KIP-179 - Change ReassignPartitionsCommand to use AdminClient

Note this was initially erroneously assigned as KIP-178, which was already taken, and has been reassigned KIP-179.

Status

Current state: Withdrawn

Discussion thread: *here (when initially misnumbered as KIP-178)* and here (when assigned KIP-179)

JIRA: *here*

Unable to render Jira issues macro, execution error.

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).e

Motivation

- The AdminClient API currently lacks any functionality for reassigning partitions: Users have to use the `kafka-reassign-partitions.sh` tool (`ReassignPartitionsCommand`) which talks directly to ZooKeeper. This prevents the tool being used in deployments where only the brokers are exposed to clients (i.e. where the zookeeper servers are intentionally not exposed). In addition, there is a general push to refactor/rewrite /replace tools which need ZooKeeper access with equivalents which use the `AdminClient` API.
- `ReassignPartitionsCommand` currently has no proper facility to report progress of a reassignment; While `--verify` can be used periodically to check whether the request assignments have been achieved the tool provides no means of knowing how quickly new replicas are catching up. It would be useful if the tool could report progress better.
- `ReassignPartitionsCommand`, when used with a replication throttle, requires a `--verify` invocation when the reassignment has finished in order to remove the throttle. So there exists the possibility that throttles are not removed after reassignment, with negative consequences for the performance of the cluster. It would be better if throttles could be removed automatically.

Public Interfaces

The `AdminClient` API will have new methods added (plus overloads for options):

- `reassignPartitions(Map<TopicPartition, List<Integer>>)`
- `alterInterbrokerThrottledRates(Map<Integer, ThrottledRate> throttledRates)`

- `alterInterbrokerThrottledReplicas(Map<TopicPartition, ThrottledReplicas> replicas)`

The options for `reassignPartitions()` will support setting a throttle, and a flag for its automatic removal at the end of the reassignment. Likewise the options for changing the throttled rates and replicas will include the ability to have the throttles automatically removed.

New network protocol APIs will be added to support these AdminClient APIs

- `ReassignPartitionsRequest` and `ReassignPartitionsResponse`
- `AlterInterbrokerThrottledRatesRequest` and `AlterInterbrokerThrottledRatesResponse`
- `AlterInterbrokerThrottledReplicasRequest` and `AlterInterbrokerThrottledReplicasResponse`

The options accepted by `kafka-reassign-partitions.sh` command will change:

- `--zookeeper` will be deprecated, with a warning message
- a new `--bootstrap-server` option will be added
- a new `--progress` action option will be added

When run with `--bootstrap-server` it will no longer be necessary to run `kafka-reassign-partitions.sh --verify` to remove a throttle: This will be done automatically.

Proposed Changes

Summary of use cases

Use case	command	AdminClient
Change replication factor	*kafka-reassign-partitions --execute --reassignment-json-file J*	`reassignPartitions()`
Change partition assignment	kafka-reassign-partitions --execute --reassignment-json-file *J*	`reassignPartitions()`
Change partition assignment with throttle	kafka-reassign-partitions --execute --reassignment-json-file *J* --throttle *R*	`reassignPartitions()` // with throttle option
Change throttled rate	kafka-reassign-partitions --execute --reassignment-json-file *J* --throttle *R*	alterInterbrokerThrottledRates() alterInterbrokerThrottledReplicas() **`// TODO how to do this conveniently?`**
Check progress of a reassignment	kafka-reassign-partitions --progress --reassignment-json-file *J*	`describeReplicaLogDirs()` (see KIP-113)

Check result and clear throttle	kafka-reassign-partitions --verify --reassignment-json-file *J*	`reassignPartitions(validateOnly)`
		
		// TODO checks none in progress, doesn't confirm states match

`kafka-reassign-partitions.sh` and `ReassignPartitionsCommand`

The `--zookeeper` option will be retained and will:

1. Cause a deprecation warning to be printed to standard error. The message will say that the `--zookeeper` option will be removed in a future version and that `--bootstrap-server` is the replacement option.
2. Perform the reassignment via ZooKeeper, as currently.

