KIP-903: Replicas with stale broker epoch should not be allowed to join the ISR

Status

Current state: *"Accepted"*

Discussion thread: *here*

Vote thread*: here*

JIRA: ⚠ Unable to render Jira issues macro, execution error.

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

Motivation

The idea starts from the consequences of a broker reboot in the cloud environment. Let's say the broker reboots and some logs have not flushed from the page cache. When this happens, the data on this broker is lost. Normally, this is ok because the controller will remove a reboot broker from all the ISRs. Then the damage is limited and the broker can catch up later. However, if in the meantime, a leader of one of the partitions crashes as well, such a broker can become the partition leader and make the data loss a real one. Here is an example of such scenarios quoted from https://issues.apache.org/jira/browse/KAFKA-14139

Suppose we have a partition with two replicas: A and B. Initially A is the leader and it is the only member of the ISR.

1. Broker B catches up to A, so A attempts to send an AlterPartition request to the controller to add B to the ISR.
2. Before the AlterPartition request is received, replica B has a hard failure.
3. The current controller successfully fences broker B. It takes no action on this partition since B is already out of the ISR.
4. Suppose that replica B finishes startup, but the disk has been replaced (all of the previous states have been lost).
5. The new controller sees the registration from broker B first.
6. Finally, the AlterPartition from A arrives which adds B back into the ISR even though it has an empty log.

The controller and the leader need extra info on the ISR brokers to prevent such data loss cases.

Scope

This potential data loss case exists in both ZK mode and Kraft mode. Because we are moving forward to the Kraft mode, this proposal just fixes it in Kraft mode.

Proposed Design

The invariant to introduce:

- A replica with a stale broker epoch is not eligible to join the ISR, where the broker epoch is provided to the leader via the Fetch request.

When the controller receives an AlterPartition request, it has no idea whether the leader was aware of the latest broker status before creating the request. In an ideal situation, the controller should be able to reject the AlterPartition request if it notices the request is stale (stale means the ISR broker crashed before the request is accepted).

On the other hand, the broker epoch can be a good indicator of the stale AlterPartition request. When a broker restarts, it will first register with the controller to get a new broker epoch. This means that the controller will always be the first one to know the latest broker epochs which makes it ahead of the leaders. So, suppose the leader includes the broker epochs in the AlterPartition request based on its best knowledge, the controller can identify the stale request if the broker epochs mismatch with the latest.

The last piece to think about, the leader should learn the broker epoch from the follower fetch request instead of using the broker metadata directly. Reasons are:

- The ISR expansion is triggered by the Fetch requests. Using Fetch requests makes the expansion synchronized. Broker is online Fetch and catch up candidate for ISR expand with AlterPartition. The flow is straightforward and easy to maintain.
- Using broker metadata requires more careful thinking about potential issues. For example, if the last fetch request comes super late, the leader already knows the new broker epoch and sends the AlterPartition with the new one, then the rebooted broker can still be added to ISR.
- Also, as a precautions check, the leader will only attempt to include the broker in ISR if the broker epoch from the metadata cache matches the epoch from the Fetch request.

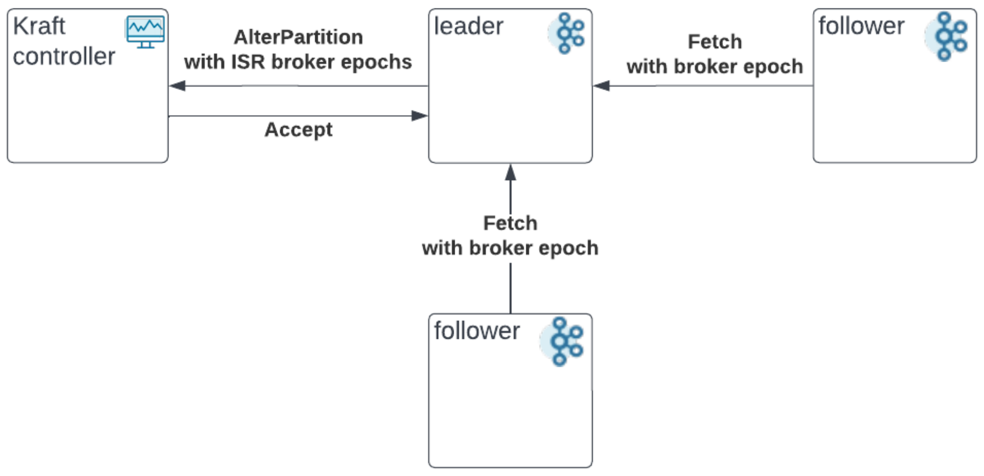
The new behaviors break down:

