KIP-977: Partition-Level Throughput Metrics

Status

Current state: *Approved*

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

Currently, the MessagesInPerSec metrics of Kafka is a topic-level metrics. It is emitted without the partition info, see ReplicaManager. This metric provides vital information regarding topic throughput and is usually leveraged in topic quota/partition management. However, if the traffic of a topic is not balanced, the overall throughput does not provide any insight into this imbalance nature. This is particularly problematic for keyed topics as messages may be produced with hotspots in the spectrum and overload certain partitions only. In this case:

1. Expansion of the topic partition won't solve the scalability problem on the broker side, as messages may still overload certain partitions.
2. We lose the opportunity to identify the imbalance partition and rebalance/alert the producer accordingly.
3. In most cases today, a Kafka partition is still closely bound to a consumer instance. Combined with an imbalance topic, it may easily overload the consumer instance, and we don't have the opportunity to identify and act on this situation beforehand.

Prior Work

There are many attempts on this topic and some even reached the PR stage(PR#13571). But unfortunately, none of them is actually merged.

- [KAFKA-14907] Add the traffic metric of the partition dimension in BrokerTopicStats
- [KIP-583] Add tag "partition" to BrokerTopicMetrics so as to observe the partition metrics on the same broker

The major reason why the PRs are not merged or the KIPs are left in discussion is that the metrics may fan out too much. Consider the typical situation that currently exists in a production Kafka cluster(here we use Uber as an example) today: topics have at least 4-8 partitions and in the most extreme cases, a topic may have 256 partitions. This means the fan-out rate is at least 4-8 times and can go as high as 256x. Reporting throughput with partition info brings a higher burden to the metrics emission and collection infrastructure if we opt in blindly. To control this behavior, we would like to propose the following changes.

Public Interfaces

- This change will introduce a new dynamic configuration option(details below): **`metrics.verbosity`**
- This change also involves a behavior change of the following metrics: **`MessagesInPerSec`**, **`BytesInPerSec`**, and **`BytesOutPerSec`**

Proposed Changes

Part 1: Verbosity Level Control

`metrics.verbosity` will be a new dynamic config introduced to control the verbosity(fan-out rate) of the metrics. It's a config with JSON format specifying the condition controlling fan-out of the metrics. If the value **`high`** is set for the **`level`** key of the configured JSON(see below for example values), high fan-out tags(e.g. **`partition`**)will be added to metrics specified by the **`name`** filter and will apply to all the topics that meet the conditions in the **`filters`** section. In the **`low`** settings, these tags will be assigned with an empty value. We elected to make it central so that this implementation can be generalized in the future either into a library, or allow other means for centralized control.

To further extend the control for future cases, the metrics name and topic can be controlled in this single configuration too(see details in the JSON Config Format section).

Type	String(regex expression)
Default	`"[]"`

Valid Values / Examples	A string representing a JSON array, details below.
	Example(actual string will be skipped):
	`[{`
	`  "level": "high",`
	`  "name": "Bytes*",`
	`  "filters" = [{topics: ["avro-topic-car", "avro-topic-bus", "avro-topic-plane"]}]`
	`}]`
	
	The above config means the **`BytesInPerSec`** and **`BytesOutPerSec`** will be having partition-level metrics for the 3 topics of avro-topic-car, avro-topic-bus, and avro-topic-plane

Importance	medium
Update Mode	Dynamic Read & Write. NOTE: To prevent unexpected behavior, dynamic configuration will not persist and the permanent value will be used once the broker is restarted.

JSON Config Format

The JSON array mentioned above should have JSON objects configuring the corresponding metrics verbosity details at each level. Here are the valid keys in the JSON object, and their corresponding value details.

Key Name	Default Value	Valid Values	Restrictions	Example
level	`low`	`low, (medium), high` NOTE: We don't have any medium-level metrics as of now. We leave this option open for future extension.	JSON entries are validated with no particular order, or they are order-irrelevant. The entries can be duplicated, but are required to have no conflict between each other. If there's a conflict between values configured with different metrics level settings, the one that appears first in the sequence `[high, medium, low]` will take precedence. For example, a topic configured in both low and high will be treated with the configuration values in **`high`**. NOTE: It's okay to have multiple entries of `level:` **`high`**, these entries may well apply to different metric names or dimension filters.	`"level": "high"`
names	`.*`	Pattern corresponding to the names of the metric series in scope		`"names": "Bytes*"`

filters	`[]` (empty JSON array, applies to nothing)	`[{"topicPattern": "<topic pattern>"}]`
		
		or
		
		`[{"topics": [<a list of topics>](string array)}]`
			• The filters in the array are of AND relationship. For example, if there are two filters **`A=Topic:a,b`** and **`B=Topic:a,c,d`** then the rule will be applied to **`a`** only.
			• topicPattern and topics should not be used at the same time
			• For now, we only allow the filter to take topics, but it can be easily extended.
				`[{"topicPattern": "avro-topics-*"}]`
				
				or
				
				`[{"topics": ["avro-topic-car", "avro-topic-bus", "avro-topic-plane"]}]`

All regex patterns mentioned above should be `java.util.regex`-compatible, see more details in Java util's regex engine.

