KIP-xxx: Cache the UpdateMetadata requests in controller

Status

Current state: Under discussion

Discussion thread:

Vote thread:

JIRA:

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

Motivation

When a node takes over as the active controller, it needs to read from zookeeper for the whole cluster state and then constructs full UpdateMetadata requests to be sent to all brokers in the cluster. The full state includes all partitions in the current cluster with the live nodes. Thus the total memory occupied by these UpdateMetadata requests is proportional to the number of partitions in the cluster * the number of brokers. In a large cluster with hundreds of thousands of partitions, and 200 brokers. These UpdateMetadata requests can take up to 30G of ram, and cause the JVM to run out of memory and crash sometimes.

The situation is exacerbated by the fact that when there are N brokers in a large cluster, not only does the controller construct N copies of UpdateMetadataRequestData objects, but also N serialized byte buffers for these objects.

This KIP tries to address the problem by making the UpdateMetadata requests cacheable so that a single copy can be used for all brokers. The end result is that the memory footprint will be proportional only to the number of partitions, regardless of how many brokers are in the cluster.

Public Interfaces

This KIP bumps the versions of the 3 types of control requests in order to introduce a new field max_broker_epoch and remove the existing field broker_epoch:

LeaderAndIsrRequest V5

```
LeaderAndIsr Request => controller_id controller_epoch broker_epoch [topic_states] [live_leaders]
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64                     <-- Will be removed
  max_broker_epoch => INT64                 <-- New
  topic_states => topic [partition_states]
    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 V7

```
UpdateMetadata Request => controller_id controller_epoch broker_epoch [topic_states] [live_brokers]
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64                       <-- Will be removed
  max_broker_epoch => INT64                   <-- New
  topic_states => topic [partition_states]
    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 V4

```
StopReplica Request => controller_id controller_epoch broker_epoch delete_partitions [topic_partitions]
  controller_id => INT32
  controller_epoch => INT32
  broker_epoch => INT64                       <-- Will be removed
  max_broker_epoch => INT64                   <-- New
  delete_partitions => BOOLEAN
  topic_partitions => topic [partition]
    partition => INT32
```

Proposed Changes

Replacing the per broker epoch with cluster-wide maximum broker epoch

Currently all requests sent from the controller to brokers contains a broker epoch field, which is introduced in KIP-380.

The broker epoch is used to differentiate controller requests meant to be sent to a particular epoch of a broker, so that

1. oblolete control requests can be rejected by a broker
2. obsolete ControlledShutdown requests can be fenced off by the controller
3. the controller can use the broker epoch to detect bounced brokers

Currently the controll request sent to different brokers have different broker epoch values. And each broker has a different the broker epoch field.

This field turns out to be the only field that's different between the UpdateMetadata requests sent due to one metadata change.

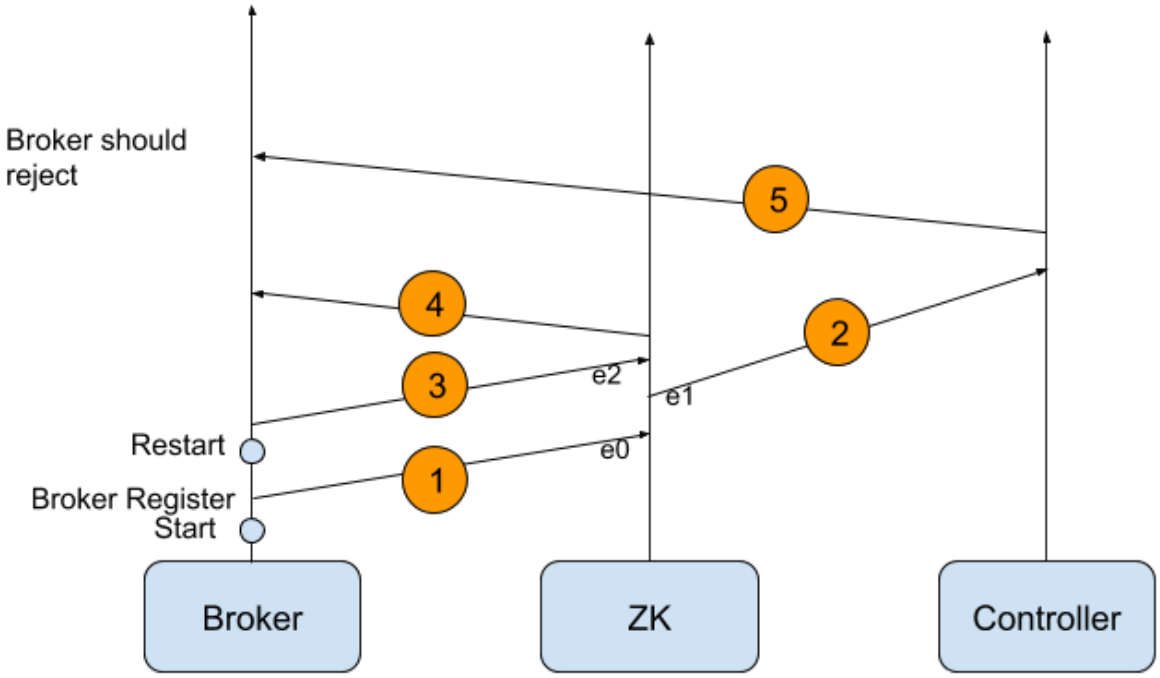
In order for all brokers to reuse the same UpdateMetadata request, we need a way to cache the same UpdateMetadata request payloads.

