KIP-369: Alternative Partitioner to Support "Always Round-Robin" Selection

Status

Current state: [*Accepted]*

Discussion thread: *here*

JIRA: *KAFKA-3333 Alternative Partitioner to Support "Always Round-Robin" Selection*

Motivation

In my organisation, we have been using kafka as the basic publish-subscribe messaging system provider. Our goal is to send event-based (secure, encrypted) SQL messages reliably, and process them accordingly. For us, the message keys represent some metadata which we use to either ignore messages (if a loop-back to the sender), or log some information. We have the following use case for messaging:

1) A business object transaction occurs at SQL server/HANA Database.

2) The event is captured at the ORM layer, and messaged across multiple data centres around the world.

3) A group of consumers (for each data centre with a unique consumer-group ID) will process messages from their respective partitions. 1 consumer per partition.

Under the circumstances, we only need a guarantee that same message won't be sent to multiple partitions. In other words, 1 partition will never be sought by multiple consumers.

Using DefaultPartitioner, we can achieve this only with NULL keys. But since we need keys for metadata, we cannot maintain "Round-robin" selection of partitions because a key hash will determine which partition to choose. We need to have round-robin style selection regardless of key type (NULL or not-NULL).

Proposed Changes

To address this issue, we are proposing an alternative, and more concrete partitioner - "RoundRobinPartitioner". We use "Round-Robin" as the new partitioner does not focus on the key or paritions. The partitioner code will almost be identical to DefaultPartitioner.partition() method, except that it will simply execute the "Null Key and No Partition" logic from DefaultPartitioner. The following is the content of partition() method for our new partitioner.

```
List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
int numPartitions = partitions.size();


    int nextValue = nextValue(topic);
    List<PartitionInfo> availablePartitions = cluster.availablePartitionsForTopic(topic);
    if (availablePartitions.size() > 0) {
        int part = Utils.toPositive(nextValue) % availablePartitions.size();
        return availablePartitions.get(part).partition();
    } else {
        // no partitions are available, give a non-available partition
        return Utils.toPositive(nextValue) % numPartitions;
    }
```

We would also like to clarify that this is not code "Duplication". We do not wish to change the DefaultPartitioner class, but want to "Reuse" certain portion of its logic to achieve this solution.

Public Interfaces

There is no requirement to change any interfaces. We simply use the existing ***paritioner.class*** *config in* ***server.properties*** *and use a* *<u>different</u>* *class name. But we are not changing the default value, which is DefaultPartitioner.*

We will extend default partitioner and override partition() method to achieve this functionality.

We will be adding some unit tests, but they will simply be a re-use for round-robin tests already performed for DefaultPartitioner.

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*
  There is no impact to existing users. This class does not need to be used unless someone has similar requirements.
- *If we are changing behavior how will we phase out the older behavior?*
  No change in any existing behaviour, since the class usage is controlled by partitioner.class property in server.properties. We are not changing the default value.
- *If we need special migration tools, describe them here.*
  Not required.
- *When will we remove the existing behavior?*
  Not required.

Rejected Alternatives

We could package it within our own custom jar and use it with every Kafka release. But our objective is to get this approved by the user community, include it in Kafka trunk, and allow other developers to build upon it.