A new `--bootstrap-server` option will be added and will:

1. Perform the reassignment via the AdminClient API (described below) using the given broker(s) as bootstrap brokers.
2. When used with `--execute` and `--throttle`, the throttle will be an auto-removed one.

Using both `--zookeeper` and `--bootstrap-server` in the same command will produce an error message and the tool will exit without doing the intended operation.

It is anticipated that a future version of Kafka would remove support for the `--zookeeper` option.

A new `--progress` action option will be added. This will only be supported when used with `--bootstrap-server`. If used with `--zookeeper` the command will produce an error message and the tool will exit without doing the intended operation. `--progress` will report on the synchronisation of each of the partitions and brokers in the reassignment given via the `--reassignment-json-file` option

For example:

```
# If the following command is used to start a reassignment
bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9878 \
  --reassignment-json-file expand-cluster-reassignment.json \
  --execute

# then the following command will print the progress of
# that reassignment, then exit immediately
bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9878 \
  --reassignment-json-file expand-cluster-reassignment.json \
  --progress
```

That might print something like the following:

```
Topic      Partition  Broker  Status
-------------------------------------
my_topic   0          0       In sync
my_topic   0          1       Behind: 10456 messages behind
asdf       0          1       Unknown topic
my_topic   42         1       Unknown partition
my_topic   0          42      Unknown broker
my_topic   1          0       Broker does not host this partition
```

The implementation of `--progress` will make use of the `describeReplicaLogDirs()` method from KIP-113 to find the lag of the syncing replica.

Internally, the `ReassignPartitionsCommand` will be refactored to support the above changes to the options. An interface will abstract the commands currently issued directly to zookeeper.

There will be an implementation which makes the current calls to ZooKeeper, and another implementation which uses the `AdminClient` API described below.

In all other respects, the public API of `ReassignPartitionsCommand` will not be changed.

AdminClient: reassignPartitions()

Notes:

- This API is asynchronous in the sense that the client cannot assume that the request is complete (or the request was rejected) once they have obtained the result for the topic from the `ReassignPartitionsResult`.
- The `describeReplicaLogDirs()` method from KIP-113 can be used to determine progress.
- A call to `reassignPartitions()` with the `validateOnly` option can be used to determine whether a reassignment is currently running, and therefore whether the last reassignment has finished.

```
/**
 * <p>Reassign the partitions given as the key of the given <code>assignments</code> to the corresponding
 * list of brokers. The first broker in each list is the one which holds the "preferred replica".</p>
 *
 * <p>Inter-broker reassignment causes significant inter-broker traffic and can take a long time
 * in order to copy the replica data to brokers. The given options can be used impose a quota on
 * inter-broker traffic for the duration of the reassignment so that client-broker traffic is not
 * adversely affected.</p>
 *
 * <h3>Preferred replica</h3>
 * <p>When brokers are configured with <code>auto.leader.rebalance.enable=true</code>, the broker
 * with the preferred replica will be elected leader automatically.
 * <code>kafka-preferred-replica-election.sh</code> provides a manual trigger for this
 * election when <code>auto.leader.rebalance.enable=false</code>.</p>
 *
 * @param assignments The partition assignments.
 * @param options The options to use when reassigning the partitions
 * @return The ReassignPartitionsResult
 */
public abstract ReassignPartitionsResult reassignPartitions(Map<TopicPartition, List<Integer>> assignments,
                                                            ReassignPartitionsOptions options);
```

