KIP-444: Augment metrics for Kafka Streams

Status

Current state: *Accepted*

Discussion thread: link

JIRA:

key	summary	type	created	updated	due	assignee	reporter	priority	status	resolution

JQL and issue key arguments for this macro require at least one Jira application link to be configured

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

Motivation

From collected community feedbacks on Streams operational experience, we are lacking several key metrics for the following tasks:

- **Monitoring**: users would build UI consoles that demonstrate some key metrics 24-7. Only the most critical high-level health and status metrics would be consoled here (e.g. instance state, thread state). Alert triggers will usually be set on some threshold for these metrics (e.g. skip-record > 0, consume-latency > 10k, etc).
- **Information:** this can be considered under the monitoring category as well but with different categories of metrics. Such information could include, for example, kafka version, application version (same appId may evolve over time), num.tasks hosted on instance, num.partitions subscribed on clients, etc. These are mostly static gauges that Users normally would not built console for them, but may commonly query these metrics values in operational tasks.
- **Debugging**: when some issues were discovered, users would need to look at finer grained metrics. In other words, they are less frequently queried than the second categories.
- **Programmables:** some time users would like to programmatically query the metrics, either inside their JVMs or as side-cars collocated with additional reporting logic on top of that.

For the above purposes, we want to 1) cleanup **Streams Built-in Metrics** to have more out-of-the-box useful metrics while trimming those non-useful ones because the current APIs are not very intuitive from its naming to reason about its semantics (this proposal includes removing some redundant APIs as well as refactoring the parent-child metrics relationships, details below), and 2) improve APIs for **User Customized Metrics** that let users register them own metrics, based on its "operationName / scopeName / entityName" notions; we would simplify this interface for user's needs, plus making sure it functions correctly.

Public Interfaces

StreamsMetrics Interface

First for user customizable metrics APIs, here's the proposed changes on `StreamsMetrics` interface:

```
// deprecated APIs: use {@link Sensor#record(double)} directly instead.

@Deprecated
```

```
void recordLatency(final Sensor sensor, final long startNs, final long endNs);

@Deprecated
void recordThroughput(final Sensor sensor, void final long value);

@Deprecated
* @deprecated since 2.5. Use {@link addLatencyRateTotalSensor} instead
Sensor addLatencyAndThroughputSensor(... )

@Deprecated
* @deprecated since 2.5. Use {@link addRateTotalSensor} instead
Sensor addThroughputSensor(... )

// updated APIs javadocs

  /*
   * Add a latency, rate and total sensor for a specific operation, which will include the following metrics:
   * <ol>
   * <li>average latency</li>
   * <li>max latency</li>
   * <li>invocation rate (num.operations / time unit)</li>
   * <li>total invocation count</li>
   * </ol>
   * Whenever a user record this sensor via {@link Sensor#record(double)} etc,
   * it will be counted as one invocation of the operation, and hence the rate / count metrics will be updated
accordingly;
   * and the recorded latency value will be used to update the average / max latency as well. The time unit of
the latency can be defined
   * by the user.
   *
   * Note that you can add more metrics to this sensor after created it, which can then be updated upon {@link
Sensor#record(double)} calls;
   * but additional user-customized metrics will not be managed by {@link StreamsMetrics}.
   *
   * @param scopeName          name of the scope, which will be used as part of the metrics type, e.g.: "stream-
[scope]-metrics".
   * @param entityName         name of the entity, which will be used as part of the metric tags, e.g.: "[scope]
-id" = "[entity]".
   * @param operationName      name of the operation, which will be used as the name of the metric, e.g.:
"[operation]-latency-avg".
   * @param recordingLevel     the recording level (e.g., INFO or DEBUG) for this sensor.
   * @param tags               additional tags of the sensor
   * @return The added sensor.
   */
  Sensor addLatencyRateTotalSensor(final String scopeName,
                                   final String entityName,
                                   final String operationName,
                                   final Sensor.RecordingLevel recordingLevel,
                                   final String... tags);

  /*
   * Add a rate and a total sensor for a specific operation, which will include the following metrics:
   * <ol>
   * <li>invocation rate (num.operations / time unit)</li>
   * <li>total invocation count</li>
   * </ol>
   * Whenever a user record this sensor via {@link Sensor#record(double)} etc,
   * it will be counted as one invocation of the operation, and hence the rate / count metrics will be updated
accordingly.
   *
   * Note that you can add more metrics to this sensor after created it, which can then be updated upon {@link
Sensor#record(double)} calls;
   * but additional user-customized metrics will not be managed by {@link StreamsMetrics}.
   *
   * @param scopeName          name of the scope, which will be used as part of the metrics type, e.g.: "stream-
[scope]-metrics".
   * @param entityName         name of the entity, which will be used as part of the metric tags, e.g.: "[scope]
-id" = "[entity]".
   * @param operationName      name of the operation, which will be used as the name of the metric, e.g.:
"[operation]-latency-avg".
   * @param recordingLevel     the recording level (e.g., INFO or DEBUG) for this sensor.
```

```
   * @param tags               additional tags of the sensor
   * @return The added sensor.
   */
  Sensor addRateTotalSensor(final String scopeName,
                            final String entityName,
                            final String operationName,
                            final Sensor.RecordingLevel recordingLevel,
                            final String... tags);
```