Several options are available:
- either we only cache the other fields, and treat the broker-epoch field differently
- we replace the per-broker broker-epoch field in the control request with some new field so that all brokers can use the same value.

Between these two approaches, we feel the latter is more elegant if the new mechanism works.

Requirement of the new field for broker side fencing

Let us examine the case where a broker needs to fence off an obsolete request from the controller in the following figure:



The vertical lines pointing upward represent the time lines for the broker, the zookeeper cluster and the controller respectively.

The process involves message passing between the three entities and the five messages have been labeled with circled numbers from 1 to 5.

Message 1: The broker has started and is sending a registration request to ZK when creating the /brokers/ids/<broker id> znode. On the ZK side, **e0** repre sents the event of receiving this registration request and creating the znode.

Message 2: Three steps happen due to the creation of the new znode

a. ZK notifies the controller that there is a broker change.

b. the controller sends a request to read all the brokers' endpoints and current epochs

c. ZK sends back the response containing the broker's epoch

Message 2 represents the response sent by ZK in step c, and **e1** represents the event of sending Message 2.

Message 3: The broker has restarted, and is sending a 2nd registration request to ZK, similar to Message 1. **e2** represents the event of receiving this 2nd registration request.

Message 4: ZK replies to the broker with the Zxid that created the broker's znode at /brokers/ids/<broker id>.

Message 5: The controller sends a control request to the broker, using a previously learned broker epoch that has become obsolete. Upon receiving the control request, the broker should have rejected it.

In order for this to work, ZK and the controller must work together to ensure Message 5 carries the info e0 e1 e2, where represents the "happens before" relationship.

The broker epoch field introduced in KIP-380 relies on the monotonically increasing property of zxid in Zookeeper: [1]

To represent the "happens before" relationship, we are still relying on the monotonically increasing property of zxid. At the time of event e1, the event e2 has not happened yet. So at the time of e1, the requirement e0 e1 e2 can be translated to zxid(e0) <= zxid reported at time of e1 < all future transaction events.

For instance, in the response message (Message 2), ZK could have reported its latest zxid, or it could have reported the value max(all brokers' zxids). There is an equivalent approach to ZK reporting the max(all brokers' zxids): ZK still reports the per-broker epochs in its reply in Message 2, while the controller calculates the max(all brokers' zxids) and uses the aggregated max in Message 5.

Using max broker epochs in control requests

The analysis above only considered the case of broker-side fencing:

1. oblolete control requests can be rejected by a broker

Since the per-broker epoch has worked well for the other two use cases

1. obsolete ControlledShutdown requests can be fenced off by the controller
2. the controller can use the broker epoch to detect bounced brokers

we propose to keep the current mechanism unchanged by maintaining the per-broker epoch in the controller's memory.

In summary, our proposed new approach works by

1. replace the per-broker epochs in control requests with a new field MaxBrokerEpoch, that represents the maximum of all brokers' zxids (epochs).
2. still maintain the per-broker epochs in the controller's memory to fence off obsolete broker requests, and detect bounced brokers

By doing that, we'd not only satisfy the safety requirements, but also unify the payload of UpdateMetadata requests sent to different brokers for the same metadata change.

More specifically, our proposed change is

1. we bump the request versions to introduce the new MaxBrokerEpoch field for the following 3 types of control requests: UpdateMetadata, StopReplica and LeaderAndIsr
2. when the IBP is above a certain value (to be filled in after the code change), the controller will start populating the MaxBrokerEpoch field instead of the BrokerEpoch field (representing the per-broker epoch), and reuse the same cached UpdateMetadata request for all brokers due to the same metadata change.

Compatibility, Deprecation, and Migration Plan

Adopting the change in this KIP requires a two pass deployment:

1. Deploy the code changes so that brokers can handle both the existing control requests (with the per-broker epoch field) as well as new control requests (with the max_broker_epoch field).
2. Upgrade the Inter-Broker-Protocol version and deploy the new config. Upgrading the Inter-Broker-Protocol version will cause the controller to use the new request format and reuse the same UpdateMetadata request.

Rejected Alternatives

We considered the following alternatives:

1. Cache all other fields in the UpdateMetadata request and treat the broker epoch field differently. This approach is rejected since treating the broker epoch field differently makes the code less elegant.
2. Retrieving the latest zxid from zookeeper and use it in control requests. This approach is not much different than the proposed solution. Since the controller still needs to retrieve and maintain the per-broker zxids anyway and it seems retrieving the latest zxid is not supported yet in the zookeeper Java client library, we have preferred to use the max(all brokers' zxids).