KIP-343: Add a Controller Heartbeat Mechanism

Status

Current state: Under Discussion

Discussion thread (none)

JIRA*: (none)*

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

Motivation

Currently, Apache Kafka nodes are considered "alive" if they can reach *Apache ZooKeeper*. However, there are cases where a node can reach *ZooKeeper*, but not the current controller. When a node is partitioned from the controller, yet remains part of the cluster, it receives no metadata updates. As the cluster changes over time, this can result in divergent behavior where the node believes it has partitions that it no longer has, or still believes it has partitions that it does not. This can cause persistent under-replicated partitions, difficult-to-analyze failure scenarios, and in the worst cases, even data loss. Because clients fetch metadata from a random broker, it can also cause clients to receive arbitrarily old data if they happen to contact the partitioned broker or brokers. With this stale metadata, clients are not guaranteed to make progress.

We would like to fix these scenarios, and improve the overall maintainability of the system.

Proposed Changes

Broker Incarnation ID

Currently, brokers are identified only by their IDs. If a broker is restarted, it will have the same ID after the restart as before. This makes it difficult to detect cases where metadata needs to be resent to new instances of a broker. Similarly, if multiple brokers are accidentally configured with the same ID, major problems can result. To resolve these issues, we will add a new ID, the incarnation ID (IID) to uniquely identify each instance of a specific broker ID.

The incarnation ID is a uniquely, monotonically increasing 64-bit ID. In order to get this ID, we will use the czxid of the ephemeral ZNode /broker/id/$BROKER_ID.

Broker State

The controller maintains a state for each broker in the cluster. This state may be either "inactive," "active," or "stopping."

When a broker is in the "inactive" state, it is not included in the metadata sent to brokers or clients. For most purposes, an inactive broker is treated the same as an absent broker. When the cluster is first initialized, all nodes will be in the inactive state.

Brokers in the "active" and "stopping" states handle traffic normally. When a broker is "stopping," that means it is undergoing controlled shutdown. During controlled shutdown, we gradually migrate leaders away from the broker.

Controller Heartbeat

Broker-Side Heartbeat Timeout

In order to test if it is partitioned, each node will periodically send a heartbeat to the current controller. If the node cannot contact the controller within a certain period of time, it will fence itself.

When a node enters the fenced state, existing network requests will be completed, but new ones will not be accepted. Nodes will transition back and forth between fenced and unfenced states as appropriate.

Controller-Side Heartbeat Timeout

The controller will declare a broker inactive if it fails to heartbeat after a certain amount of time. Conversely, if a previously inactive broker successfully heartbeats, it can be declared active again.

This timeout is separate from the broker-side timeout; it happens on the controller, rather than on the requesting broker. The broker-side timeout should be longer than the controller-side timeout. We do not want the broker to fence itself until it has been declared inactive by the controller.

Controller Logic

Heartbeat requests contain a target broker state which the broker would like to get to. The responses to these heartbeats contain the actual state which the controller has determined that the broker should transition to.

The heartbeat request includes the broker incarnation ID. The controller will verify that this is valid for the given broker ID. If the broker's incarnation ID is lower than what the controller thinks it should be, the controller will assume that the broker is an old incarnation, and return an error code. If the broker's incarnation ID is higher, the controller will update the target incarnation ID in memory, and treat the broker as valid.

Once it has verified that the incarnation ID is valid, the controller will check the target state contained in the heartbeat request. If the broker is targetting "stopping" state, then it will be transitioned into that state. Similarly, if it is targetting "active" state, it will transitioned into that state.

blocked URL

Initiating ISR Changes

When the leader of a partition wants to change the ISR, it will make an RPC to the controller to request the change. We will create a new RPC request and response for this purpose, IsrChangeRequest / IsrChangeResponse.

Public Interface Changes

Public API Changes

BrokerState Enum

Value	Name	Description
0	Initial	The broker has not yet contacted the controller.
1	Inactive	The broker should be fenced from the cluster.
2	Active	The broker is ready to handle requests.
3	Stopping	The broker is ready to handle requests. It is in the process of migrating its partition leaderships to another node.