Where:

```
public class ReassignPartitionsOptions extends AbstractOptions<ReassignPartitionsOptions> {

    // Note timeoutMs() inherited from AbstractOptions

    public boolean validateOnly()

    /**
     * Validate the request only: Do not actually trigger replica reassignment.
     */
    public ReassignPartitionsOptions validateOnly(boolean validateOnly)

    public long throttle() {
        return throttle;
    }

    /**
     * <p>Set the throttle rate and throttled replicas for the reassignments.
     * The given throttle is in bytes/second and should be at least 1 KB/s.
     * Interbroker replication traffic will be throttled to approximately the given value.
     * Use Long.MAX_VALUE if the reassignment should not be throttled.</p>
     *
     * <p>A positive throttle is equivalent to setting:</p>
     * <ul>
     *     <li>The leader and follower throttled rates to the given value given by throttle.</li>
     *     <li>The leader throttled replicas of each topic in the request to include the existing brokers having
     *     replicas of the partitions in the request.</li>
     *     <li>The follower throttled replicas of each topic in the request to include the new brokers
     *     for each partition in that topic.</li>
     * </ul>
     *
     * <p>The value of {@link #autoRemoveThrottle()} will determine whether these
     * throttles will be removed automatically when the reassignment completes.</p>
     *
     * @see AdminClient#alterInterbrokerThrottledRate(int, long, long)
     * @see AdminClient#alterInterbrokerThrottledReplicas(Map)
     */
    public ReassignPartitionsOptions throttle(long throttle) { ... }

    public boolean autoRemoveThrottle() { ... }

    /**
     * True to automatically remove the throttle at the end of the current reassignment.
     */
    public ReassignPartitionsOptions autoRemoveThrottle(boolean autoRemoveThrottle) { ... }
}

public class ReassignPartitionsResult {
    public Map<TopicPartition, KafkaFuture<Void>> values();
    public KafkaFuture<Void> all();
}
```

Network Protocol: ReassignPartitionsRequest and ReassignPartitionsResponse

A `ReassignPartitionsRequest` initiates the movement of replicas between brokers, and is the basis of the `AdminClient.reassignPartitions()` method

Notes:

- The request must be sent to the controller.
- The request requires the `Alter` operation on the `CLUSTER` resource, since it can require significant inter-broker communication.
- The request will be subject to a policy, as described in KIP-201.

```
ReassignPartitionsRequest => [topic_reassignments] timeout validate_only throttle remove_throttle
  topic_reassignments => topic [partition_reassignments]
    topic => STRING
    partition_reassignments => partition_id [broker]
      partition_id => INT32
      broker => INT32
  timeout => INT32
  validate_only => BOOLEAN
  throttle => INT64
  remove_throttle => BOOLEAN
```

Where

Field	Description
`topic`	the name of a topic
`partition_id`	a partition of that topic
`broker`	a broker id
`timeout`	The maximum time to await a response in ms.
`validate_only`	when true: validate the request, but don't actually reassign the partitions

Algorithm:

1. The controller validates the request against configured authz, policy etc.
2. The controller computes set of topics in the request, and writes this as JSON to the new `/admin/throttled_replicas_removal` znode
3. The controller then updates the existing `leader.replication.throttled.replicas` and `follower.replication.throttled.replicas` properties of each topic config.
4. The controller computes the union of 1) the brokers currently hosting replicas of the topic partitions in the request 2) the brokers assigned to host topic partitions in the request, and write this as JSON to the new `/admin/throttled_rates_removal` znode.
5. The controller then updates the existing `leader.replication.throttled.rates` and `follower.replication.throttled.rates properties` of each broker config.
6. The controller writes reassignment JSON to the `/admin/reassign_partitions` znode

The intent behind this algorithm is that should the controller crash during the update, the reassignment won't have started and the throttles will be removed on controller failover.

The broker will use the same algorithm for determing the values of the topic and broker configs as is currently used in the `ReassignPartitionsCommand`

A `ReassignPartitionsResponse` describes which partitions in the request will be moved, and what was wrong with the request for those partitions which will not be moved.

```
ReassignPartitionsResponse => throttle_time_ms [reassign_partition_result]
  throttle_time_ms => INT32
  reassign_partition_result => topic [partition_error]
    topic => STRING
    partition_error => partition_id error_code error_message
      partition_id => INT32
      error_code => INT16
      error_message => NULLABLE_STRING
```

Where

Field	Description
`throttle_time_ms`	duration in milliseconds for which the request was throttled
`topic`	a topic name from the request
`partition_id`	a partition id for that topic, from the request
`error_code`	an error code for that topic partition
`error_message`	more detailed information about any error for that topic

Anticipated errors:

