# KIP-694: Support Reducing Partitions for Topics

## 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]*

## Motivation

When Kafka is used as messaging infrastructure of consumer business, we often need to adjust the capacity of certain topics to accommodate the bursty nature of the data. The adjustment involves both adding more partitions hence the throughput, and desirably reducing partitions so that they can be reclaimed. Operating clusters with ever growing partitions adds overhead to the operation. First, the performance degrades when a single disk needs to support large number partitions; and second larger cluster footprints makes it more vulnerable to disruption at infrastructure level such as machine or rack decommission (which is not uncommon in large enterprise). This motivates us to add an easy and transparent way to reduce partitions for topics, which is particularly convenient in the following situation.

- The cluster has a large number of topics and total number of partitions is close to the limit.
- Most of the dynamic topics do not contain keyed messages. (See limitation section)
- The data retention period is relatively short.

This proposal is based on what is described in KIP-500 group of changes, mainly the removal of zookeeper and migrating metadata to an internal Raft quorum.

We have implemented the described changes and deployed it in various setups internally in production environment.

## Public Interfaces

### Configuration Changes

Two configuration fields are added

- delete.topic.partition.enable is a boolean value indicating wether this functionality is enabled.
- delete.topic.partition.interval.delay.ms is the check interval during delayed deletion. The default value is 300000 (5 minutes).

### Metadata Changes

We propose to modify PartitionRecord and add a DeleteTopicPartitionRecord in __cluster_metadata

#### PartitionRecord

We propose to add new field 'mode' for a partition that indicates if a partition is to be removed. specifically at any time a partition can be

- ReadWrite (code=0)it means the partition can be read from and written to.
- ReadOnly (code=1)it means the partition can only be read
- None (code=-1)it means the partition should be filtered and not written to, but consumption is not impacted.

```
{
  "apiKey": 3,
  "type": "metadata",
  "name": "PartitionRecord",
  "validVersions": "0",
  "fields": [
    { "name": "PartitionId", "type": "int32", "versions": "0+", "default": "-1",
      "about": "The partition id." },
    { "name": "TopicId", "type": "uuid", "versions": "0+",
      "about": "The unique ID of this topic." },
    { "name": "Replicas", "type":  "[]int32", "versions":  "0+",
      "about": "The replicas of this partition, sorted by preferred order." },
    { "name": "Isr", "type":  "[]int32", "versions":  "0+",
      "about": "The in-sync replicas of this partition" },
    { "name": "RemovingReplicas", "type":  "[]int32", "versions":  "0+", "nullableVersions": "0+",
      "about": "The replicas that we are in the process of removing." },
    { "name": "AddingReplicas", "type":  "[]int32", "versions":  "0+", "nullableVersions": "0+",
      "about": "The replicas that we are in the process of adding." },
    { "name": "Leader", "type": "int32", "versions": "0+", "default": "-1",
      "about": "The lead replica, or -1 if there is no leader." },
    { "name": "LeaderEpoch", "type": "int32", "versions": "0+", "default": "-1",
      "about": "An epoch that gets incremented each time we change the ISR." },
    { "name": "Mode", "type": "int32", "versions": "0+", "default": "0",
      "about": "The read write mode of the current partition." }
  ]
}
```

### DeleteTopicPartitionRecord

```
{
  "apiKey": xxx,
  "type": "metadata",
  "name": "DeleteTopicPartitionsRecord",
  "validVersions": "0",
  "fields": [
    { "name": "TopicId", "type": "uuid", "versions": "0+",
    "about": "The topicpartition to remove. " },
    { "name": "DeletePartitions", "type": "int32", "versions": "0+","about": "The partition to remove. All
associated partitions will be removed as well." },
     { "name": "DeletePartitionsDelayTimestamp", "type": "bool", "versions": "0+","about": "Delay timestamp
deletion of data." },
     { "name": "CreateTimestamp", "type": "int64", "versions": "0+","about": "The record create timestamp." },
  ]
}
```

## AdminClient API changes

The AdminClient API will have new methods added

```
deletePartitions(Map<String, DeletePartitions> partitions)

@InterfaceStability.Evolving
public class DeletePartitions {
    private int totalCount;
    private boolean immediateDelete;
```

## Protocol RPC changes

## DeletePartitions API

A new API *DeletePartitions* will be added with the following *DeletePartitionsRequest* and *DeletePartitionsResponse*

```
DeletePartitionsRequest Request (Version: 0) => timeout_ms
immediate_delete [topics] TAG_BUFFER
  timeout_ms => INT32
  delete_partitions_delay => LONG
  topics => name [partitions] TAG_BUFFER
    name => COMPACT_STRING
    partitions => INT32 TAG_BUFFER
```

```
DeletePartitionsResponse Response (Version: 0) => throttle_time_ms error_code error_message [responses]
TAG_BUFFER
        throttle_time_ms => INT32
        error_code => INT16
        error_message => COMPACT_NULLABLE_STRING
        responses => name [partitions] TAG_BUFFER
                name => COMPACT_STRING
                partitions => partition_index status_code error_code error_message TAG_BUFFER
                        partition_index => INT32
                        status_code => INT8
                        error_code => INT16
                        error_message => COMPACT_NULLABLE_STRING
```

