KIP-113: Support replicas movement between log directories

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: *Completed*

Discussion thread: *here*

JIRA: *KAFKA-5163 and KAFKA-5694*

Released: 1.1.0

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

Motivation

The most expensive part of a Kafka cluster is probably its storage system. At LinkedIn we use RAID-10 for storage and set Kafka's replication factor = 2. This setup requires 4X space to store data and tolerates up to 1 broker failure. We are at risk of data loss with just 1 broker failure, which is not acceptable for e.g. financial data. On the other hand, it is prohibitively expensive to set replication factor = 3 with RAID-10 because it will increase our existing hardware cost and operational cost by 50%.

The solution is to use JBOD and set replication factor = 3 or higher. It is based on the idea that Kafka already has replication across brokers and it is unnecessary to use RAID-10 for replication. Let's say we set the replication factor = 4 with JBOD. This setup requires 4X space to store data and tolerate up to 3 broker failures in order not to lose any data. In comparison to our existing setup, this allows us to obtain 3X broker failure tolerance without increasing storage hardware cost.

We have evaluated the possibility of using other RAID setup in LinkedIn. But none of them addresses our problem as JBOD does. RAID-0 stops working entirely with just one disk failure. RAID-5 or RAID-6 has sizable performance loss as compared to RAID-0 and probably JBOD as well, due to their use of block-level striping with distributed parity.

Unfortunately, JBOD is not recommended for Kafka because some important features are missing. For example, Kafka lacks good support for tools as well as load balancing across disks when multiple disks are used. Here is a list of problems that need to be addressed for JBOD to be useful:

1) Broker will shutdown if any disk fails. This means a single disk failure can bring down the entire broker. Instead, broker should still serve those replicas on the good disks as long as there is good disk available.

2) Kafka doesn't provide the necessary tools for users to manage JBOD. For example, Kafka doesn't provide script to re-assign replicas between disks of the same broker. These tools are needed before we can use JBOD with Kafka.

3) JBOD doesn't by itself balance load across disks as RAID-10 does. This will be a new problem for us to solve in order for JBOD setup to work well. We should have a better solution than round-robin which we are using to select disk to place a new replica. And we should probably figure out how to re-assign replicas across disks of the same broker if we notice load imbalance across disks of a broker.

For ease of discussion, we have separated the design of JBOD support into two different KIPs. This KIP address the second problem. See KIP - Handle disk failure for JBOD to read our proposal of how to address the first problem.

Since Kafka configuration and implementation does not expose "disk", we will use log directory and disk interchangably in the rest of the KIP.

Goals

The goal of this KIP is to allow administrator to re-assign replicas to the specific log directories of brokers so that they can balance load across disks. This addresses the second problem raised in the motivation section. See KIP - Handle disk failure for JBOD to read our proposal of how to address the first problem.

Proposed change

1) How to move replica between log directories on the same broker

Problem statement:

Kafka doesn't not allow user to move replica to another log directory on the same broker in runtime. This is not needed previously because we use RAID-10 and the load is already balanced across disks. But it will be needed to use JBOD with Kafka.

Currently a replica only has two possible states, follower or leader. And it is identified by the 3-tuple (topic, partition, broker). This works for the current implementation because there can be at most one such replica on a broker. However, now we can have two such replicas on a broker when we move replica from one log directory to another log directory on the same broker. Either a replica should be identified by its log directory as well, or the broker needs to persist information under the log directory to tell the destination replica from source replica that is being moved.

In addition, user needs to be able to query list of partitions and their size per log directory on any machine so that they can determine how to move replicas to balance the load. While these information may be retrieved via external tools if user can ssh to the machine that is running the Kafka broker, development of such tools may not be easy on all the operating systems that Kafka may run on. Further, a user may not even have the authorization to access the machine. Therefore Kafka needs to provide new request/response to provide this information to users.

