KIP-211: Revise Expiration Semantics of Consumer Group Offsets

Status

Current state: *Accepted*

Discussion thread: *here*

Vote thread: *here*

JIRA: **KAFKA-4682** - Getting issue details... STATUS

Released: 2.1.0

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

Motivation

The offset of a topic partition within a consumer group expires when the expiration timestamp associated with that partition is reached. This expiration timestamp is usually affected by the broker config `offsets.retention.minutes`, unless user overrides that default and uses a custom retention. This is an overview of how offset expiration works today with respect to different versions of the `OffsetCommit` protocol:

- Version 0: Offsets are stored in ZooKeeper. ZooKeeper based offset storage is not in scope of this KIP.
- Version 1: An optional commit timestamp is associated with each topic partition in the request. The broker's `offsets.retention.minutes` is added to the commit timestamp to determine the expiration timestamp of the partition. In this case, clients cannot override the default retention enforced by the broker.
- Version 2, 3: Similar to Version 1 except there is no explicit commit timestamp for each partition.The `retention_time` field in the request replaces the value of broker's offset retention config in calculating the expiration timestamp.

The following table summarizes this:

Version of `OffsetCommit`	Commit Timestamp	Offset Retention	Expiration Timestamp
0	[ZooKeeper based offset management - out of scope]		
1 - no explicit commit timestamp	Current timestamp	Broker's `offsets.retention.minutes`	Commit Timestamp + Offset Retention
1 - with explicit commit timestamp	Partition-specific `timestamp` in the request	Broker's `offsets.retention.minutes`	
2, 3	Current timestamp	Request's `retention_time`	

For versions 1-3, once the expiration timestamp is reached, the offset is removed from the offset cache (during the next cleanup) regardless of the group state. KAFKA-4682 reports an issue related to this offset expiration, where committed offsets are removed even when there are still active, but rarely committing, consumers in the (`Stable`) group.

In other words, if the corresponding retention period or longer has passed since an active consumer has committed offset for a topic partition, that committed offset will be removed from the consumer group metadata. If then there is a rebalance or the consumer restarts the last committed offset for that topic partition will not be found, and the consumer is forced to start from the start or end of the log (depending on the value of `auto.offset.reset` configuration) leading to potential duplicate consumption or missing records. This situation can be avoided if the offsets are preserved beyond their expiration timestamp if the group is still in a `Stable` state.

There are workarounds to this issue and some of them are described in KAFKA-4682, but they come with their own limitations and drawbacks, as discussed in that JIRA.

Public Interfaces

This is the current `OffsetCommit` protocol:

```
OffsetCommit Request (Version: 3) => group_id group_generation_id
member_id retention_time [topics]
  group_id => STRING
  group_generation_id => INT32
  member_id => STRING
  retention_time => INT64
  topics => topic [partitions]
    topic => STRING
    partitions => partition offset metadata
      partition => INT32
      offset => INT64
      metadata => NULLABLE_STRING


OffsetCommit Response (Version: 3) => throttle_time_ms [responses]
  throttle_time_ms => INT32
  responses => topic [partition_responses]
    topic => STRING
    partition_responses => partition error_code
      partition => INT32
      error_code => INT16
```

The only change made to this protocol is dropping the field `retention_time` from the request. Retention time will be enforced through the broker config `offsets.retention.minutes` in the new version of the protocol and normally takes effect once the group transitions into `Empty` state.

```
OffsetCommit Request (Version: 4) => group_id group_generation_id
member_id [topics]
  group_id => STRING
  group_generation_id => INT32
  member_id => STRING
  topics => topic [partitions]
    topic => STRING
    partitions => partition offset metadata
      partition => INT32
      offset => INT64
      metadata => NULLABLE_STRING

OffsetCommit Response (Version: 4) => throttle_time_ms [responses]
  throttle_time_ms => INT32
  responses => topic [partition_responses]
    topic => STRING
    partition_responses => partition error_code
      partition => INT32
      error_code => INT16
```

Proposed Changes

A more viable solution for KAFKA-4682 can be achieved by changing how group offset expiration works: preserve committed offsets as long as the group is active (has consumers). The expiration timer should start ticking the moment all group members are gone and the group transitions into `Empty` state. This expiration semantics implies that there is no longer a need to enforce individual offset retention times and keep individual expiration timestamps for each topic partition in the group. This is because all committed offsets in the group will expire at the same time.

This proposed change has an impact on the existing offset commit value schema. There is an `expire_timestamp` field in this schema that, as a result of expiring all group offsets at the same time, would become redundant (as it would repeat the same value for each offset in the group).

```
Offset Commit Value Schema (Version: 1) =>
  offset => Long
  metadata => String
  commit_timestamp => Long
  expire_timestamp => Long
```

The proposal is to create a new version of this schema and drop the `expire_timestamp` field:

```
Offset Commit Value Schema (Version: 2) =>
  offset => Long
  metadata => String
  commit_timestamp => Long
```

To make up for the per-offset expiration timestamp we lose in the new version of offset commit value schema, a new field is added in the group metadata value schema that indicates when the group last changed state.

```
Group Metadata Value Schema (Version: 1) =>
  protocol_type => String
  generation => Int
  protocol => String
  leader => String
  members => [member]
    ...
```

```
Group Metadata Value Schema (Version: 2) =>
  protocol_type => String
  generation => Int
  protocol => String
  leader => String
  current_state_timestamp => Long
  members => [member]
    ...
```

The rest of this section explains how these suggested changes help in implementing the new group expiration semantics.