- `CLUSTER_AUTHORIZATION_FAILED` (31) Authorization failed
- POLICY_VIOLATION(44) The request violated the configured policy
- `INVALID_TOPIC_EXCEPTION` (17) If the topic doesn't exist
- `UNKNOWN_MEMBER_ID` (25) If any broker ids in the partition_reassignments included an unknown broker id
- `INVALID_REQUEST` (42) If duplicate topics appeared in the request
- `PARTITION_REASSIGNMENT_IN_PROGRESS` (new) If the reassignment cannot be started because a reassignment is currently running (i.e. the `/admin/reassign_partitions` znode exists)
- INVALID_THROTTLE (new) If the given throttle is <=0.
- `INVALID_REPLICA_ASSIGNMENT` (39) If a partition, replica or broker id in the partition_assignment doesn't exist or is incompatible with the requested num_partitions and /or replication_factor. The error_message would contain further information.
- `NONE` (0) reassignment has started

AdminClient: alterInterbrokerThrottledRates()

```
/**
 * Change the rate at which interbroker replication is throttled, replacing existing throttled rates.
 * For each broker in the given {@code rates}, the {@code leaderRate} of the corresponding
 * {@code ThrottledRate} is the throttled rate when the broker is acting as leader and
 * the {@code followerRate} is the throttled rate when the broker is acting as follower.
 * For the throttled rates to take effect, the given broker must also be present in the
 * list of throttled replicas, which can be set by {@link #alterInterbrokerThrottledReplicas()}.
 * The throttle will be automatically removed at the end of the current reassignment,
 * unless overridden in the given options.
 *
 * The current rates can be obtained from {@link #describeConfigs(Collection)}.
 *
 * @param rates Map from broker id to the throttled rates for that broker.
 * @param options The options.
 */
public abstract AlterInterbrokerThrottledRateResult alterInterbrokerThrottledRate(
        Map<Integer, ThrottledRate> rates,
        AlterInterbrokerThrottledRateOptions options);
```

Where:

```
/**
 * The throttled rate for interbroker replication on a particular broker.
 */
public class ThrottledRate {
    public ThrottledRate(long leaderRate, long followerRate) { ... }
    /**
     * The throttled rate when the broker is acting as leader.
     */
    long leaderRate() { ... }
    /**
     * The throttled rate when the broker is acting as follower.
     */
    long followerRate() { ... }
}

public class AlterInterbrokerThrottledRateOptions extends AbstractOptions<AlterInterbrokerThrottledRateOptions>
{

    public boolean autoRemoveThrottle() { ... }

    /**
     * True to automatically remove the throttle at the end of the current reassignment.
     */
    public AlterInterbrokerThrottledRateOptions autoRemoveThrottle(boolean autoRemoveThrottle) { ... }
}

public class AlterInterbrokerThrottledRateResult {
    // package-access ctor

    public Map<Integer, KafkaFuture<Void>> values() { ... }

    public KafkaFuture<Void> all() { ... }
}
```

Network API: AlterInterbrokerThrottledRatesRequest and AlterInterbrokerThrottledRatesResponse

```
AlterInterbrokerThrottledRatesRequest => [broker_throttles] remove_throttle timeout validate_only
  broker_throttles => broker_id leader_rate follower_rate
    broker_id => INT32
    leader_rate => INT64
    follower_rate => INT64
  timeout => INT32
  validate_only => BOOLEAN
  remove_throttle => BOOLEAN
```

Algorithm:

1. The controller validates the brokers and rates in the request and that the principal has `Alter` operation on the `CLUSTER` resource.
2. The controller gets the current value of the /admin/throttled_rates_removal znode, forms the union of those brokers with those in the request and updates the /admin/throttled_rates_removal znode with JSON representation of this union
3. The controller then subtracts the brokers in the request from the current brokers and removes the leader.replication.throttled.rates and follower.replication.throttled.rates properties from each broker config
4. The controller then, for each broker in the request, adds the leader.replication.throttled.rates and follower.replication.throttled.rates properties to each broker config.
5. The controller then updates /admin/throttled_rates_removal znode with JSON representation of brokers in the request.