Solution:

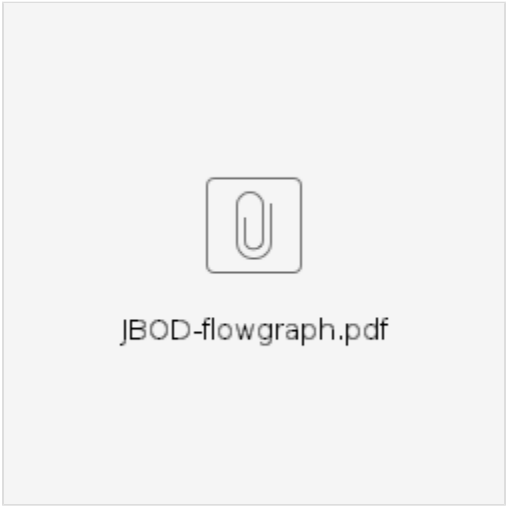
The idea is that user can send a `AlterReplicaDirRequest` which tells broker to move `topicPartition` directory (which contains all log segments of the `topicPartition` replica) from the source log directory to a destination log directory. Broker can create a new directory with .move postfix on the desti nation log directory to hold all log segments of the replica. This allows broker to tell log segments of the replica on the destination log directory from log segments of the replica on the source log directory during broker startup. The broker can create new log segments for the replica on the destination log directory, push data from source log to the destination log, and replace source log with the destination log for this replica once the new log has caught up.

In the following we describe each step of the replica movement.

1. Initiate replica movement using AlterReplicaDirRequest

User uses *kafka-reassignment-partitions.sh* to send AlterReplicaDirRequest to broker to initiate replica movement between its log directories. The flow graph below illustrates how broker handles `AlterReplicaDirRequest`.

Notes:
- Broker will cancel existing movement of the replica if "any" is specified as destination log directory.
- If broker doesn't not have already replica created for the specified topicParition when it receives AlterReplicaDirRequest, it will reply ReplicaNotAvailableException AND remember (replica, destination log directory) pair in memory to create the replica in the specified log directory when it receives LeaderAndIsrRequest later.



2. Complete replica data movement

Here we describe how a broker moves a `Log` from source to destination log directory and swaps the `Log`. This corresponds to the "Initiate replica data movement" box in the flow graph above. Note that the broker responds to `AlterReplicaDirRequest` with `MoveInProgress` after step 1) described below.

1) The `Replica` instance is updated to track two instances of `Log`, one referencing the directory topicPartition on the source log directory and the other ref erencing the directory topicPartition.move on the destination log directory.
2) If there is thread available in `ReplicaMoveThreadPool`, one thread gets allocated to move this replica. Otherwise, the movement of this replica gets delayed until there is available thread.
3) The `ReplicaMoveThread` keeps reading data from the `Log` in the source log directory to the `Log` (i.e. toipcPartition.move) in the destination log directory using zero-copy. `ReplicaMoveThread` may need to sleep to ensure that total throughput in byte-rate used by all `ReplicaMoveThread` instances does not exceed the configured value of `intra.broker.throttled.rate`.
4) If the `Log` in the destination log directory has caught up with the `Log` in the source log directory, the `ReplicaMoveThread` grabs lock on the Replica instance.
5) The `ReplicaMoveThread` continues to move data as specified in step 3) until the `Log` in the destination log directory has caught up with the `Log` in the source log directory. We need to do this again because RequestHandlerThread or ReplicaFetcherThread may be appending data to the log in the source log directory concurrently after the check in step 4).
6) The `ReplicaMoveThread` renames directory topicPartition to topicPartition.delete on the source log directory. topicParition.delete will be subject to asynchronous delete.
7) The `ReplicaMoveThread` renames directory topicParition.move to topicParition on the destination log directory
8) The `ReplicaMoveThread` updates the corresponding `Replica` instance to track only the `Log` in the destination log directory.
9) The `ReplicaMoveThread` releases lock on the Replica instance.

Notes:
- The replica movement will stop if either source or destination replica becomes offline due to log directory failure.
- The `RequestHandlerThread` or `ReplicaFetcherThread` needs to grab lock of the Replica instance in order to append data to the Replica. This prevents race condition while `ReplicaMoveThread` is swapping the Log in the source log directory with the log in the destination log directory.
- The log segments in topicPartition.move directory will be subject to log truncation, log retention in the same way as the log segments in the source log directory. But we may not do log cleaning on the topicPartition.move to simplify the implementation.

3. Handle failure that happens when broker is moving data or swapping replica

Broker does the following to recover from failure when it starts up.

- If both the directory topicPartition and the directory topicPartition.move exist on good log directories, broker will start `ReplicaMoveThread` to copy data from `topicPartition` to `topicPartition.move`. The effect is the same as if broker has received `AlterReplicaDirRequest` to move replica from `t opicPartition` to `topicPartition.move`.
- If topicPartition.move exists but topicPartition doesn't exist on any good log directory, and if there is no bad log directory, then broker renames topicPartition.move to topicPartition.
- If topicPartition.move exists but topicPartition doesn't exist on any good log directory, and if there is bad log directory, then broker considers topicPartition as offline and would not touch topicPartition.move.
- If topicPartition.delete exists, the broker schedules topicParition.delete for asynchronous delete.

