KIP-380: Detect outdated control requests and bounced brokers using broker generation

Status

Current state: *Accepted*

Discussion thread: *here*

Vote thread: *here*

JIRA: KAFKA-7235

Pull Request: PR-5821

Relates to:

- KAFKA-6604
- KAFKA-1120

Motivation

Currently, controller interacts with brokers by sending out control requests (LeaderAndIsrRequest, UpdateMetadataRequest, StopReplicaRequest) to brokers and detecting broker state change by listening on the /brokers/ids/znode. Broker can only rejects controller requests if they are sent from a stale controller with older controller epoch. The following assumptions have been made in the current implementation:

1. Broker should always process the control requests sent by the active controller.
2. Controller should always process the ControlledShutdownRequest sent by the broker.
3. The first LeaderAndIsrRequest received by a broker after it starts up always contains all partitions hosted by the broker.
4. By the time a broker finishes bouncing, controller should have processed the BrokerChange event.

These assumptions does not always hold and there will be correctness issues if either one of the assumption breaks. For example:

- If a broker bounces during controlled shutdown, the bounced broker may accidentally process its earlier generation's StopReplicaRequest sent from the active controller for one of its follower replicas, leaving the replica offline while its remaining replicas may stay online
- If controller receives old ControlledShutdownRequest (due to retries) after the broker has been bounced, controller will proactively send out control requests to the broker, which may leave the broker in a bad state.
- If the first LeaderAndIsrRequest that a broker processes is sent by the active controller before its startup, the broker will overwrite the high watermark checkpoint file and may cause incorrect truncation (KAFKA-7235)
- If a broker bounces very quickly, the controller may start processing the BrokerChange event after the broker already re-registers itself in zk. In this case, controller will miss the broker restart and will not send any requests to the broker for initialization. The broker will not be able to accept traffics.

Also, these issues will happen more frequently if we further optimize the broker rolling bounce time. The optimization (KAFKA-7283) is currently blocked by this KIP.

The root cause of these correctness issues is that we don't have a way to distinguish the broker's state after a bounce. To fix these issues, this KIP proposes the concept of broker generation, which is a unique & monotonically increasing value that changes every time when a broker (re)-joins the cluster. Control requests will include the broker generation of the destination broker and a broker will reject requests that are intended to be sent to the previous generations. Controller will use the broker generation to detect bounced brokers.

Public Interfaces

This KIP will include the broker generation (broker_epoch) in LeaderAndIsrRequest, UpdateMetadataRequest, StopReplicaRequest, ControlledShutdownRequest and bump up their protocol versions. Since we will evolve the schema of control requests in this KIP, I would also normalize the schema of these requests by avoiding data redundancy for the topic strings to reduce the memory footprint in the controller side and reduce the amount of data we send across the network.

Changes in Control Requests:

LeaderAndIsrRequest V2

```
LeaderAndIsr Request => controller_id controller_epoch broker_epoch [topic_states] [live_leaders]
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64                                                    <-- NEW
  topic_states => topic [partition_states]          <-- NEW
    topic => STRING
    partitions_states => partition controller_epoch leader leader_epoch [isr] zk_version [replicas] is_new
            partition => INT32
            controller_epoch => INT32
            leader => INT32
            leader_epoch => INT32
            isr => INT32
            zk_version => INT32
            replicas => INT32
            is_new => BOOLEAN
  live_leaders => id host port
    id => INT32
    host => STRING
    port => INT32
```

UpdateMetadataReuqest V5

```
UpdateMetadata Request => controller_id controller_epoch broker_epoch [topic_states] [live_brokers]
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64                                                                  <-- NEW
  topic_states => topic [partition_states]                  <-- NEW
    topic => STRING
    partition_states => partition controller_epoch leader leader_epoch [isr] zk_version [replicas]
[offline_replicas]
            partition => INT32
            controller_epoch => INT32
            leader => INT32
            leader_epoch => INT32
            isr => INT32
            zk_version => INT32
            replicas => INT32
            offline_replicas => INT32
  live_brokers => id [end_points] rack
    id => INT32
    end_points => port host listener_name security_protocol_type
      port => INT32
      host => STRING
      listener_name => STRING
      security_protocol_type => INT16
    rack => NULLABLE_STRING
```

