure, that time does not go "backwards" for aggregations within a single window what is undesired behavior atm

- Joins: KStream-KStream and KTable-KTable:
    - output records will have timestamp "max(r1.t,r2.t)", ie, the larger of both joining records
    - the idea is, that the output record's timestamp should reflect the ***event-time*** when the result record could have been computed *earliest*: assume two streams containing adds and clicks and we want to join clicks with adds. We know, that adds must have a smaller timestamp than clicks and a click on an add happens when the click itself is done. If for some reason, we process some clicks before the actual add, currently, all output records would get the timestamp of the add assigned (even if this timestamp is smaller). With the new semantics, we ensure that we use the timestamp of the click.
    - in other words: if there are two record r1 and r2 with r1.t < r2.t, assigning r1.t to the output records is undesired, because at this point in time (event-time), the output record cannot possible exist, because r2 does not exist yet.

This KIP builds on KIP-259 and KIP-251. We can store timestamps for aggregations and join in the underlying RocksDB stores and thus use "max (oldTimestamp, recordTimestamp)" (for aggregation) and "max(r1.t,r2.t)" (for joins) to compute the output record's timestamp and set the timestamp explicitly when forwarding the output record downstream.

Compatibility, Deprecation, and Migration Plan

This is a semantical change and is thus backward compatible.

Test Plan

We test the new behavior of the operators with existing unit and integration tests that we update to also check the output record timestamps.

Rejected Alternatives

Note.