KIP-517: Add consumer metrics to observe user poll behavior

Status

Current state: ACCEPTED

Discussion thread: here

Vote Thread: here

JIRA: Unable to render Jira issues macro, execution error.

Motivation

The KafkaConsumer is a complex client that incorporates different configurations for detecting consumer failure to allow remaining consumers to pick up the partitions of failed consumers. One such configuration is **max.poll.interval.ms** which is defined as:

max.poll.interval.ms	The maximum delay between invocations of poll() when using consumer group management. This places an upper bound on the amount of time that the consumer can be idle before fetching more records. If poll() is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member.	int	300000	[1,...]	medium

Timeout from this configuration typically happens when the application code to process the consumer's fetched records takes too long (longer than `max.poll.interval.ms`). Hitting this timeout will cause the consumer to leave the group/rebalance (if it is **not** a static member as described in KIP-345: Introduce static membership protocol to reduce consumer rebalances). The consumer will end up rejoining the group if processing time was the only issue (and not a static member). This scenario is not ideal as rebalancing will disrupt processing and take additional time.

Additionally, sometimes a long processing time is unavoidable if:

1. Minimum processing time is long to begin with (ex. hit a database which takes 1 second per record minimum)
2. Processing involves talking to a downstream service which **sometimes** causes a spike in processing time (ex. intermittent load issues)

In such cases, the user must fine-tune the configurations to fit their use-case however detection of such events is difficult. The only way to definitely identify this scenario is by searching application logs or the user must record their processing time on their own. The consumer will log an error when max.poll.interval.ms is hit:

```
Member {} sending LeaveGroup request to coordinator {} due to consumer poll timeout has expired. This means the
time between subsequent calls to poll() was longer than the configured max.poll.interval.ms, which typically
implies that the poll loop is spending too much time processing messages. You can address this either by
increasing max.poll.interval.ms or by reducing the maximum size of batches returned in poll() with max.poll.
records.
```

An application owner has the ability to write code to measure processing time, but Kafka operators are out of luck as they must get the application owner to implement such instrumentation. If the application owner does not provide this, then the Kafka operator does not have this data.

It would be beneficial to add metrics to track poll calls as it can be used by both Kafka application owners and operators to:

- Easily identify if/when `max.poll.interval.ms` needs to be changed (and to what value)

- View trends/patterns
- Verify `max.poll.interval.ms` was hit using the max metric when debugging consumption issues (if logs are not available)
- Configure alerts to notify when average/max time is too close to `max.poll.interval.ms`

Example Usage

An application owner reports that their consumers are seeing the `max.poll.interval.ms` timeout error log mentioned in the previous section. The application owner may claim that their application code is fine and that the Kafka infrastructure is broken. The application does not have any instrumentation measuring the processing time of their application which makes it difficult for the Kafka operator to prove otherwise and resolve the issue.

The Kafka operator can only propose to:

- Increase `max.poll.interval.ms`
- Decrease `max.poll.records`

It is not clear what values to use without instrumentation, and it ends up taking a few attempts/deployments to find a value that works.

The metric proposed in this KIP would help in this scenario as it would provide proof that `time-between-poll-max` took longer than the configured `max.poll.interval.ms`. Additionally, it would give an actual value that can be used as a starting point for increasing `max.poll.interval.ms`.

It would also show long-term trends to identify if this incident was:

- Unusual anomaly: Check health/performance of dependencies and consumer host
- Processing time was always close to `max.poll.interval.ms`: Increase `max.poll.interval.ms` or decrease `max.poll.records` to avoid hitting it due to variance

Public Interfaces

We will add the following metrics:

Metric Name	Mbean Name	Description
`time-between-poll-avg`	`kafka.consumer:type=consumer-metrics, client-id=([-.\w]+)`	The average delay between invocations of poll().
`time-between-poll-max`	`kafka.consumer:type=consumer-metrics, client-id=([-.\w]+)`	The max delay between invocations of poll().
`last-poll-seconds-ago`	`kafka.consumer:type=consumer-metrics, client-id=([-.\w]+)`	The number of seconds since the last poll() invocation.
`poll-idle-ratio-avg`	`kafka.consumer:type=consumer-metrics, client-id=([-.\w]+)`	The average fraction of time the consumer's poll() is idle as opposed to waiting for the user code to process records.

Proposed Changes

time-between-poll (avg/max)

As we want this metric to measure the time that the user takes to call `poll()`, we will store a `long lastPollMs` in KafkaConsumer. We will calculate the elapsed time (and update `lastPollMs`) on every call to poll.

On each call to `poll()`, we will calculate the elapsed time since `lastPollMs`, record it in the `Sensor` which has `Avg` & `Max`, and update `lastPollMs` .

last-poll-seconds-ago

The `last-poll-seconds-ago` metric will measure the time since the user last called `poll()` to provide more insight into how/when the user code calls `poll()` .

poll-idle-ratio-avg

There will be an additional metric that measures the idle time of the consumer in waiting for the user to process records returned from poll:

$\textrm{poll-idle-ratio-avg} = \frac{\textrm{time-inside-poll}}{\textrm{total-time}}$

This metric will have a value between 0.0 and 1.0.

A value approaching 1.0 means consumer is idle in poll (ex. waiting for records) while a value approaching 0.0 means the application is busy processing in user code.

A low value (approaching 0.0) could indicate a **potential** issue or performance bottleneck in user code.

Compatibility, Deprecation, and Migration Plan

As this KIP simply adds new metrics, there is no issue regarding compatibility, deprecation, or migration.

Rejected Alternatives

None at the moment.