489: Kafka Consumer Record Latency Metric

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

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: *Under Discussion*

Discussion thread: *here*

JIRA: *here*

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

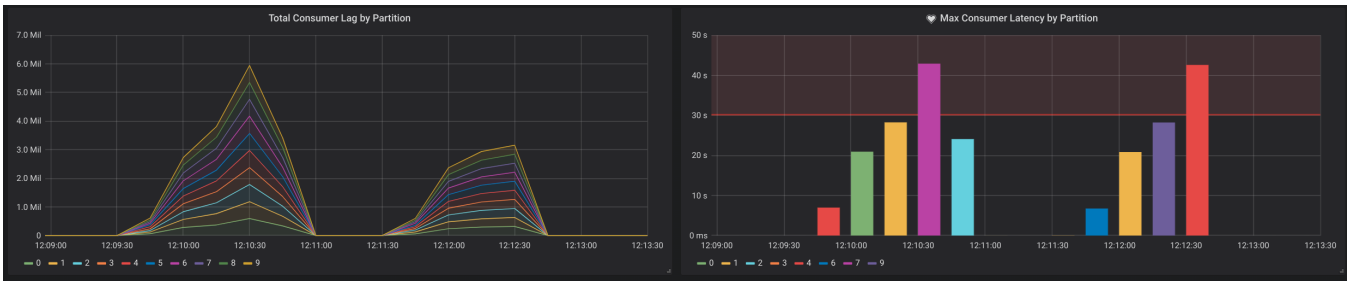
Motivation

Consumer lag is a useful metric to monitor how many records are queued to be processed. We can look at individual lag per partition or we may aggregate metrics. For example, we may want to monitor what the maximum lag of any particular partition in our consumer subscription so we can identify hot partitions, caused by an insufficient producing partitioning strategy. We may want to monitor a sum of lag across all partitions so we have a sense as to our total backlog of messages to consume. Lag in offsets is useful when you have a good understanding of your messages and processing characteristics, but it doesn't tell us how far behind *in time* we are. This is known as wait time in queueing theory, or more informally it's referred to as latency.

The latency of a message can be defined as the difference between when that message was first produced to when the message is received by a consumer. The latency of records in a partition correlates with lag, but a larger lag doesn't necessarily mean a larger latency. For example, a topic consumed by two separate application consumer groups A and B may have similar lag, but different latency per partition. Application A is a consumer which performs CPU intensive business logic on each message it receives. It's distributed across many consumer group members to handle the load quickly enough, but since its processing time is slower, it takes longer to process each message per partition. Meanwhile, Application B is a consumer which performs a simple ETL operation to land streaming data in another system, such as HDFS. It may have similar lag to Application A, but because it has a faster processing time its latency per partition is significantly less.

If the Kafka Consumer reported a latency metric it would be easier to build Service Level Agreements (SLAs) based on non-functional requirements of the streaming system. For example, the system must never have a latency of greater than 10 minutes. This SLA could be used in monitoring alerts or as input to automatic scaling solutions.

As an example, below is a screenshot of a Grafana dashboard with a panel for measuring the sum of consumer lag per partition on the LHS and a panel for measuring the top partition with the highest latency on the RHS. The latency panel also defines an alert that is triggered when latency exceeds 30 seconds over a 5 minute window.



Public Interfaces

Define new latency metrics under the Consumer Fetch Metrics metrics group.

`kafka.consumer:type=consumer-fetch-manager-metrics,client-id="{client-id}"`

Attribute Name	Description
`records-latency-max`	The latency of the partition with the longest latency. Picks the largest partition record-latency-max of all partitions assigned to this client.

`kafka.consumer:type=consumer-fetch-manager-metrics,client-id="{client-id}",topic="{topic}"`

Attribute Name	Description
`records-latency-max`	The latency of the partition with the longest latency. Picks the largest partition record-latency-max of all partitions of a topic assigned to this client.

`kafka.consumer:type=consumer-fetch-manager-metrics,partition="{partition}",topic="{topic}",client-id="{client-id}"`

Attribute Name	Description
`records-latency`	The latest latency of the partition
`records-latency-max`	The max latency of the partition in this sample window
`records-latency-avg`	The average latency of the partition in this sample window

Proposed Changes

Report latency metrics in the Kafka Consumer at the client (max latency) and partition level, similar to how consumer lag is currently reported.

