KIP-347: Enable batching in FindCoordinatorRequest

Status

Current state: Superseeded by KIP-699

Discussion thread: *here* *[Change the link from the KIP proposal email archive to your own email thread]*

JIRA: *KAFKA-7206*

Motivation

Today each *FindCoordinatorRequest* only contains a single consumer id at a time. That means if we have *N* number of groupIds that need to find their coordinator, we would have to send out *N* number of requests. This can be optimized by batching up the groupIds and send out fewer requests. Therefore we want to extend *FindCoordinatorRequest* to have multiple consumer ids.

Furthermore, this is the foundation of some later possible optimizations(enable batching in *describeConsumerGroups*, batching in *deleteConsumerGroups*). For example, now in KafkaAdmin, we are sending *describeConsumerGroups* and *deleteConsumerGroups* with one pair of *coordinatorId* and *groupId* at a time. If we enable batching in *FindCoordinatorRequest*, we can group all the *groupId* with the same *coordinatorId,* and only need to send out one request for those with the same *coordinatorId.*

Proposed Changes

Send a single *FindCoordinatorRequest* to a broker asking for coordinators of all consumer groups. These are the changes that need to be made:

1. Update the *Schema* in *FindCoordinatorRequest* to store an array of groupID (with a new *FIND_COORDINATOR_REQUEST_V3*); modify the class structure to take more than one *coordinatorKey* (*List*<*String*> *groupIds*).
2. Update FindCoordinatorResponse 's class to have a map structure (now it only has Node to support one result, we need to support multiple results so something like <GroupId, CoordinatorNodeMetadata>)
3. Update handleFindCoordinatorRequest to correctly parse the new FindCoordinatorRequest and map the result to the new *FindCoordinatorResponse*.

Public Interfaces

API Changes

FindCoordinatorRequest.java

I suggest that we modify *FindCoordinatorRequest.java* to support multiple coordinator keys by adding a new *FIND_COORDINATOR_REQUEST_V3*:

```
public class FindCoordinatorRequest extends AbstractRequest {
    private static final String COORDINATOR_KEY_KEY_NAME = "coordinator_key";
    private static final String COORDINATOR_TYPE_KEY_NAME = "coordinator_type";
        private static final String COORDINATOR_GROUPIDS_KEY_NAME = "coordinator_groupIds"


        private static final Schema FIND_COORDINATOR_REQUEST_V0 = new Schema(GROUP_ID);

        private static final Schema FIND_COORDINATOR_REQUEST_V1 = new Schema(
        new Field("coordinator_key", STRING, "Id to use for finding the coordinator (for groups, this is the
groupId, " +
```

```
                        "for transactional producers, this is the transactional id)"),
        new Field("coordinator_type", INT8, "The type of coordinator to find (0 = group, 1 = transaction)"));


        /**
         * The version number is bumped to indicate that on quota violation brokers send out responses before
throttling.
         */
        private static final Schema FIND_COORDINATOR_REQUEST_V2 = FIND_COORDINATOR_REQUEST_V1;


        private static final Schema FIND_COORDINATOR_REQUEST_V3 = new Schema(
        new Field(COORDINATOR_GROUPIDS_KEY_NAME, new ArrayOf(STRING), "All the coordinator ids " +
        "for this request"));


        public static Schema[] schemaVersions() {
            return new Schema[] {FIND_COORDINATOR_REQUEST_V0, FIND_COORDINATOR_REQUEST_V1,
FIND_COORDINATOR_REQUEST_V2, FIND_COORDINATOR_REQUEST_V3};
        }

        ...

    public static class Builder extends AbstractRequest.Builder<FindCoordinatorRequest> {
       private final Map<String, String> groupIds; // Use to store (coordinator_key, coordinator_type) pair
       private final short minVersion;

       public Builder(CoordinatorType coordinatorType, String coordinatorKey) {
          super(ApiKeys.FIND_COORDINATOR);
          groupIds.put(coordiateType, coordinatorKey);
          this.minVersion = coordinatorType == CoordinatorType.TRANSACTION ? (short) 1 : (short) 0;
       }

      @Override
      public FindCoordinatorRequest build(short version) {
          if (version < minVersion)
            throw new UnsupportedVersionException("Cannot create a v" + version + " FindCoordinator request " +
                    "because we require features supported only in " + minVersion + " or later.");
          return new FindCoordinatorRequest(coordinatorType(), coordinatorKey(), version);
      }

      public List<FindCoordinatorRequest> buildToList(short version){
          if (version < minVersion)
              throw new UnsupportedVersionException("Cannot create a v" + version + " FindCoordinator request "
+
                      "because we require features supported only in " + minVersion + " or later.");
          return Collections.singletonList(new FindCoordinatorRequest(coordinatorType, coordinatorKey,
version));
      }
          public String coordinatorKey() {
             return groupIds.keySet().stream().findFirst().get();
          }

      public CoordinatorType coordinatorType() {
        return groupIds.get(coordinatorKey());
      }


     public Map groupIds(){
        return groupIds;
     }
    }


    ...
          private final Map<String, String> groupIds; // Use to store (coordinator_key, coordinator_type) pair


        ...
```