2) How to reassign replica between log directories across brokers

Problem statement:

kafka-reassign-partitions.sh should provide an option for user to specify destination log directory of the replica in the reassignment json file. And user should be able to verify that the replica has been moved to the specified log directory after the reassignment is completed. This is needed by user to balance load across log directories of brokers in the cluster.

Solution:

The idea is that user should be able to specify log directory when using *kafka-reassign-partitions.sh* to reassign partition. If user has specified log directory on the destination broker, the script should send `AlterReplicaDirRequest` directly to the broker so that broker can start `ReplicaMoveThread` to move the replica. Finally, the script should send `DescribeLogDirsRequest` to broker to verify that the replica has been created/moved in the specified log directory when user requests to verify the assignment.

Here are the steps to execute partition reassignment:

- User specifies a list of log directories, one log directory per replica, for each topic partition in the reassignment json file that is provided to *kafka-reassignment-partitions.sh*. The log directory specified by user must be either "any", or absolute path which begins with '/'. If "any" is specified as the log directory, the broker is free to choose any log directory to place the replica. Current broker implementation will select log directory using round-robin algorithm by default. See `Scripts` section for the format of this json file.
- The script sends AlterReplicaDirRequest to those brokers which need to move replicas to user-specified log directory. This step can be skipped if user has specified "any" as log directory for all replicas. The script exits with error if the broker to receive AlterReplicaDirRequest is offline or if the AlterReplicaDirResponse contains any error that is not ReplicaNotAvailableException.
- The script creates reassignment znode in zookeeper.
- The script retries AlterReplicaDirRequest to those brokers which have responded with ReplicaNotAvailableException in the AlterReplicaDirResponse previously. The script keeps retrying up to user-specified timeout. The timeout is 10 seconds by default. The script exits with error if the broker to receive AlterReplicaDirRequest is offline or if the AlterReplicaDirResponse contains any error that is not ReplicaNotAvailableException.
- The script returns result to user.

Here are the steps to verify partition assignment:

- *kafka-reassignment-partitions.sh* will verify partition assignment across brokers as it does now.
- For those replicas with destination log directory != "any", *kafka-reassignment-partitions.sh* groups those replicas according to their brokers and and sends `DescribeLogDirsRequest` to those brokers. The `DescribeLogDirsRequest` should provide the log directories and partitions specified in the expected assignment.
- Broker replies with `DescribeLogDirsResponse` which shows the current log directory for each partition specified in the `DescribeLogDirsRequest`.
- *kafka-reassignment-partitions.sh* determines whether the replica has been moved to the specified log directory based on the `DescribeLogDirsResponse`.

3) How to retrieve information to determine the new replica assignment across log directories

Problem statement:

In order to optimize replica assignment across log directories, user would need to figure out the list partitions per log directory, the size of each partition. As of now Kafka doesn't expose this information via any RPC and user would need to either query the JMX metrics of the broker, or use external tools to log onto each machine to get this information. While it is possible to retrieve these information via JMX, users would have to manage JMX port and related credentials. It is better if Kafka can expose this information via RPC.

Solution:

We introduce `DescribeLogDirsRequest` and `DescribeLogDirsResponse`. When a broker receives `DescribeLogDirsRequest` with empty list of log directories, it will respond with a `DescribeLogDirsResponse` which shows the size of each partition and lists of partitions per log directory for all log directories. If user has specified a list of log directories in the `DescribeLogDirsRequest`, the broker will provide the above information for only log directories specified by the user. If user has specified an empty list of topics in the `DescribeLogDirsRequest`, all topics will be queried and included in the response. Otherwise, only those topics specified in the `DescribeLogDirsRequest` will be queried. Non-zero error code will be specified in the `DescribeLogDirsResponse` for each log directory that is either offline or not found by the broker.

User can use command such as *./bin/kafka-log-dirs.sh --describe --zookeeper localhost:2181 --broker 1* to get the above information per log directory.