The intent behind this algorithm is that should the controller crash during the update, throttles will still be removed on completion of reassignment.

```
AlterInterbrokerThrottledRatesResponse => [broker_error]
  broker_error => broker_id error_code error_messgae
      broker_id => INT32
      error_code => INT16
      error_message => NULLABLE_STRING
```

Anticipated Errors:

- NOT_CONTROLLER (41) if the request was sent to a broker that wasn't the controller.
- `CLUSTER_AUTHORIZATION_FAILED` (31) Authorization failed
- INVALID_THROTTLE (new) if the throttled rate is <= 0.
- `UNKNOWN_MEMBER_ID(25) if the broker id in the request is not a broker in the cluster`

AdminClient: alterInterbrokerThrottledReplicas()

```
/**
 * Set the partitions and brokers subject to the
 * {@linkplain #alterInterbrokerThrottledRate(Map)
 * interbroker throttled rate}.
 * The brokers specified as the {@link ThrottledReplicas#leaders()} corresponding to a
 * topic partition given in {@code replicas} will be subject to the leader throttled rate
 * when acting as the leader for that partition.
 * The brokers specified as the {@link ThrottledReplicas#followers()} corresponding to a
 * topic partition given in {@code repicas} will be subject to the follower throttled rate
 * when acting as the follower for that partition.
 *
 * The throttle will be automatically removed at the end of the current reassignment,
 * unless overridden in the given options.
 *
 * The current throttled replicas can be obtained via {@link #describeConfigs(Collection)} with a
 * ConfigResource with type {@link ConfigResource.Type#TOPIC TOPIC} and name "leader.replication.throttled.
replicas"
 * or "follower.replication.throttled.replicas".
 */
public abstract AlterInterbrokerThrottledReplicasResult alterInterbrokerThrottledReplicas(
        Map<TopicPartition, ThrottledReplicas> replicas,
        AlterInterbrokerThrottledReplicasOptions options);
```

Where:

```
public class ThrottledReplicas {

    public ThrottledReplicas(Collection<Integer> leaders, Collection<Integer> followers) { ... }

    /**
     * The brokers which should be throttled when acting as leader. A null value indicates all brokers in the
cluster.
     */
    public Collection<Integer> leaders() { .. }

    /**
     * The brokers which should be throttled when acting as follower. A null value indicates all brokers in the
cluster.
     */
    public Collection<Integer> followers() { ... }

}

public class AlterInterbrokerThrottledReplicasOptions extends
AbstractOptions<AlterInterbrokerThrottledReplicasOptions> {

    public boolean autoRemoveThrottle() { ... }

    /**
     * True to automatically remove the throttle at the end of the current reassignment.
     */
    public AlterInterbrokerThrottledReplicasOptions autoRemoveThrottle(boolean autoRemoveThrottle) { ... }
}

public class AlterInterbrokerThrottledReplicasResult {
    // package-access ctor
    public Map<TopicPartition, KafkaFuture<Void>> values() { ... }
    public KafkaFuture<Void> all() { ... }
}
```

Network API: AlterInterbrokerThrottledReplicasRequest and AlterInterbrokerThrottledReplicasResponse

```
AlterInterbrokerThrottledRatesRequest => [topic_throttles] remove_throttle timeout validate_only
  topic_throttles => topic [partition_throttles]
    topic => STRING
    partition_throttles => partition_id [broker_id]
      partition_id => INT32
      broker_id => INT32
  timeout => INT32
  validate_only => BOOLEAN
  remove_throttle => BOOLEAN
```

Algorithm:

1. The controller validates the partitions and brokers in the request and that the principal has `Alter` operation on the `CLUSTER` resource.
2. The controller gets the current value of the `/admin/throttled_replicas_removal` znode, forms the union of those topics with those in the request and updates the `/admin/throttled_rates_removal` znode with JSON representation of this union
3. The controller then subtracts the topics in the request from the current topics and removes the `leader.replication.throttled.replicas` and `follower.replication.throttled.replicas` properties from each topic config
4. The controller then, for each topic in the request, adds the `leader.replication.throttled.rates` and `follower.replication.throttled.rates` properties to each topic config.
5. The controller then updates `/admin/throttled_replicas_removal` znode with JSON representation of topics in the request.