FindCoordinatorResponse.java

We also want to modify *FindCoordinatorResponse.java* to support multiple coordinator keys by adding a new *FIND_COORDINATOR_REQUEST_V3*:

```
public class FindCoordinatorResponse extends AbstractResponse {
    private static final String COORDINATOR_KEY_NAME = "coordinator";

    // coordinator level field names
    private static final String NODE_ID_KEY_NAME = "node_id";
    private static final String HOST_KEY_NAME = "host";
    private static final String PORT_KEY_NAME = "port";

    private static final Schema FIND_COORDINATOR_BROKER_V0 = new Schema(
            new Field(NODE_ID_KEY_NAME, INT32, "The broker id."),
            new Field(HOST_KEY_NAME, STRING, "The hostname of the broker."),
            new Field(PORT_KEY_NAME, INT32, "The port on which the broker accepts requests."));

    private static final Schema FIND_COORDINATOR_RESPONSE_V0 = new Schema(
            ERROR_CODE,
            new Field(COORDINATOR_KEY_NAME, FIND_COORDINATOR_BROKER_V0, "Host and port information for the
coordinator " +
                    "for a consumer group."));

    private static final Schema FIND_COORDINATOR_RESPONSE_V1 = new Schema(
            THROTTLE_TIME_MS,
            ERROR_CODE,
            ERROR_MESSAGE,
            new Field(COORDINATOR_KEY_NAME, FIND_COORDINATOR_BROKER_V0, "Host and port information for the
coordinator"));


    private static final Schema FIND_COORDINATOR_RESPONSE_V3 = new Schema(
            THROTTLE_TIME_MS,
                        new ArrayOf(
                                new Schema(
                                        GROUP_ID,
                                ERROR_CODE,
                                      ERROR_MESSAGE,
                                new Field(COORDINATOR_KEY_NAME, FIND_COORDINATOR_BROKER_V0, "Host and port
information for the coordinator"));));

    /**
     * The version number is bumped to indicate that on quota violation brokers send out responses before
throttling.
     */
    private static final Schema FIND_COORDINATOR_RESPONSE_V2 = FIND_COORDINATOR_RESPONSE_V1;

    public static Schema[] schemaVersions() {
        return new Schema[] {FIND_COORDINATOR_RESPONSE_V0, FIND_COORDINATOR_RESPONSE_V1,
FIND_COORDINATOR_RESPONSE_V2, FIND_COORDINATOR_RESPONSE_V3};
    }


        ...


        private final int throttleTimeMs;
        private final String errorMessage;
        private final Errors error;
        private final Map<String, Node> nodes; //store (groupID, Node)


        ...
```