1. Follower
    - The follower will send its epoch id (received when registering) in the Fetch request.
2. Leader
    - The leader will keep track of the follower epochs received through the Fetch request. The epochs can be maintained in the ReplicaState.
    - The leader will include the proposed ISR follower epochs in the AlterPartition request. Also, the leader will verify these epochs against the metadata cache. Before the broker epochs from the Fetch request and the metadata cache are consistent, the leader will not propose to include the broker in the ISR.
    - The leader will discard the proposed ISR change and revert to the old committed ISR if the controller returns INELIGIBLE_REPLICA for the partition.
3. Controller
    - The controller will make sure all the broker epochs in the ISR match with the latest broker epochs before persisting this change. If not, it will reject the request with the error code INELIGIBLE_REPLICA.

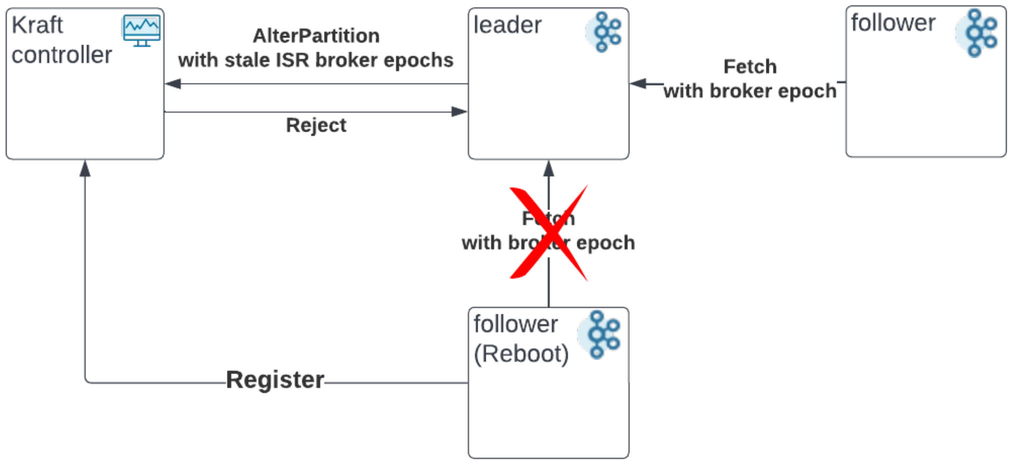
A notable exception to the proposal:

When the ISR only contains one last broker, if this broker crashes and gets fenced, the current behavior will keep it in the ISR. It is an exception to the proposal of keeping stale replicas out of the ISR. However, it is also a fair idea considering we do not have a good way to validate the broker state after the reboot or to elect the best leader in the replica set from the minimal data loss perspective. As the proposal does not focus on this last one in the ISR problem but tries to solve a more common problem of multiple replicas in the ISR, we will leave this behavior as it is.

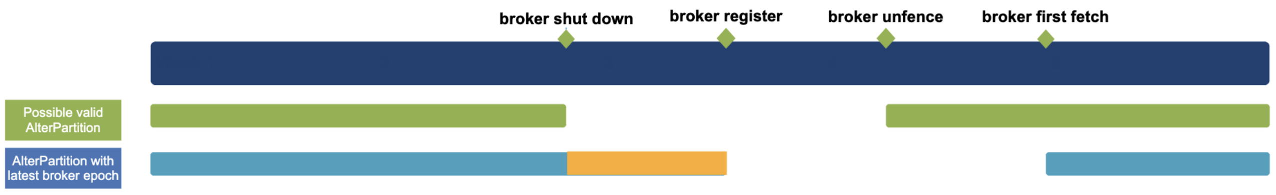
After the change:



When reboot has a race with the AlterParition request:



Timeline of valid AlterPartition requests. One of the ISR brokers reboots.



Note:

1. Ideally, the controller should only persist the AlterPartition requests received before the broker shut down and after the broker is unfenced(in green).
2. After the change, the controller can accept valid AlterPartition requests with the correct epoch info(in blue).
3. Depending on whether the shutdown is clean or not, the controller may still accept the AlterPartition request after the shutdown(in orange). However, it is fine, because the controller will remove the broker from all the ISRs when fencing the broker(after the registration).