Transitioning to `Empty` State

The expiration time of offsets in a group will be when the group becomes `Empty` plus retention time of `offsets.retention.minutes` (assuming during that time the group does not become active again). Whenever the group transitions to `Empty` state, `current_state_timestamp` resets to the value of current timestamp. Then, during any scheduled offset cleanup task, if "current timestamp" minus `current_state_timestamp` is greater than or equal to broker's `offsets.retention.minutes` for any group, all offsets in that group will be removed and the group will transition to `Dead` state.

Note that consumers may rejoin the group while the group is in `Empty` state. As soon as that happens, the group transitions out of `Empty` state, and that practically disables offset expiration. This is a breakdown of group states and how the offsets expiration works in those states:

- `Stable`: Group offsets will not expire in this state (group state `Empty`)
- `PreparingRebalance`: Group offsets will not expire in this state (group state `Empty`)
- `CompletingRebalance`: Group offsets will not expire in this state (group state `Empty`)
- `Empty`: The field `current_state_timestamp` is set to when group last transitioned to this state. If the group stays in this for `offsets.retention.minutes`, the following offset cleanup scheduled task will remove all offsets in the group (as explained above).
- `Dead`: Group offsets have expired (group deletion); or the group is unloaded from the coordinator cache (coordinator change). No offset expiration action required.

The default retention time for group offsets can be customized through the existing `offsets.retention.minutes` broker configuration. If, in the future, a need arises for enforcing a per group retention configuration, it can be implemented via a separate KIP.

There are also a couple particular cases that need to be addressed with this new semantics:

1. If a group consumer unsubscribes from a topic but continues to consume from other subscribed topics, the offset information of that unsubscribed topic's partitions should be deleted at the appropriate time.
2. Standalone (simple) consumer does not use Kafka's group management mechanism, and requires special handling when it comes to offset expiration.

Unsubscribing from a Topic

If the group state is not `Empty`, when there is a change in subscribed topics of a group consumer, and, as a result, the group stops consuming from a topic, the associated offsets for that topic should go through the expiry process – to avoid unnecessary expansion of the offset cache.

Unfortunately, there is no notification mechanism in place for member subscription change within a group. Therefore, a poll mechanism can be implemented to run at specific intervals and check whether group subscription has deviated from what is stored in the cache. One place to do this is the repeating offset cleanup scheduled jobs, which by default run every 10 minutes, making them a good choice as the group subscription check will not be executed very frequently. At every execution of this job we collect a list of all topic partitions the group is consuming from (this can be calculated based on the data in each group member's metadata), and cross reference it with the stored offsets for the group. If there are partitions the group has offset for but no longer consumes from, and `offsets.retention.minutes` has passed since their last commit timestamp, the corresponding offsets will be removed from the offset cache.

Standalone (Simple) Consumer

The standalone consumer uses Kafka for offset storage only. For this consumer the group state is always `Empty`, and the corresponding `protocolType` is `None`. Since the above mentioned expiration mechanism will not work for these consumers, the offset of a partition will be expired for them when `offsets.retention.minutes` passes since their last commit timestamp.

The following table summarize how the new offset expiration semantics would be implemented.

Group State	Additional Check in Offset Cleanup Job	Action if Check Holds
= `Empty` `(protocolType != None)`	current timestamp - `current_state_timestamp` broker's `offsets.retention.minutes`	1. Remove all group offsets 2. Transition the group to `Dead`
`Empty`	(Non-subscribed partitions = partitions group has offset for - partitions group is consuming from) partition non-subscribed partitions: • current timestamp - partition's `commit_timestamp` broker's `offsets.retention.minutes`	Remove offset of partition
= `Empty` `(protocolType = None)`	current timestamp - partition's `commit_timestamp` broker's `offsets.retention.minutes`	Remove offset of partition

Note that there are different valid `protocolType` values, such as `consumer` and `stream`, and the above semantics applies to them all.

Another Related Change

When group names are automatically generated by the console consumer they are very likely not to be reused. Therefore, it makes sense to skip storing offsets for them by default to avoid one of the top factors for offset cache size growth. The proposal is to disable auto offset commit by default in this situation. Implementing this change would become more critical once **KAFKA-3806** - Getting issue details... STATUS (KIP-186) lands: it changes the default retention from 1 day to 7 days.

Compatibility, Deprecation, and Migration Plan

- The new protocol does not allow clients to customize the retention time of specific offsets in the group. The old consumers, however, could still commit offsets with a customized retention time. Such old consumers will continue to be supported:
    - If a consumer uses the old API without customizing the retention time, the new approach will be applied; i.e., the broker's `offsets.retention.minutes` config will be used as the retention time of its offsets once it becomes Empty. The same retention will be used for offsets of partitions the group no longer consumes from (or is subscribed to).
    - If a consumer uses the old API with a customized retention time, the provided retention time will become the retention time of the offsets in question from the offset commit timestamp (this fully matches the current behavior). In this scenario, version 1 of offset commit value schema (with the `expire_timestamp` field) will be used.
- This should be rare, but clients who rely on the auto offset commit functionality of the consumer when the group name is auto-generated by console consumer, will need to manually set the auto offset commit to `true`.

Rejected Alternatives

1. Making all group offsets expire at the same time: Even though this is a good solution for when the group becomes `Empty`, it fails to address the scenario where the group stops consuming from a particular partition and causes those offsets to remain while the group exists, which leads to unnecessary expansion of the group metadata cache.