Users can create a sensor via either `addLatencyAndRateSensor` or `addRateSensor`, which will be pre-registered with the latency / rate metrics already; more metrics can then be added to the returned sensors in addition to the pre-registered ones. When recording a value to the sensor, users should just use `Sensor#record()` directly on the sensor itself.

Streams build-in Metrics

And for Streams built-in metrics, we will clean them up by 1) adding a few instance-level metrics, 2) removing a few non-useful / overlapped-in-function metrics, 3) changing some metrics' recording level as well. Note the symbols tags in the tables below (the descriptions of the metrics are omitted since their semantics are all straight-forward based on the names of "rate, total, max, avg, static gauge" etc).

$: newly added

! : breaking changes

* : the sensors are created lazily

() : parent sensor

	LEVEL 0	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 3	LEVEL 3
	Per-Client	**Per-Thread**	**Per-Task**	**Per-Processor-Node**	**Per-State-Store**	**Per-Cache**
TAGS	***type=stream-metrics, client-id=[client-id]***	***type=stream-thread-metrics, thread-id=[threadId]*** ***(! tag name changed)***	***type=stream-task-metrics, thread-id=[threadId], task-id=[taskId]*** ***(! tag name changed)***	***type=stream-processor-node-metrics,thread-id=[threadId],task-id=[taskId],processor-node-id=[processorNodeId]*** ***(! tag name changed)***	***stream-state-metrics,thread-id=[threadId],task-id=[taskId], [storeType]-state-id=[storeName]*** ***(! tag name changed)***	***type=stream-record-cache-metrics, thread-id=[threadId], task-id=[taskId], record-cache-id=[storeName]*** ***(! tag name changed)***
version | commit-id (static gauge)	INFO ($)					
application-id (static gauge)	INFO ($)					
topology-description (static gauge)	INFO ($)					
state (dynamic gauge)	INFO ($)					
alive-stream-threads (dynamic gauge)	INFO ($)					
process-latency (avg | max)		INFO	DEBUG	(! removed for now)		

process (rate | total)		INFO	DEBUG ( ) on source-nodes only	DEBUG on source-nodes only		
punctuate-latency (avg | max)		INFO	DEBUG			
punctuate (rate | total)		INFO	DEBUG			
commit-latency (avg | max)		INFO	DEBUG			
commit (rate | total)		INFO	DEBUG			
poll-latency (avg | max)		INFO				
poll (rate | total)		INFO				
process | punctuate | commit | poll-ratio (dynamic gauge)		INFO				
task-created | closed (rate | total)		INFO				
poll-records (avg | max)		INFO				
process-records (avg | max)		INFO				
active-process-ratio (dynamic gauge)			INFO ($) (percentage of time the hosting thread is spending with this active task)			
standby-process-ratio (dynamic gauge)			INFO ($) (percentage of time the hosting thread is spending with this standby task)			
dropped-records (rate | total)			INFO ($) (number of records dropped within this task due to all kinds of scenarios)			
active-buffer-count (dynamic gauge)			DEBUG			
enforced-processing (rate | total)			DEBUG			
record-lateness (avg | max)			DEBUG			

suppression-emit (rate | total)				DEBUG * (suppress processor only)		
suppression-buffer-size (avg | max)					DEBUG * (suppression buffer only)	
suppression-buffer-count (avg | max)					DEBUG * (suppression buffer only)	
(put | put-if-absent .. | get)-latency (avg | max)					DEBUG * (excluding suppression buffer) (! name changed)	
(put | put-if-absent .. | get) (rate)					DEBUG * (excluding suppression buffer) (! name changed)	
hit-ratio (avg | min | max)						DEBUG (! name changed)

A few philosophies behind this cleanup:

