KIP-846: Source/sink node metrics for Consumed /Produced throughput in Streams

- Status
- Motivation
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives

Status

Current state: Adopted

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

With the metrics available today it is possible for users of Kafka Streams to derive the **consumed** throughput of their applications at the subtopology level, but the same is not true for the **produced** throughput. The consumer client currently reports a cumulative sum metric at the topic level, ie the **-total** metrics which reports both the number of records and the bytes consumed. These are tagged with the topic name and the (consumer) client id, which is sufficient for computing the bytes/records consumed per subtopology by aggregating over all clients in the Stream application. This computation relies on the fact that Kafka Streams topologies can only consume from a given topic at most **once**. Unfortunately there is no such guarantee on the other end, and applications can have one or more subtopologies producing to any given output topic.

To fill in this gap and give end users a way to compute the production rate of each subtopology, this KIP proposes two new metrics for the throughput at its sink nodes.

Furthermore, we propose to also add the corresponding metrics at the source nodes in order to give a fully granular view of the consumed throughput. Even though one can derive a topic-level view of this using the existing client-level metrics, we feel it's worth adding the more granular **-consumed** metrics alongside the **-produced** metrics for two main reasons: (a) to provide an equally fine-grained metrics scope, and (b) to simplify the user experience such that the same processing/parsing logic can be applied to handle the throughput metrics at the sink and source nodes. It's worth noting that a third benefit to adding the corresponding **consumed** metrics lies in enabling users to do more future-proof handling that does not rely on any assumptions about the limits of consumer groups such as the 1:1 mapping of source topic to consumer client.

Public Interfaces

The following metrics would be added:

- bytes-consumed-total
- records-consumed-total
- bytes-produced-total
- records-produced-total

These will be exposed on the **topic level** (newly added metrics scope by this KIP) with the following tags:

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

The **-consumed** and **-produced** metrics will be reported at the **source** and **sink nodes** respectively, at the recording level **INFO.**

Proposed Changes

Pretty much everything outside of the metric names, recording levels, and tags is an implementation detail so this document won't go much farther here. But just to assuage any concerns about how these metrics will be computed, note that the serialization into raw bytes occurs within Streams *before* handing them off to the embedded Producer client. And so we should have everything we need to record both the number of records and the size in bytes while we are still within Streams code and have the taskId/subtopology number available.

Compatibility, Deprecation, and Migration Plan

N/A – this KIP is only adding new metrics and should not present any compatibility problems.

Test Plan

Since we're just adding metrics we should have sufficient coverage with basic unit and integration tests. We will also monitor for performance regressions in our benchmarks that sink to output topics just in case this incurs some unexpected overhead and would be better suited at the **DEBUG** level. See #2 under Rejected Alternatives for more on this.

Rejected Alternatives

1. Including the corresponding **-rate** metrics alongside the **-total** metrics, as is often done for Streams metrics. For one thing reporting the cumulative sums should be sufficient for any monitoring service to compute the rate if desired, and furthermore allows that service to have full control over how this rate is defined and computed over what window of time.