KIP-874: TopicRoundRobinAssignor

Status

Current state: *Under Discussion*

Discussion thread: None

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

GitHub : https://github.com/apache/kafka/pull/12705

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

Motivation

We are using Kafka as a messaging system (without streaming) with the following properties :

- Each topic can contain only one type of message
- Each consumer can subscribe to any partitions of topics (subscriptions only depends on the partition assignment strategy)
- Each consumer uses the same error exception handling strategy, they stopped when the consumption failed (after retrying from 0 to n times)

Sometimes ago, we were using only one consumer per microservice so we have to scale up the number of pods (if topics have multiple partitions) if we would like to improve performance consumption.

Therefore, we have decided to have multiple consumers in a same microservice (1 thread per consumer).

Then some questions came up :

- How can we keep a good data consistency for a topic ? We basically want to consume all or nothing from a topic to avoid that some data are not be visibles for client for a long time whereas others (possibly created 1 or more hours after) are visibles
- How can we be thread safe as much as possible ? We are writing a company framework on top of Spring Boot, we would like to limit the risk of concurrency issues
- As consumers are stopped after failing, how can we avoid to propagate the "corrupt" message to other consumers of a same microservice to continue consumption of other topics ? We handle this by keeping in memory partitions that failed.

Combining these hypothesis and requirements, we though that having as many consumers as topics is the best option if each partitions of a topic is assigned to only one consumer.

So, we have needed of a new partition assignment strategy, the TopicRoundRobinAssignor.

How does it work ?

Suppose there are two consumers `C0` and `C1`, two topics `t0` and `t1`, and each topic has 3 partitions, resulting in partitions `t0p0`, `t0p1`, `t0p2`, `tp1p0`, `t1p1` and `t1p2`.

The assignment will be:

```
C0: [t0p0, t0p1, t0p2]
C1: [t1p0, t1p1, t1p2]
```

Why is it better for data consistency per topic ?

If we use one of the four existing assignor, we always get mixed partitions assigned to a topic (ie. a partition of different topic).
If one of the consumer fails, we kept in memory which partition failed in order to not propagate the "corrupt" message.
Otherwise, after rebalancing, when the "corrupt" message will be consumed by an other consumer, it will be stopped also, and so on, and so on...

Let me take the same example.
Suppose, there are 2 consumers `C0` and `C1`, and two topics `t0` and `t1`, and each has 3 partitions, resulting in partitions `t0p0`, `t0p1`, `t0p2`, `tp1p0`, `t1p1` a nd `t1p2`.

If I use the standard **RoundRobinAssignor**, the assignement will be :
```
C0: [t0p0, t0p2, t1p1]
C1: [t0p1, t1p0, t1p2]
```

If `C0` fails, it will be stopped and the new assignement will be :
```
C1: [t0p1, t1p0, t1p2, t0p0, t0p2, t1p1]
```

As we keep in memory that we do not want to consume `t0p0`, `t0p2`, `t1p1`, only records from `t0p1`, `t1p0`, `t1p2` will continue to be consumed.
Therefore, if we do not fix the issue quickly about the "corrupt" message, the data from each topic will diverge (some records will be consumed, some other not).

If we use the **TopicRoundRobinAssignor**, the assignment will be :
```
C0: [t0p0, t0p1, t0p2]
C1: [t1p0, t1p1, t1p2]
```

If `C0` fails, it will be stopped and the new assignment will be :
```
C1: [t0p0, t0p1, t0p2, t1p0, t1p1, t1p2]
```

As we keep in memory that we do not want to consume `t0p0`, `t0p1`, `t0p2`, only records from `t0p1`, `t0p1`, `t0p2` will continue to be consumed.
Therefore, I will continue to consume all records of a same topic and my data stay consistent as much as possible.

Why is it thread safer ?

If we use one of the four existing assignor, we always get mixed partitions assigned to a topic (ie. a partition of different topic) and so records of a same topic can be proceed concurrently.

Let me take an example.
Suppose, there are 2 consumers `C0` and `C1`, and two topics `t0` and `t1`, and each has 3 partitions, resulting in partitions `t0p0`, `t0p1`, `t0p2`, `tp1p0`, `t1p1` a nd `t1p2`.

If we use the standard **RoundRobinAssignor**, the assignement will be :
```
C0: [t0p0, t0p2, t1p1]
C1: [t0p1, t1p0, t1p2]
```

In that case, we must ensure that the processes executed when we received records from `t0` and `t1` must be thread safe because they came from two different threads `C0` and `C1` in parallel.

If we use the **TopicRoundRobinAssignor**, the assignement will be :
```
C0: [t0p0, t0p1, t0p2]
C1: [t1p0, t1p1, t1p2]
```

In that case, each record record of a same topic is processed in the same thread, therefore there are less risks of concurrency issues.

How does it work if we have multiple containers running the same application ?

Using the TopicRoundRobinAssignor, the partitions are uniformly balanced to each container, therefore we can get better performances.

Let suppose there are 2 instances of the same application `A0` and `A0`, 2 consumers `C0` and `C1`, and two topics `t0` and `t1`. `t0` has 3 partitions and `t1` has two partitions resulting in partitions : `t0p0`, `t0p1`, `t0p2`, `tp1p0`, `t1p2`.

If we use the **TopicRoundRobinAssignor**, the assignment will be :
```
A0: [C0: [t0p0, t0p2], C1: [t1p0]]
A1: [C0: [t0p1], C1: [t1p1]]
```

Public Interfaces

- org.apache.kafka.clients.consumer.TopicRoundRobinAssignor

Proposed Changes

I propose to add the TopicRoundRobinAssignor as a possible partition assignment strategy, please refer to : https://github.com/apache/kafka/pull/12705

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?* No impact
- *If we are changing behavior how will we phase out the older behavior?* Existing behavior will not change as it is a new possibility
- *If we need special migration tools, describe them here.* No migration tool needed
- *When will we remove the existing behavior?* No need to remove the existing behavior

Test Plan

The TopicRoundRobinAssignor is unit tested, please refer to : https://github.com/apache/kafka/pull/12705.

Rejected Alternatives

Anyone that needs the same assignment partition strategy can create its own assignor or simply copy/paste the content of the TopicRoundRobinAssignor in its code and use it by configuring the `partition.assignment.strategy` property.