Public interface

Protocol

Create AlterReplicaDirRequest

```
AlterReplicaDirRequest => topics
  topics => [AlterReplicaDirRequestTopic]

AlterReplicaDirRequestTopic => topic partitions
  topic => str
  partitions => [AlterReplicaDirRequestPartition]

AlterReplicaDirRequestPartition => partition log_dir
  partition => int32
  log_dir => str
```

Create AlterReplicaDirResponse

```
AlterReplicaDirResponse => topics
  topics => [AlterReplicaDirResponseTopic]

AlterReplicaDirResponseTopic => topic partitions
  topic => str
  partitions => [AlterReplicaDirResponsePartition]

AlterReplicaDirResponsePartition => partition error_code
  partition => int32
  error_code => int16
```

Create DescribeLogDirsRequest

```
DescribeLogDirsRequest => topics
  topics => DescribeLogDirsRequestTopic // If this is empty, all topics will be queried

DescribeLogDirsRequestTopic => topic partitions
  topic => str
  partitions => [int32]
```

Create DescribeLogDirsResponse

```
DescribeLogDirsResponse => log_dirs
  log_dirs => [DescribeLogDirsResponseDirMetadata]

DescribeLogDirsResponseDirMetadata => error_code path topics
  error_code => int16
  path => str
  topics => [DescribeLogDirsResponseTopic]

DescribeLogDirsResponseTopic => topic partitions
  topic => str
  partitions => [DescribeLogDirsResponsePartition]

DescribeLogDirsResponsePartition => partition size offset_lag is_temporary
  partition => int32
  size => int64
  offset_lag => int64  // If this is not a temporary replica, then offset_lag = max(0, HW - LEO). Otherwise,
offset_lag = primary Replica's LEO - temporary Replica's LEO
  is_temporary => boolean  // True if replica is *.move
```

Broker Config

1) Add config `intra.broker.throttled.rate`. This config specified the maximum rate in bytes-per-second that can be used to move replica between log directories. This config defaults to MAX_LONG. The intra.broker.throttled.rate is per-broker and the specified capacity is shared by all replica-movement-threads.

2) Add config `num.replica.alter.log.dirs.threads`. This config specified the number of threads in `ReplicaMoveThreadPool`. The thread in this thread pool is responsible to moving replica between log directories. This config defaults to the number of log directories. Note that we typically expect 1-1 mapping between log directories and disks. Thus setting the config to number of log directories by default provides a reasonable way to keep the movement capacity in proportion with the number of disks.

Scripts

1) Add *kafka-log-dirs.sh* which allows user to get list of replicas per log directory on a broker.

./bin/kafka-log-dirs.sh --describe --zookeeper localhost:2181 --broker 1 --log-dirs dir1,dir2,dir3 --topics topic1,topic2 will show list of partitions and their size per log directory for the specified topics and the specified log directories on the broker. If no log directory is specified by the user, then all log directories will be queried. If no topic is specified, then all topics will be queried. If a log directory is offline, then its error code in the DescribeLogDirsResponse will indicate the error and the log directory will be marked offline in the script output.

Note that the script can be used to check whether there is ongoing replica movement between log directories by looking for partition with is_temporary = True. User can also track movement progress by comparing LEO of the *.log and *.move
The script output would have the following json format.

```
{
  "version" : 1,
  "log_dirs" : [
    {
      "is_live" : boolean,
      "path" : str,
      "partitions": [
        {
          "topic" : str,
          "partition" : int32,
          "size" : int64,
          "offset_lag" : in64,
          "is_temporary" : boolean
        },
        ...
      ]
    },

    ...
  ]
}
```

2) Change *kafka-reassignment-partitions.sh* to allow user to specify the log directory that the replica should be moved to. This is provided via the reassignment json file with the following new format:

```
{
  "version" : int,
  "partitions" : [
    {
      "topic" : str,
      "partition" : int,
      "replicas" : [int],
      "log_dirs" : [str]    <-- NEW. A log directory can be either "any", or a valid absolute path that begins
with '/'. This is an optional filed. It is treated as an array of "any" if this field is not explicitly
specified in the json file.
    },
    ...
  ]
}
```

Note: the quota specified by the argument `–throttle` will be applied to only inter-broker replica reassignment. It does not affect the quota for replica movement between log directories.

3) Add optional argument --timeout to *kafka-reassignment-partitions.sh*. This is because *kafka-reassignment-partitions.sh* may need to re-send `AlterReplicaDirRequest` to broker if replica hasn't already been created there. The timeout is set to 10 seconds by default.