Public Interfaces

AlterPartition RPC

The version of the AlterPartition API is bumped to version 3.

Request

```
{
  "apiKey": 56,
  "type": "request",
  "listeners": ["zkBroker", "controller"],
  "name": "AlterPartitionRequest",
  "validVersions": "0-3",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "BrokerId", "type": "int32", "versions": "0+", "entityType": "brokerId",
      "about": "The ID of the requesting broker" },
    { "name": "BrokerEpoch", "type": "int64", "versions": "0+", "default": "-1",
      "about": "The epoch of the requesting broker" },
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0-1", "ignorable": true, "entityType": "topicName",
        "about": "The name of the topic to alter ISRs for" },
      { "name": "TopicId", "type": "uuid", "versions": "2+", "ignorable": true,
        "about": "The ID of the topic to alter ISRs for" },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index" },
        { "name": "LeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The leader epoch of this partition" },
// Old Field deprecated.
        { "name": "NewIsr", "type": "[]int32", "versions": "0-2", "entityType": "brokerId",
          "about": "The ISR for this partition" },
// Old Field deprecated.
// New Field begin
        { "name": "NewIsrWithEpochs", "type": "[]BrokerState", "versions": "3+", "fields": [
            { "name": "BrokerId", "type": "int32", "versions": "3+", "entityType": "brokerId",
              "about": "The ID of the broker." },
               { "name": "BrokerEpoch", "type": "int64", "versions": "3+", "default": "-1",
              "about": "The epoch of the broker. It will be -1 if the epoch check is not supported." }
         ]},
// New Field End
        { "name": "LeaderRecoveryState", "type": "int8", "versions": "1+", "default": "0",
          "about": "1 if the partition is recovering from an unclean leader election; 0 otherwise." },
        { "name": "PartitionEpoch", "type": "int32", "versions": "0+",
          "about": "The expected epoch of the partition which is being updated. For legacy cluster this is the
ZkVersion in the LeaderAndIsr request." }
      ]}
    ]}
  ]
}
```