## UpdateMeta API

In UpdateMeta API we add mode field to return the partition read/write status of the current topic partitions.

```
UpdateMetadata Request (Version: 7) => controller_id controller_epoch broker_epoch [topic_states]
[live_brokers] TAG_BUFFER
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64
  topic_states => topic_name [partition_states] TAG_BUFFER
    topic_name => COMPACT_STRING
    partition_states => partition_index controller_epoch leader leader_epoch [isr] zk_version [replicas]
[offline_replicas] mode TAG_BUFFER
      partition_index => INT32
      controller_epoch => INT32
      leader => INT32
      leader_epoch => INT32
      isr => INT32
      zk_version => INT32
      replicas => INT32
      offline_replicas => INT32
      mode => INT8
  live_brokers => id [endpoints] rack TAG_BUFFER
    id => INT32
    endpoints => port host listener security_protocol TAG_BUFFER
      port => INT32
      host => COMPACT_STRING
      listener => COMPACT_STRING
      security_protocol => INT16
    rack => COMPACT_NULLABLE_STRING
```

```
UpdateMetadata Response (Version: 7) => error_code TAG_BUFFER
  error_code => INT16
```

## MetaData API

Add 'mode' field in Metadata API that represents the read/write model of the partitions of current topic, meanwhile incrementing the version of ApiKey.

```
Metadata Request (Version: 10) => [topics] allow_auto_topic_creation include_cluster_authorized_operations
include_topic_authorized_operations TAG_BUFFER
  topics => name TAG_BUFFER
    name => COMPACT_STRING
  allow_auto_topic_creation => BOOLEAN
  include_cluster_authorized_operations => BOOLEAN
  include_topic_authorized_operations => BOOLEAN

Metadata Response (Version: 10) => throttle_time_ms [brokers] cluster_id controller_id [topics]
cluster_authorized_operations TAG_BUFFER
  throttle_time_ms => INT32
  brokers => node_id host port rack TAG_BUFFER
    node_id => INT32
    host => COMPACT_STRING
    port => INT32
    rack => COMPACT_NULLABLE_STRING
  cluster_id => COMPACT_NULLABLE_STRING
  controller_id => INT32
  topics => error_code name is_internal [partitions] topic_authorized_operations TAG_BUFFER
    error_code => INT16
    name => COMPACT_STRING
    is_internal => BOOLEAN
    partitions => error_code partition_index leader_id leader_epoch [replica_nodes] [isr_nodes]
[offline_replicas] mode TAG_BUFFER
      error_code => INT16
      partition_index => INT32
      leader_id => INT32
      leader_epoch => INT32
      replica_nodes => INT32
      isr_nodes => INT32
      offline_replicas => INT32
      mode => INT8
    topic_authorized_operations => INT32
  cluster_authorized_operations => INT32
```

## Client API changes

- Add a new partition field 'mode' in org.apache.kafka.common.PartitionInfo
- Both KafkaProducer and KafkaConsumer are aware of state of the partition and filter accordingly.
    - Partitions with ReadOnly and None status will be filtered out for writes.
    - Partitions with None status will be filtered out for reads.

```
public class PartitionInfo {
    private final String topic;
    private final int partition;
    private final Node leader;
    private final Node[] replicas;
    private final Node[] inSyncReplicas;
    private final Node[] offlineReplicas;
    private int mode;
}
```

## Kafka Topic Command changes

- Support using --partitions options to specify a smaller number than current partitions
- Add --delete-partitions-delay option (Long) to specify when the data should be deleted. The default value is 0 meaning the partition should be deleted right away.

```
./kafka-topics.sh --alter --topic ... --partitions ...--delete-partitions-delay ...
```

- The command might issue warning under certain situations, such as
    - *"WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected"*
    - *"WARNING: This feature is only enabled with Metadata version above 10 and delete.topic.partition.enable turned on"*
    - *"WARNING: The topic is currently under changes"*

# Proposed Changes

The following change in Controller will be made:

- Added controller event TopicPartitionDeletion
- Add a class TopicPartitionDeletionManager to handle TopicPartitionDeletion event
- When KafkaController starts, a scheduleDelayDeletePartitionTask is scheduled periodically to check retention for delayed deletion.

The workflow involving TopicPartitionDeletionManager class is summarized as below:

- TopicCommand executes the DeletePartition RPC command to KafkaController and saves DeleteTopicPartitionsRecord in the KafkaController metadata.
- TopicPartitionDeleteManager starts to execute onPartitionDeletion method, updates the mode of Partition to *ReadOnly*. The partition remains in *OnlinePartition* state. All brokers are notified through PartitionStateMachine.
- ScheduleDelayDeletePartitionTask will update the Partition mode to *None* after specified delay period. The partition state changes to *OfflinePartition* and *NonExistentPartition*. The brokers are notified through PartitionStateMachine. and the partition replica status changes to *OfflineReplica* and *ReplicaDeletionStarted*, stops synchronizing data and clear data at all broker through ReplicaStateMachine.
- When Controller gets the successful stopReplica response from Broker, the Partition replica status is changed to ReplicaDeletionSuccessful, Then it cleans up metadata as well. otherwise, the Partition replica status changes to ReplicaDeletionIneligible, and waits for KafkaController to try again.

# Compatibility, Deprecation, and Migration Plan

The proposed change is compatible with Kafka clients backwards and forwards with current constraints

- When older clients access new broker and cannot interpret 'mode', we suggest the administrator to manually set delete.topic.partition.enable to true. This will restrict the Metadata  request to be above the specified version otherwise it will get LEADER_NOT_AVAILABLE.
- When newer clients access older broker (with new version org.apache.kafka.common.PartitionInfo class), the default value of the partition status field of the 'mode' field is ReadWrite, hence there will be no impact.

Upgrading the cluster from any older version is possible with the above situation handled.

# Rejected Alternatives