KIP-240: AdminClient.listReassignments AdminClient.describeReassignments

Status

Current state: Under Discussion

Discussion thread: *here*

JIRA: *KAFKA-6379*

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

Motivation

This proposal builds on the proposal in KIP-236, which allows individual partition reassignments to be identified.

Currently the AdminClient has no visibility of the partition reassignments occurring in a Kafka cluster. It would be valuable for this information to be available in the AdminClient because there will eventually be an API for partition reassignment in the AdminClient, and knowing about the current reassignments is important when creating more.

Public Interfaces

AdminClient.listReassignments()

```
AdminClient {
    /**
     * List the current reassignments affecting the given {@code partitions}
     * or all current reassignments if the given {@code partitions} is null.
     * This requires describe access to the Cluster.
     */
    ListReassignmentsResult listReassignments(Collection<TopicPartition> partitions);
    ListReassignmentsResult listReassignments(Collection<TopicPartition> partitions,
      ListReassignmentsOptions options);
}
```

Where:

```
public class ListReassignmentsOptions extends AbstractOptions<ListReassignmentsObjects> {}

/** The result of {@link AdminClient#listReassignments()} */
public class ListReassignmentsResult {
    public KafkaFuture<Collection<Reassignment>> all();
    public KafkaFuture<Reassignment> get(TopicPartition partition);
}

/**
 * Identifies the reassignment of a topic partition to some (new) set
 * of brokers.
 */
class Reassignment {
    // implictly this contains the czxid of the reassignment
    // but this is never exposed to clients
    /** The partition being reassigned */
    public TopicPartition topicPartition() { ... }
    public boolean equals(Object other) { ... }   // identity based on czxid
    public int hashCode() { ... } // identity based on czxid
    public String toString() { ... }
}
```

Network Protocol: ListReassignmentsRequest

A `ListReassignmentsRequest` must be sent to the controller of a cluster.

```
ListReassignmentsRequest => timeout_ms [topic_partition]
  timeout_ms => INT32
  topic_partition => topic [partition_id]
    topic => STRING
    partition_id => INT32
```

Network Protocol: ListReassignmentsResponse

```
ListReassignmentsResponse => throttle_time_ms, error_code error_message [reassignment]
  throttle_time_ms => INT32
  error_code => INT16
  error_message => NULLABLE_STRING
  reassignment => topic [topic_reassignments]
    topic => STRING
    topic_reassignments => partition reassignment_id
      partition_id => INT32
      reassignment_id => INT64
```

Where:

- `throttle_time_ms` is the throttle time
- `error_code` is an error code
- `error_message` is an error message
- `topic` is a topic name
- `partition_id` is a partition id
- `reassignment_id` identifies a reassignment znode

Possible errors include:

- `CLUSTER_AUTHORIZATION_FAILED` if the client didn't have describe on the cluster
- `NOT_CONTROLLER` if the request was sent to a broker that was not the controller
- `UNKNOWN_TOPIC_OR_PARTITION` if the partition in the request does not exist

AdminClient.describeReassignments()

```
AdminClient {
    /**
     * Get the status of the given {@code reassignments}.
     * This requires describe access to the Cluster.
     */
    DescribeReassignmentsResult describeReassignments(
      Collection<Reassignment> reassignments);
    DescribeReassignmentsResult describeReassignments(
      Collection<Reassignment> reassignments, DescribeReassignmentsOptions options);
}
```

Where `Reassignment` is one returned from `AdminClient.listReassignments()` and the other classes are as follows:

```
public class DescribeReassignmentsOptions extends AbstractOptions<DescribeReassignmentsOptions> { }

/** The result of {@link AdminClient#describeReassignments()} */
public class DescribeReassignmentsResult {
   /**
    * Get (a future for) the description of the given reassignment, or
    * null if the given reassignment was no longer running at time
    * the controller processed the
    * {@link AdminClient#describeReassignments()} call.
    *
    * If the given {@code reassignment} was not given in the call to
    * {@link AdminClient#describeReassignments()} the future will throw
    * NoSuchElementException.
    */
    public KafkaFuture<ReassignmentDescription> get(Reassignment reassignment);

    /**
     * The current reassignments. This is only useful when
     * {@link AdminClient#describeReassignments()} was called with a null
     * {@code reassignments} argument.
     */
    public KafkaFuture<Collection<Reassignment>> reassignments();
}

/** Describes a reassignment */
public class ReassignmentDescription {
    /**
     * The reassignment that this description is describing.
     */
    public Reassignment reassignment() { ... }

    /**
     * The approximate time (as an offset from the unix epoch) that the reassignment
     * was started. This will not change over the life of this reassignment.
     */
    public long startTime() { ... }

    /**
     * The id of the broker that's currently leading the partition
     *
     * It is possible for this value of change over the
     * life of the reassignment if the leader changes.
     */
    public int currentLeader() { ... }

    /**
     * The throttle currently applying to the leader for this partition.
     *
     * It is possible for this value of change over the
     * life of the reassignment if the reassignment is changed, or if the leader changes.
     */
    public long leaderThrottle() { ... }

    // In the future this might also include information about the throttle(s)
    // for the reassignment
    /**
     * The brokers which will maintain a replica after this reassignment
     * is complete. The first broker in the list is the preferred leader.
     * When the preferred broker is in sync it will be elected leader of the partition
     * if the {@code auto.leader.rebalance.enable} broker config is set, or
     * when electPreferredLeader() is invoked.
     *
     * It is possible for this list of change over the
     * life of the reassignment if the reassignment is changed.
     */
    public List<Integer> newAssignedBrokers() { ... }

    /**
     * A map from newly assigned brokers to the corresponding throttle for that broker.
     * The keyset of this map is precisely {@link #newAssignedBrokers()}. If
     * a broker is not throttled, its throttle will be {@link Long#MAX_VALUE}.
     *
     * It is possible for this map of change over the
     * life of the reassignment if the reassignment is changed.
     */
    public Map<Integer, Long> newAssignedThrottles() { ... }
}
```