MetadataHeartbeatRequest

```
MetadataHeartbeatRequest => broker_id incarnation_id target_state
broker_id => INT32
incarnation_id => INT64
target_state => INT8
```

The heartbeat request contains the broker ID and incarnation ID of the broker. It also has the target state which the broker would like to achieve.

```
MetadataHeartbeatResponse => error_code error_string
error_code => INT16
error_string => NULLABLE_STRING
current_state => INT8
```

The response will contain an error code of *INVALID_INCARNATION_ID* if the incarnation ID was not accepted by the controller.

The response will contain an error code of *INVALID_REQUEST_EXCEPTION* if the request did not contain valid endpoint fields, or had an invalid negative broker ID.

If the response contains an error code of *NONE*, the broker ID and incarnation ID will be set to the appropriate values for the broker to use.

IsrChangeRequest

Brokers make an *IsrChangeRequest* to the controller to request that an ISR be changed.

```
IsrChangeRequest => broker_id incarnation_id [topic]
broker_id => INT32
incarnation_id => INT64
topic => topic_name [partition]
topic_name => STRING
partition => leader_epoch partition_index [isr_node]
leader_epoch => INT32
partition_index = INT32
isr_node => INT32
```

The *IsrChangeRequest* contains the broker ID and incarnation ID of the broker making the request. It also has a list of topics. Each topic contains a list of partitions, along with the leader epochs of each, and the set of nodes that should be in each.

```
IsrChangeResponse => top_level_error_code top_level_error_message [result]
top_level_error_code => INT16
top_level_error_message => NULLABLE_STRING
result => error_code error_message
error_code => INT16
error_message => NULLABLE_STRING
```

The top level error code will be *NOT_CONTROLLER* if the recipient was not the current controller.

The top level error code will contain an error code of *INVALID_INCARNATION_ID* if the incarnation ID was wrong. This means that someone started a new broker process which registered with the broker ID that we are currently using. The new registration overwrote the old one, and we are now prohibited from changing the ISR.

If the top level error code is *NONE*, there will be a list of results. The results will appear in the same order as the requests appeared. Each result contains an error code.

The partition response will contain an error code of *NOT_LEADER_FOR_PARTITION_EXCEPTION* if the requester is no longer the leader for the partition.

The partition response will contain an error code of *INVALID_REQUEST_EXCEPTION* if the request was invalid. For example, if the broker attempted to shrink the ISR to contain 0 brokers, that would be an invalid request.

The partition response will contain an error code of *NONE* if the ISR change was successfully applied.

New Broker Configuration Keys

controller.heartbeat.timeout.ms determines how many milliseconds the controller will wait for a heartbeat before declaring a node inactive.

broker.heartbeat.timeout.ms determines how many milliseconds the broker will wait to contact the controller before fencing itself.

Compatibility, Deprecation, and Migration Plan

Kafka rolling upgrades proceed in two stages, via the so-called "double roll." During the first roll, nodes are taken down one by one and restarted with the new software, but with the existing inter-broker protocol (IBP). During the second roll, nodes are taken down and restarted with the latest IBP configured.

If the IBP is too low, brokers will not fence themselves or send heartbeat requests. They will also directly modify ZooKeeper rather than making IsrChangeRequests. In other words, the new behaviors are gated behind the IBP.

Brokers that want to use the new behavior will signify their readiness by registering themselves in ZooKeeper with a BrokerInfo structure that has a version of 5 instead of 4. (Note that there are no new fields in the BrokerInfo structure. The only change is the version.)

In contrast, when the controller sees a broker register itself in ZK with an older BrokerInfo version, it will consider this a "legacy registration." Legacy brokers are always considered to be active, even if they don't send heartbeats. This is necessary to handle the case where some brokers have upgraded, but others have not.

Rejected Alternatives

Broker Registration RPC

Rather than continuing to register brokers through ZooKeeper, we could have brokers register themselves to the controller directly. However, a change to registration would be more difficult to do in a compatible fashion.