AdminClient

The following methods and classes are added.

```
public interface AdminClient extends AutoCloseable {
    /**
     * Query the log directory information for the specified log directories on the given brokers.
     * All log directories on a broker are queried if an empty collection of log directories is specified for
this broker
     *
     * This operation is supported by brokers with version 0.11.1.0 or higher.
     *
     * @param logDirsByBroker     A list of log dirs per broker
     * @param options             The options to use when querying log dir info
```

```
     * @return                   The DescribeLogDirsResult
     */
    public abstract DescribeLogDirsResult describeLogDirs(Map<Integer, Collection<String>> logDirsByBroker,
DescribeLogDirsOptions options);

    /**
     * Query the replica directory information for the specified replicas.
     *
     * This operation is supported by brokers with version 0.11.1.0 or higher.
     *
     * @param replicas      The replicas to query
     * @param options       The options to use when querying replica dir info
     * @return              The DescribeReplicaLogDirsResult
     */
    public abstract DescribeReplicaLogDirsResult describeReplicaLogDirs(Collection<TopicPartitionReplica>
replicas, DescribeReplicaLogDirsOptions options);
}


public class KafkaAdminClient extends AdminClient {
    /**
     * Alter the log directory for the specified replicas.
     *
     * Updates are not transactional so they may succeed for some resources while fail for others. The log
directory for
     * a particular replica is updated atomically.
     *
     * This operation is supported by brokers with version 0.11.1.0 or higher.
     *
     * @param replicaAssignment  The replicas with their log directory absolute path
     * @param options            The options to use when changing replica dir
     * @return                   The AlterReplicaDirResult
     */
    public AlterReplicaDirResult alterReplicaDir(Map<TopicPartitionReplica, String> replicaAssignment,
AlterReplicaDirOptions options);

}


/**
  * Options for the alterReplicaDir call.
  */
class AlterReplicaDirOptions {
    private Integer timeoutMs = null;
    public AlterReplicaDirOptions timeoutMs(Integer timeoutMs);
    public Integer timeoutMs();
}

/**
  * The result of the alterReplicaDir call.
  */
class AlterReplicaDirResult {
    /**
     * Return a map from replica to futures, which can be used to check the status of individual replica
movement.
     */
    public Map<TopicPartitionReplica, KafkaFuture<Void>> values();

    /**
     * Return a future which succeeds if all the replica movement have succeeded
     */
    public KafkaFuture<Void> all();
}

/**
  * Options for the describeDirs call.
  */
class DescribeLogDirsOptions {
    private Integer timeoutMs = null;
    public DescribeLogDirsOptions timeoutMs(Integer timeoutMs);
    public Integer timeoutMs();
```

```
}

/**
  * The result of the describeDirs call.
  */
class DescribeLogDirsResult {
    /**
     * Return a map from brokerId to futures which can be used to check the information of partitions on each
individual broker
     */
    public Map<Integer, KafkaFuture<Map<String, LogDirInfo>>> values();

    /**
     * Return a future which succeeds only if all the brokers have responded without error
     */
    public KafkaFuture<Map<Integer, Map<String, LogDirInfo>>> all();
}

/**
  * Description of a log directory
  */
class LogDirInfo {
    public final Errors error;
    public final Map<TopicPartition, ReplicaInfo> replicaInfos;
}

/**
  * Description of a replica
  */
public class ReplicaInfo {
    public final long size;
    public final long logEndOffset;
    public final boolean isTemporary;
}

/**
  * Options for the describeReplicaDir call.
  */
class DescribeReplicaLogDirsOptions {
    private Integer timeoutMs = null;
    public DescribeReplicaLogDirsOptions timeoutMs(Integer timeoutMs);
    public Integer timeoutMs();
}

/**
  * The result of the describeReplicaDir call.
  */
class DescribeReplicaLogDirsResult {
    /**
     * Return a map from replica to futures which can be used to check the log directory information of
individual replicas
     */
    public Map<TopicPartitionReplica, KafkaFuture<ReplicaDirInfo>> values();

    /**
     * Return a future which succeeds if log directory information of all replicas are available
     */
    public KafkaFuture<Map<TopicPartitionReplica, ReplicaDirInfo>> all();
}

/**
  * Log directory information of a given replica and its intra-broker movement progress
  */
class ReplicaDirInfo {
    public String currentReplicaDir;
    public String temporaryReplicaDir;
    public long temporaryReplicaOffsetLag;
}
```

Changes in Operational Procedures

In this section we describe the expected changes in operational procedures in order to run Kafka with JBOD. Administrators of Kafka cluster need to be aware of these changes before switching from RAID-10 to JBOD.