Having obtained a `ReassignmentDescription` a client can determine the LEO of the replicas on each of the newly assigned brokers by calling `AdminClient.describeLogDirs()`. It is not possible to include this in the `ReassignmentDescription` itself, this this information is not available to the controller.

The start time is provided to determine how long the reassignment has been in progress.

Network Protocol: DescribeReassignmentsRequest

A `DescribeReassignmentsRequest` must be sent to the controller of a cluster.

```
DescribeReassignmentsRequest => [topic_reassignment] timout_ms
  topic_reassignment => topic reassignment
    topic => STRING
    reassignment => partition_id reassigment_id
      partition_id => INT32
      reassignment_id => INT64
  timeout_ms => INT32
```

Network Protocol: DescribeReassignmentsResponse

```
DescribeReassignmentsResponse => throttle_time_ms error_code error_message [description]
  throttle_time_ms => INT32
  error_code => INT16
  error_message => NULLABLE_STRING
  description => reassignment_id start_time leader_id leader_throttle [new_assigned]
    start_time => INT64
    leader_id => INT32
    leader_throttle => INT64
    new_assigned => broker_id follower_throttle
      broker_id => INT32
      follower_throttle => INT64
```

Where:

- `throttle_time_ms` is the throttle time
- `error_code` is an error code
- `error_message` is an error message
- `reassignment_id` identifies the reassignment znode, obtained from a previous `ListReassignmentsRequest`.
- `start_time` is the time the reassignment started
- `leader_id` is the id of the broker that is currently leader of the partition
- `leader_throttle` is the throttle currently applying to the leader
- `broker_id` is the id of a broker in the new assignment
- `follower_throttle` is the throttle currently applying to the corresponding `broker_id`

Possible errors include:

- `CLUSTER_AUTHORIZATION_FAILED` if the client didn't have describe on the cluster
- `NOT_CONTROLLER` if the request was sent to a broker that was not the controller

Notes:

- If a `reassignment_id` provided in the DescribeReassignmentRequest does not match the current reassignment then that partition will be omitted from the DescribeReassignmentsResponse, rather than being an error.

Proposed Changes

Assuming the work proposed in KIP-236, the controller-side implementation when handling a `ListPartitionsRequest` is as follows:

1. The controller queries zookeeper to obtain the information about the reassignment for the given partitions from `/admin/reassginments/$topic/$partition`, obtaining the data (which contains the current new assigned partitions), and the zookeeper `Stat`, which includes the `czxid`. We use the `czxid` as the `reassignment_id`.

The controller-side implementation when handling a `DescribePartitionsRequest` is as follows:

1. The controller queries zookeeper to obtain the information about the reassignment for the given partitions from `/admin/reassginments/$topic/$partition`, obtaining the data (which contains the current new assigned partitions), and the zookeeper `Stat`, which includes the `czxid` and `ctime`.
    a. If the znode does not exist, or the `czxid` of the znode does not match the `reassignment_id` in the request the client has a reference to a now-completed reassignment and not information about this reassignment is returned.
    b. Otherwise:
        i. We obtain throttle information from ZooKeeper topic and broker config
        ii. We use the `ctime` as the `start_time.`

Compatibility, Deprecation, and Migration Plan

This is a new API, so has no impact on existing users.

Rejected Alternatives

1. Another method on the AdminClient also makes sense: Getting the current `ReassignmentDescription`s for a given `Collection<TopicPartition>` (rather than for a given `Collection<Reassignment>` as proposed here) The subtly with such a method is that it obscures the distinction between two successive reassignments: When invoked successively with the same arguments it might look like the same reassignment was in progress, but in fact the results could be referring to two separate reassignments. The proposed API makes this distinction more clear, but requires obtaining a Reassignment before getting its description. Implementing this KIP does not preclude implementing such a method in the future.
2. If we don't want to directly expose ZooKeeper's czxid information in the responses we could change the format of the request data stored in zookeeper to include a UUID allocated on the controller when it first creates the znode. The drawback of this is the overhead of transfering this UUID to/from ZK and the storage overhead in ZK.