KIP-660: Pluggable ReplicaPlacer

Status

Current state: Discarded

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

JIRA: *here [Change the link from KAFKA-1 to your own ticket]*

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

Motivation

When creating topics or partitions, the Kafka controller has to pick brokers to host the new partitions. The current placement logic is based on a round robin algorithm and supports rack awareness. While this works relatively well in many scenarios, in a few cases the placements it generates are not optimal because it's not aware of the state of the clusters. Many cluster administrators then rely on tools like Cruise Control to move partitions to better brokers. This process is expensive as often data has to be copied between brokers.

It would be desirable to allow custom logic for the placer to enable administrators to build assignment rules for their clusters and minimize the number of partition reassignments necessary.

Some scenarios that could benefit greatly from this feature:

- When adding brokers to a cluster, Kafka currently does not necessarily place new partitions on new brokers
- When administrators want to remove brokers from a cluster, there is no way to prevent Kafka from placing partitions on them
- When some brokers are near their storage/throughput limit, Kafka could avoid putting new partitions on them

Public Interfaces

1) New public interface: ReplicaPlacer

ReplicaAssignor

```
package org.apache.kafka.server.placer;

/**
 * The interface which a Kafka replica placement policy must implement.
 */
@InterfaceStability.Evolving
public
interface ReplicaPlacer extends Configurable, Closeable {
    /**
     * Create a new replica placement.
     *
     * @param startPartition        The partition ID to start with.
     * @param numPartitions         The number of partitions to create placements for.
     * @param numReplicas           The number of replicas to create for each partition.
     * @param iterator              An iterator that yields all the usable brokers.
     *
     * @return                      A list of replica lists.
     *
     * @throws ReplicaPlacementException            If a new replica placement can't be created
     */
    List<List<Integer>> place(int startPartition,
                              int numPartitions,
                              short numReplicas,
                              Iterator<UsableBroker> iterator)
        throws ReplicaPlacementException;
}
```

2) New public class: UsableBroker

```
/**
 * A broker where a replica can be placed.
 */
public class UsableBroker {

    public UsableBroker(int id, Optional<String> rack, boolean fenced) {
        this.id = id;
        this.rack = rack;
        this.fenced = fenced;
    }

    public int id() {
        return id;
    }

    public Optional<String> rack() {
        return rack;
    }

    public boolean fenced() {
        return fenced;
    }
}
```

3) New broker configuration:

Name: replica.placer.class.name

Type: class

Doc: The fully qualified class name that implements ReplicaPlacer. This is used by the broker to determine replicas when topics or partitions are created. This defaults to StripedReplicaPlacer.

4) New exception and error code

A new exception, org.apache.kafka.common.errors.ReplicaPlacementException, will be defined. It will be non retriable.

When ReplicaPlacer implementations throw this exception, it will be mapped to a new error code:

> REPLICA_PLACEMENT_FAILED: The replica placer could not compute a placement for the topic or partition.

Proposed Changes

The proposal is to expose the ReplicaPlacer interface. It will move from the org.apache.kafka.controller package in the metadata project to the org.apache.kafka.server.placer package in the clients project. Similarly the existing UsableBroker class will move from org.apache.kafka.metadata package in the metadata project to the org.apache.kafka.server.placer package in the clients project. Sanity checks about the replication factor that are currently performed in StripedReplicaPlacer will be performed in ControlClusterManager as they are common to all ReplicaPlacer implementations.

To address the use cases identified in the motivation section, some knowledge about the current state of the cluster is necessary. Details whether a new broker has just been added or is being decommissioned are not part of the cluster metadata. Therefore such knowledge has to be provided via an external mean to the ReplicaPlacer, for example via the configuration. Apart from configure(), this KIP does not provide a mechanism for defining the behavior of ReplicaPlacer implementation and cluster operators wanting to use this feature will have to build such a mechanism for their specific environments.

The logic assigning replicas to partition differs so much between ZooKeeper and KRaft that I propose making this feature only available in KRaft mode.

Compatibility, Deprecation, and Migration Plan

Older clients that don't have the new error code will interpret it as UNKNOWN_SERVER_ERROR but they will receive the generated error message indicating the reason for the failure.

Rejected Alternatives

- Computing replica placement for the whole create topics/partitions request: Instead of computing assignment for each topic in the CreateTopics /CreatePartitions request one at a time, I looked at computing assignment for all of them in a single call. We rejected this approach for the following reasons:
  - All current logic works on a single topic at a time. Grouping the replica assignment computation created very complicated logic
  - It's not clear if having all topics at once would significantly improve computed assignments. This is especially true for the 4 scenarios listed in the Motivation section
- Providing more details about the cluster to the placer: Instead of only passing usable brokers, I considered passing a data structure with more details about the cluster, such as Cluster. While this could allow some additional advanced use cases, this would potentially not scale well if we expect Kafka to be able to support very large number of topics with KRaft.