KIP-613: Add end-to-end latency metrics to Streams

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: KAFKA-9983, KAFKA-10054

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

Motivation

Currently the actual end-to-end latency of a record flowing through Streams is difficult to gauge at best. It's also difficult to build real-time applications without some sense of the latency you can expect between the time that an event occurs and when this event is processed and reflected in the output results. Being able to bound this latency is an important requisite for many apps, and exposing this through metrics should go a long way towards enabling users to make the right design choices.

For example, application builders may be producing events in reaction to user actions. Without a sense of how long it will take for this new event to be reflected in the results, how do you know what to do next – should they show a brief loading screen before issuing an interactive query, or will they need to wait hours for this record to be processed? (Hopefully not). Without end-to-end latency metrics this question is difficult to answer.

Public Interfaces

The following metrics would be added:

- record-e2e-latency-min [ms]
- record-e2e-latency-max [ms]
- record-e2e-latency-avg [ms]

These will be exposed on the **task-level** with the following tags:

- type = stream-processor-node-metrics
- thread-id=[threadId]
- task-id=[taskId]
- processor-node-id=[processorNodeId]

These will be reported for **source and terminal operators** at the recording level **INFO**

We will also expose these metrics for **stateful operators** at the recording level **TRACE** (which is also being added as part of this KIP)

In all cases the metrics will be computed at the *end* of the operation, once the processing has been complete

Update

The min and max task-level INFO metrics have been added in 2.6, and the remaining metrics will ship in the next version

Proposed Changes

Imagine a simple 3-node subtopology with source node **O**, filter node **F**, aggregation **A**, and sink node **I.** For any record flowing through this with record timestamp **t**, let $\mathbf{t_O}$ be the system (wallclock) time when it is sent from the source topic, $\mathbf{t_A}$ be the time when it is finished being processed by the aggregator node, and $\mathbf{t_I}$ be the time when it leaves the sink node for the output or repartition topic. The *end-to-end latency* at operator **O** for a given record i s defined as

$$L_O(t) = t_O - t$$

and likewise for the other operator-level end-to-end latencies. This represents the age of the record at the time is was processed by operator **O.** The task-level end-to-end (e2e) latency **L** will be computed based on the sink node, ie $L = L_I$. The source node e2e latency reading from the user input topics therefore represent the ***consumption latency***, the time it took for a newly-created event to be read by Streams. This can be especially interesting in cases where some records may be severely delayed: for example by a IoT device with unstable network connections, or when a user's smartphone reconnects to the internet after a flight and pushes all the latest updates. On the other side, the sink node e2e latency – which is also the task-level e2e latency, reveals how long it takes for the record to be fully processed through that subtopology. If the task is the final one in the full topology, this is the ***full end-to-end latency*****:** the time it took for a record to be fully processed through Streams.

Note that for a given record, $L_O <= L_A <= L_I$. This holds true within and across subtopologies. A downstream subtopology will always have a task-level end-to-end latency greater than or equal to that of an upstream subtopology for a single task (which in turn implies the same holds true for the statistical measures exposed via the new metrics). Comparing the e2e latency across tasks (or across operators) will also be of interest as this represents the ***processing delay***: the amount of time it took for Streams to actually process the record from point A to point B within the topology.

Late arriving records will be included in this metric, even if they are otherwise dropped due to the grace period having passed. This metric is related but ultimately orthogonal to the concept of late-ness. One difference for example is that they are dropped based on stream time, whereas this metric is always reported with respect to the current time. The stream-time may lag the system time in low traffic, for example, or when *all* records are considerably delayed. This might mean the user sees no dropped records even though the staleness is large.

Compatibility, Deprecation, and Migration Plan

N/A

Rejected Alternatives

Using stream time

> We define the staleness in terms of individual record timestamps, but we could have instead it as the difference between the system time and the stream time, ie $S_O = t_O - st$ where **st** is the stream-time. This approach has some drawbacks; first and foremost, we are losing information in the case of out-of-order data, and may not notice records with extreme delays. In the example above, an IoT device that regularly disconnects for long periods of time would push a lot of out-of-order data when it reconnects. Since these records would not advance or effect the stream-time, this delay would not be reflected as an increase in the processing latency metric. But the end-to-end latency of these records is of course quite high

Computing the metric at record intake time

> This idea was originally discussed but ultimately put to rest as it does address the specific goal set out in this KIP, to report the time for an event to be reflected in the output. This alternative metric, which we call "staleness", has some use as a gauge of the record time when received by an operator, which may have implications for its processing for some operators. However this issue is orthogonal and thus rejected in favor of measuring at the record output.