StopReplicaRequest V1

```
StopReplica Request => controller_id controller_epoch broker_epoch delete_partitions [topic_partitions]
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64                                          <-- NEW
  delete_partitions => BOOLEAN
  topic_partitions => topic [partition]          <-- NEW
    partition => INT32
```

ControlledShutdownRequest V2

```
ControlledShutdown Request => broker_id broker_epoch
  broker_id => INT32
  broker_epoch => INT64                    <-- NEW
```

Note: Normalizing the schema is a good-to-have optimization because the memory footprint for the control requests hinders the controller from scaling up if we have many topics with large partition counts. We already did the same thing in other types of request (e.g. Produce, Fetch, ...).

New Error Code:

New error code STALE_BROKER_EPOCH (77) and a new type of exception StaleBrokerEpochException will be added. This error is used in the following scenario:

- When a broker sees a LeaderAndIsrRequest/UpdateMetadataRequest/StopReplicaRequest with outdated broker epoch, it will respond back with STALE_BROKER_EPOCH error. The controller **will not** resend the request.
- When the controller sees a ControlledShutdownRequest with outdated broker epoch, it will respond back with STALE_BROKER_EPOCH error. If the broker gets quickly restarted, it will no see the error response since the channel has already been closed during broker shutdown. If the broker just gets disconnected from zookeeper and re-connect during controlled shutdown, it will retry to send the ControlledShutdownRequest with newer broker epoch to controller.

Proposed Changes

Broker Registration in Zookeeper

The czxid (create transation id) of the broker's zookeeper ephemeral znode will be used for broker generation.

When a broker (re)-joins the cluster, it needs to create the ephemeral znode in /brokers/ids/ and get back the broker generation (czxid) atomically in order to reject requests applied to old generations. Since czxid is only available in the Stat of the znode and zookeeper does not include the Stat of the znode in the CreateResponse, we need to use zookeeper multi op to atomically create and get the Stat of the znode (through a SetData Op). This also requires a ZookeeperClient refactor because we don't expose multi op explicitly.

Broker Rejects Control Requests Applied to Former Generations

A broker will extract the broker generation (czxid) in LeaderAndIsrRequest/UpdateMetadataRequest/StopReplicaRequest and will reject the requests with smaller broker generation than its current generation.

Controller Rejects ControlledShutdownRequest Sent from Former Generations

During controlled shutdown, the broker will include its current broker generation (czxid) in the ControlledShutdownRequest. Upon receiving ControlledShutdownRequest, controller will check the broker generation (czxid) in ControlledShutdownRequest and will reject the request if its broker generation is smaller broker generation than the broker generation cached in the controller side. This guarantees controller will not attempt to send out control requests to move partitions and stop replicas in reaction to a broker cleaned shutdown if the broker has already been restarted.

Controller Detects Bounced Broker

In order to avoid missing broker state change when fast broker bounce happens, the logic in controller processing BrokerChange event should be:

1. Reads all child nodes in /brokers/ids/ to get current brokers with broker generation
2. Detect new brokers, dead brokers and bounced brokers:
    - new brokers: brokers exist in current brokers list and do not exist in controller context
    - dead brokers: brokers exist in controller context but do not exist in current brokers list
    - bounce brokers: brokers exist in both current brokers list and controller context but have higher generation than the cached value in controller context
3. Update the live broker ids in controller context
4. Handle broker state change:
    - new brokers: update broker generation in controller context for new brokers, then invoke onBrokerStartUp(new brokers)
    - dead brokers: invoke onBrokerFailure(dead brokers)
    - bounced brokers: invoke onBrokerFailure(bounced brokers) first, then update broker generation in controller context for bounced brokers, finally invoke onBrokerStartUp(bounced brokers)

Compatibility, Deprecation, and Migration Plan

1. Upgrade the brokers once with the inter-broker protocol set to the previous deployed version
2. Upgrade the brokers again with an updated inter-broker protocol.

Rejected Alternatives

We considered using the zookeeper client's session id as the broker generation previously because in this case broker can avoid the multi op workaround to get the broker generation by calling ZooKeeper.getSessionId() and the controller can read the ephemeralOwner property of the broker znode to infer the broker generation. But later on we realized that this approach is incorrect because zookeeper client's session id is not monotonically increasing.