Working with the Configuration via CLI

- Querying existing config
  - Command: **`bin/kafka-configs.sh --bootstrap-server localhost:9092 --describe --entity-type brokers`**

- Example Result 1(default, if not set previously): **`metrics.verbosity=[]`**
  Expected behavior: verbosity level low is applied to all metric series and no high fan-out tags addedMessagesInPerSec
- Example Result 2: **`metrics.verbosity=[{"level": "low", names: ".*"}]`**
  Expected behavior: verbosity level low is applied to all metric series and no high fan-out tags added
- Example Result 3:
  ```
  metrics.verbosity=[{
      "level": "high",
      "name": "Bytes*",
      "filters" = [{topics: ["avro-topic-car", "avro-topic-bus", "avro-topic-plane"]}]
  }]
  ```
  Expected behavior: `BytesInPerSec`, `BytesOutPerSec`, `BytesRejectedPerSec` will include partition-level values for the 3 listed topics.

- Setting new config
  - Example Command 1(add config): **`bin/kafka-configs.sh --bootstrap-server localhost:9092 --alter --add-config metrics.verbosity=[{"level": "high", names: ".*", filters=[{"topics": ["avro-*"]}]}]`**
    Expected behavior after the change: partition-level values will be added to all the 10 metrics series for all the topics matching name regex: `` `avro-*` ``
  - Example Command 1(alter config): **`bin/kafka-configs.sh --bootstrap-server localhost:9092 --alter --alter-config metrics.verbosity=[{"level": "low", names: ".*"}]`**
    Expected behavior after the change: partition-level values will be removed for all the metrics for all the topics.

Performance Consideration

Empty metrics filtering logic here will ensure empty tags are not included in the final metrics sent out to the exporter. This behavior ensures the default setting incurs no additional performance cost but it could be an issue for the dynamical addition/removal of a tag. Certain metrics exporters(e.g. Prometheus) combine labels(tags) into the name. This means different label sets may map to different metrics and may fail the validation on the metrics exporters' side. To solve this issue, the filtering logic can be removed. By changing this behavior, empty tags will end up in the metrics, but that does not increase the fan-out rate and won't bring a big performance hit to the brokers under default mode.

For the purpose of this KIP, there are only 3 metrics that will have a high fan-out tag. However, if many metrics start to add high fan-out tags, switching the level could lead to a blast of metrics series and cause performance degradation. To resolve this issue, we propose to have additional control over which metrics should allow high verbosity. This will not be the concern of this KIP and shall be resolved in a follow-up effort.

Part 2: New Metrics with Attached Partition Tag for Metrics if Allowed

Currently, to record the messages-in-rate, the API is invoked in this way:

```
brokerTopicStats
    .topicStats(topicPartition.topic)
    .messagesInRate.mark(numAppendedMessages)
```

After the change, it should be

```
brokerTopicStats
    .topicStats(topicPartition)
    .messagesInRate.mark(numAppendedMessages)
```

Inside the `BrokerTopicStats`, the topic-level meter will be kept the same way. In addition to that, there will be an additional map of meters for partition-level metrics.

```
private val paritionStats =
    new Pool[TopicPartition,
             BrokerTopicPartitionMetrics](Some(valueFactory))
```

The new class of `BrokerTopicPartitionMetrics` will hold the following partition-level metrics. We are keeping it simple here to reduce the overhead associated with this change. Currently, only the metrics closely related to producers and consumers are listed here. We will always have the option to add more metrics, e.g. replication, if needed.

Metrics Name	Meaning
`MessagesInPerSec`	Messages entered the partition, per second
`BytesInPerSec`	Bytes entered the partition, per second
`BytesOutPerSec`	Bytes retrieved from the partition, per second
`BytesRejectedPerSec`	Bytes exceeding max message size in a partition, per second
`TotalProduceRequestsPerSec`	Produce request count for a partition, per second
`TotalFetchRequestsPerSec`	Fetch request count for a partition, per second

`FailedProduceRequestsPerSec`	Failed to produce request count for a partition, per second
`FailedFetchRequestsPerSec`	Failed to fetch request count for a partition, per second
`FetchMessageConversionsPerSec`	Broker side conversions(de-compressions) for a partition, per second
`ProduceMessageConversionsPerSec`	Broker side conversions(compressions) for a partition, per second

Compatibility, Deprecation, and Migration Plan

Version Upgrade/Downgrade

If the flag is not set, its default value will be `default`. This means in the event of a broker version upgrade or downgrade, the behavior will not change.

Exporter Compatibility

Since we do not dynamically adjust the tag list, we don't expect incompatibility with existing metrics exporters. This will also be verified in the integration tests.