The intent behind this algorithm is that should the controller crash during the update, throttles will still be removed on completion of reassignment.

```
AlterInterbrokerThrottledReplicasResponse => [topic_errors]
  topic_errors => topic [partition_errors]
    topic => STRING
    partition_errors => partition_id error_code error_messgae
      partition_id => INT32
      error_code => INT16
      error_message => NULLABLE_STRING
```

Anticipated errors:

- NOT_CONTROLLER (41) if the request was sent to a broker that wasn't the controller.
- `CLUSTER_AUTHORIZATION_FAILED` (31) Authorization failed
- UNKNOWN_TOPIC_OR_PARTITION (3) if a partition in the request is not known in the cluster.
- `UNKNOWN_MEMBER_ID(25) if the broker id in the request is not a broker in the cluster`

On Controller Failover

The algorithms presented above, using the new znodes, are constructed so that should the controller fail, on election of a new controller ZooKeeper is not left in an inconsistent state where throttles which should be removed automatically are not removed at the end of the reassignment. The recovery algorithm is as follows:

1. If the `/admin/reassign_partitions` znode exists we assume a reassignment is on-going and do nothing.
2. Otherwise, if the `/admin/reassign_partitions` znode does not exists, we proceed to remove the throttles, as detailed in "Throttle removal" section below.

On reassignment completion

When reassignment is complete:

1. The `/admin/reassign_partitions` znode gets removed.
2. We remove the throttles, as detailed in "Throttle removal" section below.

Throttle removal

The algorithm for removing the throttled replicas is:

1. If `/admin/remove_throttled_replicas` is set:
    a. For each of the topics listed in that znode:
        i. Remove the `(leader|follower).replication.throttled.replicas` properties for that topic config.
    b. Remove the `/admin/remove_throttled_replicas` znode

The symmetric algorithm is used for `/admin/remove_throttled_rates`, only with broker configs.

Compatibility, Deprecation, and Migration Plan

Existing users of the `kafka-reassign-partitions.sh` will receive a deprecation warning when they use the `--zookeeper` option. The option will be removed in a future version of Kafka. If this KIP is introduced in version 1.0.0 the removal could happen in 2.0.0.

Implementing `AdminClient.alterConfigs()` for (dynamic) broker configs was considered as a way of implementing throttle management but this would not support the auto removal feature.

Not supporting passing a throttle in the AdminClient.reassignPartitions() (and just using the APIs for altering throttles) was considered, but:

- Being able to specify a throttle at the same time at starting the reassignment is very convenient.
- Race conditions are possible if the APIs requires throttles set up before reassignment starts. What if reassignPartitions() doesn't get called, or none of the partitions in the call can be be reassigned?

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.

One alternative is to do nothing: Let the ReassignPartitionsCommand continue to communicate with ZooKeeper directly.

Another alternative is to do exactly this KIP, but without the deprecation of `--zookeeper`. That would have a higher long term maintenance burden, and would prevent any future plans to, for example, provide alternative cluster technologies than ZooKeeper.

An `alterTopics()` AdminClient API, mirroring the existing `createTopics()` API, was considered, but:

- Some calls to `alterTopics()` (such as increasing the partition count) would have been synchronous, while others (such as moving replicas between brokers) would have been long running and thus asynchronous. This made for an API which synchronousness depended on the arguments.
- `createTopics()` allows to specify topic configs, whereas `alterConfigs()` is already provided to change topic configs, so it wasn't an exact mirror

Just providing `reassignPartitions()` was considered, with changes to partition count inferred from partitions present in the `assignments` argument. This would require the caller to provide an assignment of partitions to brokers, whereas currently it's possible to increase the partition count without specifying an assignment. It also suffered from the synchronous/asynchronous API problem.

Similarly a `alterReplicationFactors()` method, separate from `reassignPartitions()` was considered, but both require a partition to brokers assignment, and both are implemented in the same way (by writing to the `/admin/reassign_partitions` znode), so there didn't seem much point in making an API which distinguished them.

Algorithms making use of the ZooKeeper "multi" feature for atomic update of multiple znodes were considered. It wasn't clear that these would be better than the algorithms presented above.