KIP-104: Granular Sensors for Streams

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

Status

Current state: Accepted

Discussion thread: *here*

JIRA: KAFKA-3715: Higher granularity Streams metrics

Motivation

- This KIP proposes the addition of latency and throughput metrics for Kafka Streams at the granularity of each processor node and the addition of count metrics at the granularity of each task. This is in addition to the global rate (which already exists). The idea is to allow users to toggle the recording of these metrics when needed for debugging. The RecordLevel for these granular metrics is DEBUG, and a client can toggle the record level by changing the "metrics.record.level" in the client config. (The introduction of RecordLevel and client config changes are covered in the separate KIP-105).
- This KIP also proposes exposing the metrics registry as read-only and several helper functions so that Kafka Streams users can register their own metrics.

Public Interfaces

- A StreamsMetrics class with the following methods:

```
@InterfaceStability.Unstable

public interface StreamsMetrics {

/**
 * Get read-only handle on global metrics registry
 * @return Map of all metrics.
 */
Map<MetricName, ? extends Metric> metrics();
```

```
 /**
 * Add a latency sensor. This is equivalent to adding a sensor with metrics on latency and rate.
 *
 * @param scopeName Name of the scope, could be the type of the state store, etc.
 * @param entityName Name of the entity, could be the name of the state store instance, etc.
 * @param recordLevel The recording level (e.g., INFO or DEBUG) for this sensor.
 * @param operationName Name of the operation, could be get / put / delete / etc.
 * @param tags Additional tags of the sensor.
 * @return The added sensor.
 */
 Sensor addLatencyAndThroughputSensor(String scopeName, String entityName, String operationName, Sensor.
RecordLevel recordLevel, String... tags);

 /**
 * Record the given latency value of the sensor.
 * @param sensor sensor whose latency we are recording.
 * @param startNs start of measurement time in nanoseconds.
 * @param endNs end of measurement time in nanoseconds.
 */
 void recordLatency(Sensor sensor, long startNs, long endNs);

 /**
 * Add a throughput sensor. This is equivalent to adding a sensor with metrics rate.
 *
 * @param scopeName Name of the scope, could be the type of the state store, etc.
 * @param entityName Name of the entity, could be the name of the state store instance, etc.
 * @param recordLevel The recording level (e.g., INFO or DEBUG) for this sensor.
 * @param operationName Name of the operation, could be get / put / delete / etc.
 * @param tags Additional tags of the sensor.
 * @return The added sensor.
 */
 Sensor addThroughputSensor(String scopeName, String entityName, String operationName, Sensor.RecordLevel
recordLevel, String... tags);

 /**
 * Records the throughput value of a sensor.
 * @param sensor sensor whose throughput we are recording.
 * @param value throughput value.
 */
 void recordThroughput(Sensor sensor, long value);



/**
 * Generic sensor creation. Note that for most cases it is advisable to use {@link #addThroughputSensor
(String, String, String, Sensor.RecordLevel, String...)}
 * or {@link #addLatencySensor(String, String, String, Sensor.RecordLevel, String...)} to ensure metric
name well-formedness and conformity with the rest
 * of the streams code base.
 * @param name Name of the sensor.
 * @param recordLevel The recording level (e.g., INFO or DEBUG) for this sensor.
 */
Sensor addSensor(String name, Sensor.RecordLevel recordLevel);



/**
 * Same as previous constructor {@link #addSensor(String, Sensor.RecordLevel, Sensor...)} sensor}, but
takes a set of parents as well.
 *
 */
Sensor addSensor(String name, Sensor.RecordLevel recordLevel, Sensor... parents);

/**
 * Remove a sensor with the given name.
 * @param sensor Sensor to be removed.
 */
void removeSensor(Sensor sensor);

}
```

- Furthermore, the KafkaStreams class can also expose all the metrics read-only. So we add the same method we added to the StreamsMetrics interface.

```
/**
 * Get read-only handle on global metrics registry
 * @return Map of all metrics.
 */
Map<MetricName, ? extends Metric> metrics();
```

Proposed Changes

- Enumeration of Sensors: This KIP proposes the introduction of the following sensors
    - Node punctuate time sensor: This sensor is associated with latency metrics depicting the average and max latency in the punctuate time of a node.
    - Node creation time sensor: This sensor is associated with latency metrics depicting the average and max latency in the creation time of a node.
    - Node destruction time sensor: This sensor is associated with latency metrics depicting the average and max latency in the destruction time of a node.
    - Node process time sensor: This sensor is associated with latency metrics depicting the average and max latency in the process time of a node.
    - Node throughput sensor: This sensor is associated with throughput metrics depicting the context forwarding rate of metrics through a node, i.e., indicating how many records were forwarded downstream from this processor node.
    - Skipped records sensor in StreamTask:This sensor is associated with a count metric, which helps monitor if streams are well synchronized. The metric measures the difference in the total record count and the number of added records between the last record time. This is useful during debugging as this count should not be off by too much during normal operations.
    - Finally all metrics can be read as read-only through the metrics() calls.
- Addition of new sensors
    - Users can use the provided helped functions addLatencySensor and addThroughputSensor to register metrics and removeSensor to remove sensors. Note that addLatencySensor already existed in the code base.

Compatibility, Deprecation, and Migration Plan

- none

Rejected Alternatives

- Allow the user to register arbitrary metrics by exposing the Metrics class. Unfortunately the class has been used internally so far and is not ready for becoming public yet (e.g., there are several unnecessary methods in there). This might have to wait until the Metrics class is cleaned up.
- Provide an interface on top of the Metrics class. This is doeable, however StreamMetrics is arguable already such an interface and allows users to register throughput and latency metrics for streams.