KIP-674: Metric Reporter to Aggregate Metrics in Kafka Streams

Status

Current state: Under Discussion

Discussion thread:

JIRA:

- Unable to render Jira issues macro, execution error.

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

Motivation

Kafka Streams exposes metrics on various levels. The number of metrics grows with the number of stream threads, the number of tasks (i.e., number of subtopologies and number of partitions), the number of processors, the number of state stores and the number of buffers in a Kafka Streams application.

Some users monitor their Kafka Streams applications with commercial monitoring services. Those services often limit the number of metrics that can be reported to them. Some providers truncate the metrics when the limit is exceeded. That means, that some metrics are then not sent to the monitoring service, which might lead to false alerts. For example, In Kafka Streams the metric `alive-stream-threads` records the number of alive stream threads. Users might configure their monitoring service to alert them on this metric when a stream thread dies. If metric `alive-stream-threads` is removed from the reported metrics because the limit of the number of reported metrics of the monitoring service is exceeded, users will get an alert although no stream thread actually died.

In this KIP, we propose to add a metric reporter to Kafka Streams that can be used to aggregate metrics before they are reported to a monitoring service. In such a way users can avoid exceeding the limit of number of reported metrics of the monitoring service and the associated possible false alerts.

Public Interfaces

```
package org.apache.kafka.streams;

public abstract class MetricsAggregations implements MetricsReporter {

        public static class ValuesProvider<V> implements Iterable<V> {
                public Iterator<V> iterator();
        }

        public interface MetricRegistrar<AGG, V> {
        ValuesProvider<V> register(final Map<String, String> tags);
        void deregister();
    }

        protected <AGG, V> void addAggregation(final String nameOfAggregation,
                                               final String groupOfMetricsToAggregate,
                                               final String nameOfMetricsToAggregate,
                                               final Collection<String> tagsForGrouping,
                                               final MetricRegistrar<AGG, V> metricRegistrar);

}
```

Proposed Changes

We propose to add the above metrics reporter to the Kafka Streams library. The reporter needs to be extended by the users and the extended class will be passed to the Kafka Streams config `metric.reporters`. The behavior of the reporter is described in this section.

As any other metrics reporter passed to a Kafka Streams client, this reporter will be instantiated and its method `configure()` will be called with a map that contains the application ID of the Kafka Streams as client will be passed to the method.

Method `KafkaStreams#addMetricsAggregation()` will create one or more metrics on client-level that record the aggregation of the metrics specified by the arguments `groupOfMetricsToAggregate` and `nameOfMetricsToAggregate`. Before the specified metrics are aggregated, they will be grouped by the tag labels provided in argument `tagLabels`. For example, if users want to aggregate state-store-level metric `size-all-mem-tables` (RocksDB specific metric) grouped by stream threads, they will provide the name `size-all-mem-tables` as argument `nameOfMetricsToAggregate`, the type `stream-state-metrics` as argument `groupOfMetricsToAggregate`, and the list of tag labels [`thread-id`] as argument `tagLabels`. If they additionally want to aggregate the metrics by task, they will provide [`thread-id`, `task-id`] as argument `tagLabels`. If users want to just aggregate by task, they can will provide [`task-id`] as argument `tagLabels`.

Assuming argument `tagLabels` has n elements, the metrics that record the aggregation of the specified metrics are added with the following configuration:

type: stream-metrics
client-id: [client-id]
[`tagLabels.get(0)`]: [tag value of the aggregated metrics for tag label `tagLabels.get(0)`]
...
[`tagLabels.get(n)`]: [tag value of the aggregated metrics for tag label `tagLabels.get(n)`]
name: [name provided as argument `name`]

In the example where users want to aggregate metric `size-all-mem-tables` by stream threads and tasks, the added metric will have the following configuration:

type: stream-metrics
client-id: [client-id]
[`thread-id`]: [thread-id of metrics `size-all-mem-tables` that are aggregated in this metric]
[task-id]: [task-id of metrics `size-all-mem-tables` that are aggregated in this metric]
name: [name provided as argument `name`]

For each combination of tag values of different tag labels for which a metric to aggregate exists, one metric will be added. Consider the previous example and let's assume there exist stream-thread-1 and stream-thread-2. Stream-thread-1 has tasks 0_1 , 1_0, 1_1 and 1_2, and stream-thread-2 has task 0_0 and 0_2. Furthermore, let's assume that only tasks 0_0, 0_1, and 0_2 contain the metric (e.g. have a RocksDB state store). Then three metrics that record aggregations are added:

- stream-thread-1 and task 0_1
- stream-thread-2 and task 0_0
- stream-thread-2 and task 0_2

Users can specify the recording level for the aggregations. The user specified recording level will not change the recording level of the metrics to aggregate. If the recording level for the application is set to INFO, a DEBUG-level metric that should be aggregated will not record values even if the metrics that records the aggregation is on recording level INFO.

The function that is used for the aggregation will be specified by argument `aggregationFunction` and the initial value of the aggregate will be specified by `initialAggregateSupplier`. The aggregation function will take the current aggregate as first argument and the value to add to the aggregate as second argument.

A metrics aggregation can only be added when the Kafka Streams client is in state `CREATED`.

The following code example shows how to add a metrics aggregation for the state-store-level metric `size-all-mem-tables` by stream threads and tasks:

```
kafkaStreams.addMetricsAggregation(
        new MetricsAggregationConfig{
                "size-all-mem-tables-aggregation",
                "records the aggregation of the sizes of all mem-tables grouped by stream threads and task",
                Arrays.asList("thread-id", "task-id"),
                RecordingLevel.INFO,
                () -> BigInteger.valueOf(0),
                BigInteger::add
        ),
        "stream-state-metrics",
        "size-all-mem-tables"
);
```

Compatibility, Deprecation, and Migration Plan

The proposal is backward-compatible because it only adds a new method and does not change any existing methods.

No methods need to be deprecated and no migration plan is required.

Rejected Alternatives

- **Add the method to the `StreamsMetrics` interface:** Adding the method to the `StreamsMetrics` interface would imply that the method could be called from everywhere within a processor that has access to an implementation of the `StreamsMetrics` interface. That would require more concurrency control than adding the method to the `KafkaStreams` class. In our opinion, the value of adding the method to the `StreamsMetrics` interface does not outweigh the additional costs of concurrency control mechanisms, thus we rejected this approach.