- Need to load balance across log directories

When running Kafka with RAID-10, we only need to take care of load imbalance across brokers. Administrator can balance load across brokers using the script kafka-reassign-partitions.sh. After switching from RAID-10 to JBOD, we will start to see load imbalance across log directories. In order to address this problem, administrator needs to get the partition assignment and their size per log directory using *kafka-log-dirs.sh*, determine the reassignment of replicas per log directory per broker, and provide partition to (broker, log_directory) mapping as input to *kafka-reassign-partitions.sh* to execute the new assignment.

Administrator also needs to be prepared that the need to rebalance across log directories will probably be much more frequent than the need to rebalance across brokers since the capacity of individual disk is likely much smaller than the capacity of existing RAID-10 setup.

Compatibility, Deprecation, and Migration Plan

This KIP is a pure addition. So there is no backward compatibility concern.

The KIP changes the inter-broker protocol. Therefore the migration requires two rolling bounce. In the first rolling bounce we will deploy the new code but broker will still communicate using the existing protocol. In the second rolling bounce we will change the config so that broker will start to communicate with each other using the new protocol.

Test Plan

The new features will be tested through unit, integration, and system tests. In the following we explain the system tests only.

Note that we validate the following when we say "validate client/cluster state" in the system tests.
- Brokers are all running and show expected error message
- topic description shows expected results for all topics
- A pair of producer and consumer can successfully produce/consume from each monitored topic without message loss or duplication.

1) Move replica between disks on the same broker and across brokers

- Start 1 zookeeper and 3 brokers. Each broker has 2 log directories.
- Create a topic of 3 partition with 3 replication_factor=1
- Start a pair of producer and consumer to produce/consume from the topic
- Get current log directory of each replica
- Run *kafka-reassignment-partitions.sh* to move each replica to the other log directory on the same broker
- Run *kafka-reassignment-partitions.sh* to periodically verify and wait for reassignment to complete within reasonable amount of time
- validate client/cluster state
- Run *kafka-reassignment-partitions.sh* to move each replica to the currently-unused log directory on the "next" broker. I.e., replica1 is moved from broker1 -> broker2, replica2 is moved from broker2 -> broker3, and replica3 is moved from broker3 -> broker1.
- Run *kafka-reassignment-partitions.sh* to periodically verify and wait for reassignment to complete within reasonable amount of time
- validate client/cluster state

2) Verify that bad log directories discovered during runtime do not affect replicas on the good log directories.

- Start 1 zookeeper and 3 brokers. Each broker has 2 log directories.
- Create topic1 of 1 partition with 3 replicas. Run *kafka-reassignment-partitions.sh* to move replicas of topic1 to the first log directory of each broker.
- Create topic2 of 1 partition with 3 replicas. Run *kafka-reassignment-partitions.sh* to move replicas of topic2 to the second log directory of each broker.
- Start a pair of producer and consumer to produce/consume from topic1
- Start a pair of producer and consumer to produce/consume from topic2
- Change permission of the second log directory of a follower broker of topic1's partition to be 000.
- Validated client/cluster state for both topics.
- Change permission of the second log directory of the leader broker of topic1's partition to be 000.
- Validated client/cluster state for both topics.
- Change permission of the second log directory of all brokers to be 000.
- Validated client/cluster state for topic1.

Rejected Alternatives

1) Write the replica -> log directory mapping in the reassignment znode and have controller send AlterReplicaDirRequest to brokers.

This alternative solution has a few drawbacks:
- There can be use-cases where we only want to rebalance the load of log directories on a given broker. It seems unnecessary to go through controller in this case.
- If controller is responsible for sending AlterReplicaDirRequest, and if the user-specified log directory is either invalid or offline, then controller probably needs a way to tell user that the partition reassignment has failed. We currently don't have a way to do this since kafka-reassign-partition.sh simply creates the reassignment znode without waiting for response. I am not sure that is a good solution to this.
- If controller is responsible for sending AlterReplicaDirRequest, the controller logic would be more complicated because controller needs to first send AlterReplicaDirRequest so that the broker memorize the partition -> log directory mapping, send LeaderAndIsrRequest, and keep sending AlterReplicaDirRequest (just in case broker restarted) until replica is created. Note that the last step needs repeat and timeout as the proposed in the KIP-113.

Overall this alterantive adds quite a bit complexity to controller and we probably want to do this only if there is strong clear of doing so. Currently in KIP-113 the kafka-reassign-partitions.sh is responsible for sending AlterReplicaDirRequest with repeat and provides error to user if it either fails or timeout. It seems to be much simpler and user shouldn't care whether it is done through controller.