Fetch request RPC

```
{
  "apiKey": 1,
  "type": "request",
  "listeners": ["zkBroker", "broker", "controller"],
  "name": "FetchRequest",
  "validVersions": "0-15",
  "flexibleVersions": "12+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "12+", "nullableVersions": "12+", "default": "null",
      "taggedVersions": "12+", "tag": 0, "ignorable": true,
      "about": "The clusterId if known. This is used to validate metadata fetches prior to broker
registration." },

// Deprecate this field. It is now in the ReplicaState. Also change the default value to -1.
        { "name": "ReplicaId", "type": "int32", "versions": "0-14", "default": "-1",
          "entityType": "brokerId", "about": "The broker ID of the follower, of -1 if this request is from a
consumer." },
// Update ends

// New Field begin
        { "name": "ReplicaState", "type": "ReplicaState", "taggedVersions":"15+", "tag": 1, "fields": [
        { "name": "ReplicaId", "type": "int32", "versions": "15+", "default": "-1", "entityType": "brokerId",
          "about": "The replica ID of the follower, of -1 if this request is from a consumer." },
        { "name": "ReplicaEpoch", "type": "int64", "versions": "15+", "default": "-1",
          "about": "The epoch of this follower." }
        ]},
// New Field End

    { "name": "MaxWaitMs", "type": "int32", "versions": "0+",
      "about": "The maximum time in milliseconds to wait for the response." },
    { "name": "MinBytes", "type": "int32", "versions": "0+",
      "about": "The minimum bytes to accumulate in the response." },
    { "name": "MaxBytes", "type": "int32", "versions": "3+", "default": "0x7fffffff", "ignorable": true,
      "about": "The maximum bytes to fetch.  See KIP-74 for cases where this limit may not be honored." },
    { "name": "IsolationLevel", "type": "int8", "versions": "4+", "default": "0", "ignorable": true,
      "about": "This setting controls the visibility of transactional records. Using READ_UNCOMMITTED
(isolation_level = 0) makes all records visible. With READ_COMMITTED (isolation_level = 1), non-transactional
and COMMITTED transactional records are visible. To be more concrete, READ_COMMITTED returns all data from
offsets smaller than the current LSO (last stable offset), and enables the inclusion of the list of aborted
transactions in the result, which allows consumers to discard ABORTED transactional records" },
    { "name": "SessionId", "type": "int32", "versions": "7+", "default": "0", "ignorable": true,
      "about": "The fetch session ID." },
    { "name": "SessionEpoch", "type": "int32", "versions": "7+", "default": "-1", "ignorable": true,
      "about": "The fetch session epoch, which is used for ordering requests in a session." },
    { "name": "Topics", "type": "[]FetchTopic", "versions": "0+",
      "about": "The topics to fetch.", "fields": [
      { "name": "Topic", "type": "string", "versions": "0-12", "entityType": "topicName", "ignorable": "true",
        "about": "The name of the topic to fetch." },
      { "name": "TopicId", "type": "uuid", "versions": "13+", "ignorable": true, "about": "The unique topic
ID"},
      { "name": "Partitions", "type": "[]FetchPartition", "versions": "0+",
        "about": "The partitions to fetch.", "fields": [
        { "name": "Partition", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "CurrentLeaderEpoch", "type": "int32", "versions": "9+", "default": "-1", "ignorable": true,
          "about": "The current leader epoch of the partition." },
        { "name": "FetchOffset", "type": "int64", "versions": "0+",
          "about": "The message offset." },
        { "name": "LastFetchedEpoch", "type": "int32", "versions": "12+", "default": "-1", "ignorable": false,
          "about": "The epoch of the last fetched record or -1 if there is none"},
        { "name": "LogStartOffset", "type": "int64", "versions": "5+", "default": "-1", "ignorable": true,
          "about": "The earliest available offset of the follower replica.  The field is only used when the
request is sent by the follower."},
        { "name": "PartitionMaxBytes", "type": "int32", "versions": "0+",
          "about": "The maximum bytes to fetch from this partition.  See KIP-74 for cases where this limit may
not be honored." }
      ]}
    ]},
    { "name": "ForgottenTopicsData", "type": "[]ForgottenTopic", "versions": "7+", "ignorable": false,
      "about": "In an incremental fetch request, the partitions to remove.", "fields": [
      { "name": "Topic", "type": "string", "versions": "7-12", "entityType": "topicName", "ignorable": true,
        "about": "The topic name." },
      { "name": "TopicId", "type": "uuid", "versions": "13+", "ignorable": true, "about": "The unique topic
ID"},
      { "name": "Partitions", "type": "[]int32", "versions": "7+",
        "about": "The partitions indexes to forget." }
    ]},
    { "name": "RackId", "type":  "string", "versions": "11+", "default": "", "ignorable": true,
      "about": "Rack ID of the consumer making this request"}
  ]
}
```

Note: The replica id is moved to ReplicaState because we can benefit from reducing the consumer fetch request message size. The ReplicaState is a tagged field and will be the default value if it is from the consumers.

Compatibility, Deprecation, and Migration Plan

1. As the AlterPartition and Fetch requests are shared between ZK and Kraft mode, we can just let the ZK controller ignore the broker epoch field.
2. The AlterPartition request will be protected by the ApiVersion bump. The brokers can use the controller-supported version accordingly.

3. The metadata version/IBP version will be bumped to gate the new version of the Fetch request. After the IBP is upgraded, the new Fetch quest (version 15) will be used.

Test Plan

Will test by simulating the race scenario of the delayed AlterPartitionRequest and the broker reboot.

Rejected Alternatives

Bump all the partition leader epochs when a broker registers.

The idea is straightforward, when a broker registers, we can bump the leader epochs for all the partitions that include this broker. The bumped leader epoch can ensure the controller can identify and reject the stale AlterPartition request.

The downside of this approach is that it will interrupt the current ongoing replications because they need to wait to sync the new leader epoch.

Using the broker epoch from the metadata instead of using the Fetch request

Using the broker epoch may still have the risk that the leader sending an AlterPartition request before receiving the updated broker epoch, as explained in the design.

On the other hand, the Fetch request directly reflects the status of the partition from the leader's perspective. The epoch may not be updated in time, but it will definitely fall behind the controller register which guarantees the stale AlterPartition request rejection.

Smart leader election when we lost the last replica in ISR

This is not necessarily an alternative to solve the problem but an edge case handling when a broker reboots. As mentioned above, if we lose the last replica in ISR, we do not remove the replica because there is no safe replica to be elected. This becomes an exception to the proposal.

It could be a good topic to go further if we can determine the data damage after the reboot, then the controller is able to choose a leader that has the least data loss or even no loss at all.