Test Plan

Unit Test

1. The partition tag should be included for the QPS and BPS metrics when invoking the `brokerTopicStats`
2. The partition tag should be included in the output metrics if high verbosity is used
3. The partition tag should be empty if default verbosity is used

Integration Test

The following 3 tests will be done for common metrics collectors: JMX, Prometheus, and OpenTelemetry.

1. The partition tag can be observed from metrics if high verbosity is used
2. The partition tag should result in an empty string or be filtered out by the metrics collector if default verbosity is used
3. Dynamically setting the verbosity can result in the behavior defined in the above tests

Performance Test

To better predict the performance penalty introduced by this KIP, we plan to run the following test cases in the following 8 setups.

1. No/low traffic broker, low partition count(~hundreds of topic partitions), low metrics emission interval
2. No/low traffic broker, low partition count(~hundreds of topic partitions), high metrics emission interval
3. No/low traffic broker, high partition count(~thousands of topic partitions), low metrics emission interval
4. No/low traffic broker, high partition count(~thousands of topic partitions), high metrics emission interval
5. High traffic broker, low partition count(~hundreds of topic partitions), low metrics emission interval
6. High traffic broker, low partition count(~hundreds of topic partitions), high metrics emission interval
7. High traffic broker, high partition count(~thousands of topic partitions), low metrics emission interval
8. High traffic broker, high partition count(~thousands of topic partitions), high metrics emission interval

Note: all setups will be tested for both code-start enabling and dynamic enabling.

Test Case 1: Empty Tag(No Topics or Metrics Names)

The purpose of this test is to ensure this KIP won't bring performance drawbacks if the feature is not enabled.

Expected result: minimal if not no performance(CPU, memory, network, disk) hit to other broker functionality, under all 8 scenarios

Test Case 2: Limited Topics & Limited Metrics

The purpose of this test is to ensure when the feature is enabled for debugging purposes, it won't destabilize the broker by too much. It will also help to answer:

- How many topic partitions, if having this feature enabled, would not overload the broker by too much?
- What will the worst performance result if this feature is enabled under the most severe case?
    - Most severe cases: high traffic broker, high partition count, low metrics emission interval, high percentage of topic-partitions with this feature enabled

Search dimensions

- Topic-partitions: 10% - 100% of topic-partitions of the broker, 10% step
- Metrics: 1-3 of the names listed in the Metrics Names

Rejected Alternatives

Using the Existing RecordingLevel Config

The `RecordingLevel` config also controls the verbosity of the metrics emitted by various Kafka components. However, we decided not to build on top of this config for the following reasons:

1. This config currently has only one string value and can only control metrics emission at the Sensor level. This does not meet our need for detailed fan-out rate control.
2. This config has to be applied to all new configs if we want to use it to control the recording level. This means we first need to create a separate sensor, which introduces unnecessary duplications.
3. This config lacks the dynamic adjustment capability demanded by this KIP.

To further reduce the overall duplication and move towards a unified control. With this KIP, we propose the following change to merge/clean up the usage of `RecordingLevel`:

1. The recording levels can be mapped to the config values defined above. For example, `RecordingLevel.TRACE` can be defined as `{"level":" high", "name":"<trace-level-metrics>"}`
2. The existing metric sensors can be updated to remove the RecordingLevel and accept the control of **`metrics.verbosity`**. Note: This will bring the config to the client side, it's currently a server-side-only config.

Topic Allow Listing for Partition-Level Reporting

To control the overall fan-out of the added tag, we can add an allowlist for the topics that have this feature enabled. We would tentatively name it to **`metrics.partition.level.reporting.topics`**. This added configuration will contain topics that will have partition-level metrics reported.

Type	list
Default	`[] (empty list)`
Valid Values	`[topic1, topic2,...]`
Importance	low
Update Mode	1. The list is used by `brokerTopicStats` to determine if a topic should have partition-level metrics emitted or not. In the event that a topic gets deleted, the topic and the topic partitions related to this topic in `brokerTopicStats` will no longer be used. This means not updating the config will not cause any issue 2. If a non-existent topic is added to the config, it will not cause any issue either since the meter objects are lazy-created in `brokerTopicStats`. A non-existent topic does not have traffic and thus will not get the topic or topic-partition meters created 3. If a pre-existed topic was added to the list, deleted, and re-created again, the partition-level metrics will be emitted. In this case, more metrics will be produced. To solve this issue, we can hook the config to the deletion flow.

Reasons why this is not selected:

1. Topic management operations(creation, deletion) are common. Unless this new dynamic config is completely hooked into the topic life cycle management, it's very likely to fall behind the actual topic existence status. This may cause a burst of metrics traffic if a topic is re-created.
2. If other highly-sought tags are going to be added, or even just add the partition tag to similar metrics, we will have to add the read logic of this dynamic config. This flag will then end up everywhere and become difficult to maintain or extend.