KIP-68 Add a consumed log retention before log retention

Status

Current state: *Under Discussion*

Discussion thread: *here [Change the link from the KIP proposal email archive to your own email thread]*

JIRA: *here [Change the link from KAFKA-1 to your own ticket]*

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

Motivation

Currently, kafka log retention will delete the log segment files which are not modified after a specified time, default is seven days. This specified time is hard to be determined in the real production environment, as several GB or TB data is generated every day. If the specified time is too large and the disk space is not enough to hold the data, the kafka broker will be out of service; If the specified time is too small, there is a risk of losing some messages which have not been consumed by all the consumer groups or even have not been consumed by any consumer.

In this KIP we want to add a consumed log retention before the original time-based log retention (we call it forced log retention), this is so called two-level time-based log retention.

In the consumed log retention, we will check whether the log segments are consumed by all the consumed groups, if so, we can safely delete the log segments and save more disk space. Otherwise we will not delete the log segments and wait until the forced log retention. For example, users can specify the forced log retention time to seven days, and consumed log retention to three days, so the messages which are consumed by all groups can be deleted after three days, those are not consumed will be deleted only after seven days.

Public Interfaces

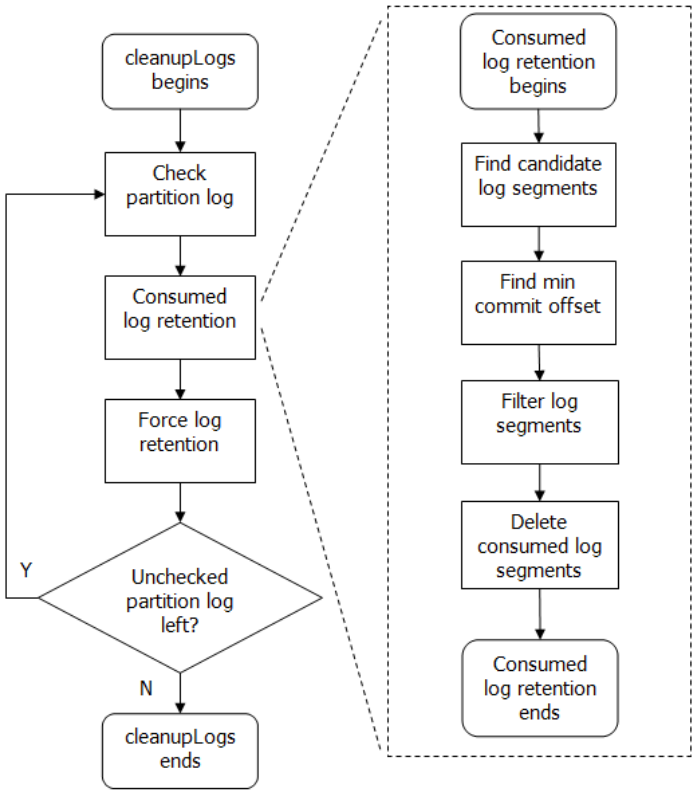
This KIP will add some configurations in the broker configuration file, please refer to the Configuration chapter.

Proposed Changes

There are two options: one is implement in the broker side, the other option is implement on the client side.

Option 1: Broker side

The changes are mainly focused on Log Manager. Below is the workflow:



1. Every time when cleanupLogs task begins to check log retention, for every partition, we add a consumed log retention process before the force log retention.
2. Then we will find the candidate log segments whose last modified time is not changed after the consumed log retention time for consumed log retention.
3. Find the min offset of all the consumer groups' commit offset of that partition.
4. Decide whether the log segment can be deleted according to the min offset.
5. After we finish the consumed log retention procedure, we will continue the forced log retention procedure for the remain log segments.

Option 2: Client Side

The client side is implement is the same in the find the min commit offset, the difference is after calculated the min commit offset, we used the KIP-47's trim Request to trim the log according to the min commit offset. This method make the broker simple but we should always running the admin tools with kafka, this may not always be possible.

Min Commit Offset

The min commit offset is the minimal commit offset of one partition of all the consumer groups. we can simply use the simple consumer's API to find every consumer group's commit offset, which had subscribed the topic containing the to-be deleted log segment. We can handle old consumer and new consumer in one function to query the commit offset like the consumer-offset-checker tool, and can handle potential failures like leader change or coordinator down. To avoid deleting the log segment which are not really consumed, whenever encounter commit offset querying exception, we will set the min commit offset to -1.

When the min commit offset is -1, we will skip the consumed log deletion procedure.

Filter Log Segment

After a valid min commit offset is found, we begin to filter the log segments which are out of date and have been consumed by all consumer groups.

We then compare the log segment's base offset with the min commit offset, if the min commit offset is larger than the base offset, we can't delete this log segment; if the min commit offset is located in the range between the base offset and the largest offset of this log segment, we can't delete this log segment as well. We can safely delete those log segments whose largest offset is smaller than the min commit offset.

Configuration

We will add some configs in the broker configuration file for the broker-level configurations:

Log.retention.commitoffset.enable: whether the consumed retention is enable or not, default is false

Log.retention.commitoffset.hours: the out of date time of the consumed retention by hours

Log.retention.commitoffset.minutes: the out of date time of the consumed retention by minutes

Log.retention.commitoffset.ms: the out of date time of the consumed retention by milliseconds

We will also add the corresponding configs in the topic-level configurations:

retention.commitoffset.hours

retention.commitoffset.minutes

retention.commitoffset.ms

The consumed retention time can’t be larger than force retention time (Log.retention.hours{miniutes, ms})

Compatibility, Deprecation, and Migration Plan

- The proposed change is backward compatible. Log.retention.commitoffset.enable default is false, no consumed log retention default. But even if this configuration is true, the original force log retention’s functionality remains the same.

Rejected Alternatives

Using consumed log retention instead of force log retention.

In this approach, we do not add the consumed log retention before the force log retention. Time-based log retention is only the consumed log retention, this can highly ensure that all the messages must be consumed at least once before deleted.

However, if many messages are not consumed, the disk space would be full and it will cause the kafka broker going down. So we should use two-level log retention instead of one-level log retention.