KafkaApis.scala/handleFindCoordinatorRequest

```
def handleFindCoordinatorRequest(request: RequestChannel.Request) {
  val findCoordinatorRequest = request.body[FindCoordinatorRequest]

  val groups = describeRequest.groupIds.asScala.map { groupId =>
        ...
        // find the topicMetadata for each one
  }
    ...
  def createResponse(requestThrottleMs: Int): AbstractResponse = {
        // Map the coordinator ID to group Ids
        }
    ...
        sendResponseMaybeThrottle(request, createResponse)
}
```

Compatibility, Deprecation, and Migration Plan

- *FindCoordinatorResponse and FindCoordinatorRequests need to update to V3.*
- *Requests that contains a single groupID still can be supported.*
- *Compatibility issues between old and new versions need to be considered, we should think about how to convert requests from a newer version to a old version.*

One-to-many mapping

Today, converting from a higher version API to a lower version, is through the *apiVersions* in *NetworkClient.java* to map the API version number for each node, and then use each request's *builder.build()* to build a lower version request schema. This conversion is a **one-to-one** mapping, meaning **ONE** higher version request gets convert to **ONE** lower version request.

However, this KIP will change this pattern, because it introduces a batching optimization in the requests, meaning combining **MANY** requests into **ONE** request. So we need support a **one-to-many** backward compatible conversion for requests along with the implementation of this KIP.

This can be implemented in *NetworkClient.java doSend(),* in doSend(), it call *builder.build()* to build ONE request and send it out. What we need to do is modify this logic to something similar:

One tricky question is, how do we know if a higher version API has a batching optimization. Discussions are still required and suggestions are highly encouraged.

a) One solution is to let the request's *builder* return either **ONE** request or a **LIST** of requests. This is backward compatible. We can have a list of one single element.

For example, in *FindCoordinatorRequest.java, builder* will have a new *buildToList()* (Should come up with a better name) method like this:

```
public List<FindCoordinatorRequest> buildToList(short version) {
    if (version < minVersion)
        throw new UnsupportedVersionException("Cannot create a v" + version + " FindCoordinator request " +
                "because we require features supported only in " + minVersion + " or later.");
    return groupIds.entrySet()
            .stream()
            .map(x-> new FindCoordinatorRequest(x.value(), x.key(), version))
            .collect(Collectors.toList());
}
```

Then in *NetworkClient.java*/*doSend()*, use *instanceof* to check if batching has been used.

```
private void doSend(ClientRequest clientRequest, boolean isInternalRequest, long now) {
...
try {
    NodeApiVersions versionInfo = apiVersions.get(nodeId);
    short version;
    // Note: if versionInfo is null, we have no server version information. This would be
    // the case when sending the initial ApiVersionRequest which fetches the version
    // information itself.  It is also the case when discoverBrokerVersions is set to false.
    if (versionInfo == null) {
        version = builder.latestAllowedVersion();
        if (discoverBrokerVersions && log.isTraceEnabled())
            log.trace("No version information found when sending {} with correlation id {} to node {}. " +
                    "Assuming version {}.", clientRequest.apiKey(), clientRequest.correlationId(), nodeId,
version);
    } else {
        version = versionInfo.latestUsableVersion(clientRequest.apiKey(), builder.oldestAllowedVersion(),
                builder.latestAllowedVersion());
    }
 // Check if there is batching optimization, if there is, break them down and send them out one by one:
 // We can use an enum to list all the batch-optimized API here.
if (builder.apiKey() == FIND_COORDINATOR){
    for (AbstractRequest request: ((FindCoordinatorRequest.Builder) builder).buildToList(version)){
        doSend(clientRequest, isInternalRequest, now, request);
    }
} else {
    doSend(clientRequest, isInternalRequest, now, builder.build(version));
}
 ...
}
```

Rejected Alternatives

- Update the batching logic in *KafkaAdmin*.java directly instead of modifying the structure of requests and response.
- Change the builder.build()'s return type to a *Collection*;