KIP-187 - Add cumulative count metric for all Kafka rate metrics

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: *Accepted (1.0.0)*

Discussion thread: *here*

JIRA: *KAFKA-5738*

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

Motivation

Kafka metrics currently group multiple metrics into an MBean with each metric as an Attribute. This is different from other metrics like yammer, where each metric has its own MBean with multiple attributes. For rate metrics, Kafka metrics provides a single attribute that is the rate within a window while yammer metrics has multiple attributes including OneMinuteRate, FiveMinuteRate etc. and cumulative count. Cumulative counts work correctly regardless of metric reporting rate or missing samples and (crucially) do not rely on knowing the window size of the sensor. It makes downstream processing simpler, more accurate, and more flexible. External reporting systems can easily calculate rates and sliding windows based on the count.

For backward compatibility, we need to retain the existing mapping of Kafka metrics to Attributes. To improve processing of rate metrics, this KIP proposes to add an additional attribute for cumulative count corresponding to each rate metric.

Public Interfaces

Every rate metric will be created with two attributes, xxx-rate and xxx-total. xxx-rate will be rate within a window as they are today. xxx-total will be the cumulative count that may be used to calculate rates. xxx-rate is being retained for backward compatibility.

New metrics added by this KIP (right hand column of each table show the new metric, first column is the corresponding rate metric):

Common monitoring metrics for producer/consumer/connect/streams:

connection-creation-rate	*connection-creation-total*
connection-close-rate	*connection-close-total*
network-io-rate	*network-io-total*
outgoing-byte-rate	*outgoing-byte-total*
request-rate	*request-total*
incoming-byte-rate	*incoming-byte-total*
response-rate	*response-total*
select-rate	*select-total*
io-wait-ratio	*io-wait-time-total*
io-ratio	*io-time-total*

Common per-broker metrics for producer/consumer/connect/streams:

outgoing-byte-rate	*outgoing-byte-total*

request-rate	*request-total*
incoming-byte-rate	*incoming-byte-total*
response-rate	*response-total*

Producer Monitoring

bufferpool-wait-time	*bufferpool-wait-time-total*
record-send-rate	*record-send-total*
record-retry-rate	*record-retry-total*
record-error-rate	*record-error-total*
byte-rate	*byte-total*
compression-rate	*compression-total*

Consumer Monitoring

Consumer Group Metrics

commit-rate	*commit-total*
heartbeat-rate	*heartbeat-total*
join-rate	*join-total*
sync-rate	*sync-total*

Consumer Fetch Metrics

bytes-consumed-rate	*bytes-consumed-total*
fetch-rate	*fetch-total*
records-consumed-rate	*records-consumed-total*

Streams Monitoring

Thread metrics

commit-rate	*commit-total*
poll-rate	*poll-total*
process-rate	*process-total*
punctuate-rate	*punctuate-total*
task-created-rate	*task-created-total*
task-closed-rate	*task-closed-total*
skipped-records-rate	*skipped-records-total*

Task Metrics

commit-rate	commit-total

Process Node Metrics

[process | punctuate | create | destroy]-rate	*[process* | *punctuate* | *create* | *destroy]-total*
forward-rate	*forward-total*

State Store Metrics

[put | put-if-absent | get | delete | put-all | all | range | flush | restore]-rate	*[put | put-if-absent | get | delete | put-all | all | range | flush | restore]-total*

Proposed Changes

A new `Stat` named `org.apache.kafka.common.metrics.stats.Meter` will be added which will be a `CompoundStat` similar to `Percentiles`, with two metrics: rate and total (e.g. `request-rate` and `request-total`). For consistency, metrics like `io-wait-ratio` which specify time ratio will also have two attributes (e.g. `io-wait-ratio` and `io-wait-time-total`).

Example:

Creating Rate Metrics

```
rateMetricName = metrics.metricName("request-rate", metricGrpName, "The average number of requests sent per
second.", metricTags);
totalMetricName = metrics.metricName("request-total", metricGrpName, "The total number of requests sent.",
metricTags);
this.bytesSent.add(new Meter(new Count(), rateMetricName, totalMetricName));
```

`Meter` would be a `CompoundStat` that returns both total and the existing rate metrics. This is already handled correctly to support `Percentiles`. The two metrics appear as two `Attributes`.

Metrics from Meter as a CompoundStat

```
 @Override
 public List<NamedMeasurable> stats() {
    List<NamedMeasurable> stats = new ArrayList<NamedMeasurable>(2);
    stats.add(new NamedMeasurable(totalMetricName, total));
    stats.add(new NamedMeasurable(rateMetricName, rate));
    return stats;
}
```

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*

Existing attributes for rate metrics are being retained to avoid any impact on existing users.

- *If we are changing behavior how will we phase out the older behavior?*

There are no plans to remove the existing metrics attributes that expose rates. When viewing JMX metrics through jconsole, it is useful to view the rate attributes.

Rejected Alternatives

Map each Kafka Metric to an MBean and provide attributes for count, rates etc.

This is similar to yammer metrics, but would be a breaking change. The proposed change provides the same data without breaking backward compatibility. It is slightly messier to add more attributes to a metric with the current format in future, but since cumulative count is flexible enough for external tools to provide different views, this is a reasonable compromise to retain compatibility.