1. We will remove most of the parent sensors with `level-tag=all` except one case. The main idea is to let users to do rolling-ups themselves only if necessary so that we can save necessary metrics value aggregations. For these exceptional case, one parent-child sensor relationship is maintained because it is a bit tricky for users to do the rolling up correctly.
2. We will keep all LEVEL-0 (instance) and LEVEL-1 (thread) sensors as INFO, and most of lower level sensors as DEBUG reporting level. They only exception is active/standby-task-process and dropped / skipp-records
    a. active/standby-task-process indicate the percentage that the current hosting thread is spending on processing them.
    b. dropped/skipped records indicate unexpected errors during processing and hence need to be paid attention by users. Their semantics though are a bit different: skipped records are those skipped at the very beginning of the process and hence not even traverse the topology at all; dropped-records are those dropped in the middle of the topology, and are not necessarily corresponding to a 1-1 mapping to the source records since one source records may be transformed to multiple intermediate records which are then dropped later.
3. For some metrics that are only useful for a specific type of entities, like "suppression-emit", we will only create the sensors lazily in order to save unnecessary costs for metrics reporters to iterate those empty sensors.
4. Some of the lower level metrics like "forward-rate" and "destroy-rate" are removed directly since they are overlapping with other existing metrics already. Here are a list of removed / replaced sensors:

```
late-records-drop: INFO at processor node level, replaced by INFO task-level "dropped-records".

skipped-records: INFO at thread and processor node level, replaced by INFO task-level "dropped-records".

expired-window-record-drop: DEBUG at state store level, replaced by INFO task-level "dropped-records".

forward-rate: DEBUG at processor-node level, replaced by DEBUG processor node level "process-rate".

destroy-rate: DEBUG at processor-node level, covered by INFO thread-level "task-closed-rate".

create-rate: DEBUG at processor-node level, covered by INFO thread-level "task-create-rate".
```

Release History

2.4

- client-level metrics:
    - added:
        - `version`
        - `commit-id`
        - `application-id`
        - `topology-description`

        - `state`

2.5

- thread-level metrics:
    - refactored:
        - `process-latency (avg | max)`
        - `process (rate | total)`
        - `punctuate-latency (avg | max)`
        - `punctuate (rate | total)`
        - `commit-latency (avg | max)`
        - `commit (rate | total)`
        - `poll-latency (avg | max)`
        - `poll (rate | total)`
        - `task-created | closed (rate | total)`
    - removed:
        - `skipped-records`
- task-level metrics:
    - refactored:
        - `process-latency (avg | max)`
        - `process (rate | total)`
        - `punctuate-latency (avg | max)`
        - `punctuate (rate | total)`
        - `commit-latency (avg | max)`
        - `commit (rate | total)`
        - `enforced-processing (rate | total)`
        - `record-lateness (avg | max)`
    - added:
        - `dropped-records (rate | total)`
    - removed:
        - `expired-window-record-drop`
- processor-node-level:
    - refactored:
        - `process (rate | total)`
        - `suppression-emit (rate | total)`
    - removed:
        - `process-latency (avg | max)`
        - `late-records-drop`
        - `skipped-records`
        - `forward-rate`
        - `destroy-rate`
        - `create-rate`
- state-store-level:
    - refactored:
        - `suppression-buffer-size (avg | max)`
        - `suppression-buffer-count (avg | max)`
        - `(put | put-if-absent .. | get)-latency (avg | max)`
        - `(put | put-if-absent .. | get) (rate)`
    - removed:
        - `expired-window-record-drop`
- cache-level:
    - refactored:
        - `hit-ratio (avg | min | max)`

2.6

- client-level metrics:
    - added:
        - `alive-stream-threads`
- thread-level metrics:
    - added:
        - `process | punctuate | commit | poll-ratio`
        - `poll-records (avg | max)`
        - `process-records (avg | max)`
- task-level metrics:
    - added:
        - `active-process-ratio`

Proposed Changes

As above.

Compatibility, Deprecation, and Migration Plan

The Streams build-in metrics changes contains metrics name changes as well as tag changes (mainly because we added LEVEL-0 instance metrics in addition to the original top-level LEVEL-1 thread metrics), which will break existing users whose monitoring systems is built on the old metric / tag names.

So in order to allow users having a grace period of changing their corresponding monitoring / alerting eco-systems, I'd propose to add a config

```
"built.in.metrics.version":


type: Enum
values: {"0.10.0-2.4", "latest"}
default: "latest"
```

When users override it to "0.10.0-2.4", then the old metrics names / tags will still be used.

Rejected Alternatives

None.