KIP-169 - Lag-Aware Partition Assignment Strategy

Status

Current state: Under Discussion

Discussion thread: link

JIRA: KAFKA-5337

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

Motivation

Under certain circumstances, the existing `RangeAssignor` and `RoundRobinAssignor` partition assignment strategies fail to produce a balanced assignment of partitions to consumers, in terms of the total lag assigned to each consumer. The above issue can be overcome by a new partition assignment that assigns partitions such that lag is distributed as evenly as possible across a consumer group.

Public Interfaces

This change would introduce a new partition assignment strategy, implemented as a new class `LagAwareAssignor` that can be used by specifying `org.apache.kafka.clients.consumer.LagAwareAssignor` for the value of the consumer property `partition.assignment.strategy`. The default value of this consumer property would not be changed.

Proposed Changes

Add a Lag-Aware Assignor option to the assignment strategies of the new consumer. The Lag-Aware Assignor operates on a **per-topic** basis and serves two purposes.

First, it guarantees an assignment that is as balanced as possible in terms of topic partition count, meaning either:

- the numbers of topic partitions assigned to consumers differ by at most one; or
- if a consumer A has 2+ fewer topic partitions assigned to it compared to another consumer B, none of the topic partitions assigned to B can be assigned to A.

Second, during a reassignment the Lag-Aware Assignor would perform the reassignment in such a way that in the new assignment,

1. topic partitions are still distributed as evenly as possible by topic partition count (see above), and
2. topic partitions are assigned such that the total lag is distributed as evenly as possible across the consumer group.

The first goal above takes precedence over the second one. This means that the distribution of total lags assigned to each consumer may still be uneven, but will be as even as possible for an assignment that is balanced in terms of topic partition count. Ensuring that assignments are balanced by topic partition count serves two purposes:

- it ensures that the assignment is balanced when all partitions have a current lag of 0 (in this case the resulting assignment will be similar to that of the `RangeAssignor`)
- it ensures that the assignment is not heavily skewed in terms of topic partition count per consumer, which would likely cause the distribution of lag across consumers to ultimately become unbalanced again if the assignment is retained for a long period.

Lag

The total lag assigned to each consumer is the sum of the lags across all assigned topic partitions. The lag on each topic partition is the difference between the offset of the next message to be published to the partition and the last offset committed by the consumer group for that partition.

Algorithm

The inputs to the lag-aware partition assignment algorithm are:

- subscription: mapping of consumer to subscribed topics
- partitionOffsetMetadataPerTopic: the earliest, latest and last-committed offsets (for the current consumer group) for each partition in each topic present in the subscription

The lag-aware partition assignment strategy maintains a number of data structures:

- topicPartitionLags - a map from topic partition to the current lag for that partition in that topic
- unassignedPartitions - a list of partitions yet to be assigned to a consumer, sorted in order of decreasing lag
- assignment - a map from consumer id to the list of topic partitions assigned to that consumer

The lag-aware partition assignment algorithm operates on a single topic at a time, within the context of a single consumer group, the following additional data structures are maintained during assignment of each topic:

- a map from consumer id to the total lag assigned to that consumer (consumerTotalLags)
- a map from consumer id to the total number of partitions assigned to that consumer (consumerTotalPartitions)

The algorithms includes the following main steps:

1. For each topic present in the subscription for at least one consumer, determine the current lag on each partition for the consumer group
    a. Lag is computed by subtracting the latest offset from the last committed offset for each partition
    b. If the consumer group has not yet committed an offset for a partition, determine the lag based on the `auto.offset.reset` consumer property, as follows:
        i. If `auto.offset.reset=latest`, the current lag for the partition is 0
        ii. If `auto.offset.reset=earliest` (or any other value) we assume a current lag equal to the total number of records currently available in that partition (i.e. latest offset - beginning offset)
2. For each topic present in the subscription:
    a. Sort partitions in order of decreasing lag
    b. While there are still unassigned partitions:
        i. Select the unassigned partition with the maximum lag
        ii. Assign this partition to the consumer with the minimum total number of assigned partitions
        iii. If all consumers have an equal number of partitions assigned, assign the partition to the consumer with the minimum total assigned lag

Obtaining Consumer Group Offsets

A means of obtaining the earliest, latest and last committed offset for each topic partition is required.

The prototype prototype implementation achieves this by creating an additional `KafkaConsumer` joined to the consumer group for which assignment is being performed. Offsets are then obtained via the Consumer API.

The prototype targeted Kafka 0.10.2.0. In Kafka 0.11.0.0 it may be more appropriate to obtain offsets via the new Admin API.

Prototype Implementation

A prototype implementation targeting Kafka 0.10.2.0 is available here.

Example

Suppose a consumer group contains two consumers C_0 and C_1, both subscribed to a topic t_0 with 3 partitions: t_0p_0 (lag 100,000), t_0p1 (lag 60,000) and t_0p2 (lag 50,000).

The assignment generated by both `RangeAssignor` and `RoundRobinAssignor` will be:

Consumer	Assigned Topic Partitions	Total Assigned Lag
C_0	t_0p_0, t_0p1	160,000
$C1$	t_0p2	50,000

The assignment generated by the proposed lag-aware assignor will be:

Consumer	Assigned Topic Partitions	Total Assigned Lag
C_0	t_0p_0	100,000
C_1	t_0p_1,t_0p_2	110,000

Compatibility, Deprecation, and Migration Plan

This change will have no impact on existing users unless the explicitly configure their consumers to use the new partition assignment strategy.

Rejected Alternatives

- Performing assignment solely based on total lag assigned to each consumer, without enforcing an even distribution of partition counts across consumers. This will produce assignments that ultimately cause the distribution of lags across consumers to become unbalanced again over time