KIP-34 Add Partitioner Change Listener to Partitioner Interface for Multiple Use Case

Status

Current state: *Under Discussion*

Discussion thread: *http://mail-archives.apache.org/mod_mbox/kafka-dev/201509.mbox/%3CCAG9AH6cp57nsCfahndtbBCPP_Nk3AW8TmHV6drSwSd8Eswv0Ag@mail.gmail.com%3E*

JIRA: *TBD (If Kafka Dev team decide to incorporate this feature)*

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

Motivation

Based on KIP-22, New (Java) Kafka Producer has capability to plug-in custom logic to assign partition to given message per topic. However, the implementation of custom partitioning strategies depends on critical information such as # of partition change, online vs offline partitions. This information is available to producer internally via each metadata refresh interval **(metadata.max.age.ms**) or when certain type of error occurs within Network Client Code (https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/NetworkClient.java) will attempt metadata refresh.

Hence, in order to make this interface reach in capabilities and functionality, **Partitioner** interface needs to be notified of following event:

1) When # of partition for given topic changes (typically one of the fundamental use case is adding more partitions to existing topic for horizontal scalability or processing topic faster)

One of the limitation of New Producer compare to Sacala Based Old Producer is that Old Producer did not impose any restriction on growing # of partition (by which I mean no impact to running producer in production) but New Producer does implicitly impose max limit before activating policy set by **block.on.buffer.full** (based on # of batch.size exceeds configured limit of **buffer.memory**). For example, you have New Producer running in production environment and you do not have inventory of configuration and you need to grow partition, but # of partition (# of batch.size) grows beyond configured for running New Producer **buffer.size**, producer will drop or block.

Here is historic context on this subject and Kafka Dev Team Position on this issue:

http://qnalist.com/questions/5386817/java-new-producer-changing-partition-number-and-its-impact

Justification for this notification:

By using this notification, Custom Partitioner will not allow growing partition beyond max limit, (or till degrade producer performance) and will not activate **block.on.buffer.full.**

Also, when a New Producer restarted (application restarts) with old configuration and # partitions does grow beyond its limit (set by **buffer.memory**), it can elect to choose random set of partitions which but restrict # would avoid above situation.

2) When partitions are not online (suppose both replica/leader broker is down for some partitions), so implementer of this Partitioner will have capability to make decision about how to redirect message others online partitions.

Another motivation for this event listener is to give operational capability to monitor change request from Producer point of view:

- Capability to Monitor Event From Producer
- Ability to implement feedback loop with Change Controller (e.g Kafka NOC/Dev Ops team) by which producer can acknowledge via change (e.g partition increase, decrease or auto change partition online/offline can be integrated) and change can be validated. Hence, provide end-end visibility to Kafka Dev Ops Team.
- Also consumer side can also benefits from this event notification to report or acknowledge that change has been propagated and accepted by the Consumer Group that is consuming topic.

Public Interfaces

Proposal is to add new on change method to existing **Partitioner** Interface:

```
package org.apache.kafka.clients.producer;
import java.util.Map;
import org.apache.kafka.common.Configurable;
import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;
import org.apache.kafka.common.TopicPartition;
/**
 * Partitioner Interface
 */
public interface Partitioner extends Configurable {
    /**
     * Compute the partition for the given record.
     *
     * @param topic The topic name
     * @param key The key to partition on (or null if no key)
     * @param keyBytes The serialized key to partition on( or null if no key)
     * @param value The value to partition on or null
     * @param valueBytes The serialized value to partition on or null
     * @param cluster The current cluster metadata
     */
    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster 
cluster);


    /**
     * When number of partition changes or Partition becomes online or offline or any meta data or information 
changes
     * on about partition, this notification is triggered, it must be quickly act upon change or may choose to 
do nothing.
     *
     * @param cluster  The new cluster metadata
     * @param partitionsChanged  Only partitions That have changed with its partition information.
     *
     */
    public void onPartitionsChange(Cluster cluster, Map<TopicPartition, PartitionInfo> partitionsChanged);

         /**
     * This is called when partitioner is closed.
     */
    public void close();
}
```

Proposed Changes

Kafka Internal Change:

Kafka already has internal way of getting notified when Metadata Request Update via following internal listener.

https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/Metadata.java

```
    // This is existing internal interface.
    /**
     * MetadataUpdate Listener
     */
    public interface Listener {
        void onMetadataUpdate(Cluster cluster);
    }
```

Leverage existing implementation for metadata refresh notification, the MetadaChangeListener class will be notified upon refresh and intern it will do diff with previous instance of Cluster and notify change via onPartitionsChange() method as shown below.

```
package org.apache.kafka.clients;
import java.util.HashMap;
import java.util.Map;
import org.apache.kafka.clients.Metadata.Listener;
import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;
import org.apache.kafka.common.TopicPartition;

public class MetadatChangeListener implements Listener {

    private Partitioner partitioner;
    private Cluster prevClusterInfo;


    public MetadatChangeListener(Partitioner partitioner, Cluster
            currentClusterInfo ){
            this.partitioner = partitioner;
            this.prevClusterInfo = currentClusterInfo;
    }
    @Override
    public void onMetadataUpdate(Cluster cluster) {
        Map<TopicPartition, PartitionInfo> partitionsChanged = new HashMap<TopicPartition, PartitionInfo>();
        /***
                 * TODO :  Implement this

         * Determine the change from prevClusterInfo
         * Do the diff with prevClusterInfo and add to partitionsChanged
         */

        partitioner.onPartitionsChange(cluster, partitionsChanged);
    }
}
```

please note the above implementation is pseudocode implementation.

Compatibility, Deprecation, and Migration Plan

None

Rejected Alternatives

1) Each of the Partitioner class also implement the Listener and each metadata refresh will call listener. This will not scale when you each one of the Patitioner will need to do diff between prev and old.