KIP-32 introduced a timestamp field to the Kafka message format. The timestamp could be provided by the user, the KafkaProducer, or the Broker, depending on how the Broker configuration message.timestamp.type is defined. When defined by Kafka the timestamp value uses current wallclock time represented as a unix epoch long in milliseconds returned by the Java Standard Library call to System.getCurrentMillis(). We assume that all application and Broker machines enable active clock synchronization services and that the latency may be calculated by taking the difference of current wall clock time with the timestamp from a fetched ConsumerRecord. When latency is calculated as negative then the metric value will be reported as `NaN`.

UNIX epoch time is always represented as UTC time, and is therefore agnostic of any machine's particular default timezone.

The implementation to calculate latency would be very similar to how records-lag is implemented in `Fetcher.fetchRecords` and the `FetchManagerRegistry`. The latency metrics would be calculated once each poll for all partitions of records returned. The start timestamp will be extracted from the last record of each partition that is returned. To retrieve the current wall clock time it would use either a `LatencyTime` class implementation configured by the user, or the default implementation which returns `System.getCurrentMillis()`. The latency would be calculated by taking the difference of the wall clock time and the provided record start timestamp. The latency would be recorded by the client and partition sensors for the `records-latency` metric.

At the partition level we can provide the latest calculated latency and the max and average latency within the metrics sample window. At the topic and client level we only provide the max latency, which is is the max(record-latency-max) of all partitions assigned to a client for particular topic, or all topics. An average, or some other percentile could also be represented. A sum of partition latencies would not make sense because it's expected that consumers will consume partitions in parallel and not in a serial manner.

> **Using Latency Metric for SLAs**
>
> If a message was produced a long time ago, and a new consumer group has been created, then the latency metrics will have very high values until the consumer group catches up. This is especially true in the context of KIP-405: Kafka Tiered Storage, which allows reading very old messages. Therefore, a consumer application that relies on reading all messages from the past will report a high records-latency for a while.
>
> Using this metric for SLAs should only be done when a consumer group is expected to be continuously consuming (in a steady state), and not for bootstrapping new consumer groups.

Compatibility, Deprecation, and Migration Plan

N/A

Rejected Alternatives

Use a Consumer Interceptor

An alternative solution to implementing this metric in the Kafka Consumer itself would be to implement it in a Consumer Interceptor and publish it as a library to the general Kafka community. This would require the library author to recreate a metrics implementation because the metrics system of the consumer cannot be reused. The simplest way would be to instantiate a new instance of `org.apache.kafka.common.metrics.Metrics` and configure it in a similar manner as the consumer configures its own internal instance. This would create a duplication of metrics configuration that could be avoided if it were implemented directly in the consumer.

Another reason the interceptor is not ideal is that some Kafka streaming integrations do not allow the user to configure interceptors, or otherwise restrict configuration properties passed to an underlying Kafka consumer. For example, the Spark Kafka integration will explicitly throw an exception when the user attempts to define interceptor.classes in the Kafka Consumer properties.

> *interceptor.classes: Kafka source always read keys and values as byte arrays. It's not safe to use ConsumerInterceptor as it may break the query.*

https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html#kafka-specific-configurations

I have implemented this solution using a Consumer Interceptor and can share it if there's interest.

Kafka Lag Exporter

Kafka Lag Exporter is an open source project that can estimate the latency of Kafka partitions. It lets you track latency (or some aggregate of it) for apps, without having to modify the app in any way. It polls the latest offset of all partitions belonging to consumer groups and topics to be reported and maintains an internal lookup table for each topic partition of offset and timestamp (wallclock time the offset was measured). At the same time, consumer group metadata is polled to retrieve the consumer group id, topic partition, and the last committed offset. With this information the an estimate is predicted by doing the following:

> 1. *Lookup interpolation table for a consumer group partition*
> 2. *Find two points within the table that contain the last consumed offset*
> a. *If there are no two points that contain the last consumed offset then use the first and last points as input to the interpolation formula. This is the extrapolation use case.*
> 3. *Interpolate inside (or extrapolate outside) the two points from the table we picked to predict a timestamp for when the last consumed message was first produced.*
> 4. *Take the difference of the time of the last consumed offset (~ the current time) and the predicted timestamp to find the time lag.*

https://www.lightbend.com/blog/monitor-kafka-consumer-group-latency-with-kafka-lag-exporter

While this tool works reasonably well it has several limiting factors

1. It only works in cases where offsets are committed back to Kafka. If an app or stream processor uses its own offset management then the current offset of a partition cannot be obtained from Kafka.
2. It can only predict an estimate. Accuracy improves with higher fidelity lookup tables (latest offsets looked up more frequently).

Disclosure: I (Sean Glover) am the author of the Kafka Lag Exporter project and the "Monitor Kafka Consumer Group Latency with Kafka Lag Exporter" blog post by Lightbend.