KIP-881: Rack-aware Partition Assignment for Kafka Consumers

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: 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).

Motivation

Kafka clusters are often distributed across multiple availability zones (AZ), especially in Cloud deployments. Rack-aware replica placement support in Kafka introduced in KIP-36 can be used to configure availability zones as racks for brokers. Replicas of each partition are distributed across different availability zones to ensure that a zonal outage doesn't affect availability of a partition. KIP-392 added support for consumers to fetch from their closest replica. This feature enables consumers to fetch data from leaders or followers within the same availability zone, when possible, to benefit from locality.

Rack-aware Kafka deployments configure `broker.rack` for every broker based on their rack, which may be the AZ identifier. Brokers may also configure `replica.selector.class` that determines the preferred replica used to read data from. The default replica selector uses the leader as the preferred replica for reads, but the built-in `RackAwareReplicaSelector` can be configured to match the rack of the replica with the rack of the consumer. Consumers using this feature benefit from locality and avoid expensive cross-AZ traffic by configuring `client.rack`. Rack ids are propagated to brokers in fetch requests and this enables the rack-aware replica selector to choose a replica on the same rack as the consumer and provide this information to the consumer to use in subsequent fetch requests . This feature works well in scenarios where every rack contains replicas of all partitions. For example, for a Kafka deployment with three AZs and replication factor 3, rack-aware replica placement places the three replicas on the 3 AZs. So every AZ has a replica for every partition. This enables consumers in any of the 3 AZs to consume from their local replica. If the number of replicas is lower than the number of AZs or racks, perhaps because number of AZs is higher than replication factor, some partitions may not have replicas on some AZs. Consumers running on those AZs will have to fetch data from the leader in another AZ. Rack-aware partition assignment for consumers will improve locality in this case.

KIP-848 describes the next generation consumer group protocol that fixes several issues with the existing protocol. This proposal already includes rack information in the protocol, making it easy to introduce rack-aware partition assignment in both the server-side partition assignors and client-side partition assignors proposed by that KIP. Since KIP-848 is a major rewrite of the consumer implementation, the existing consumer implementation is likely to be in widespread use for quite some time. Since the protocol change to support rack-aware partition assignment is a minor change, it will be good to support this feature in the existing consumer implementation as well so that it can be adopted sooner.

Public Interfaces

This KIP proposes to include rack information in `ConsumerProtocolSubscription` message. Rack will be populated by each consumer from the existing `client.rack` configuration. The new protocol including the new `RackId` field will be:

ConsumerProtocolSubscription schema

```
{
  "type": "data",
  "name": "ConsumerProtocolSubscription",
  // Subscription part of the Consumer Protocol.
  //
  // The current implementation assumes that future versions will not break compatibility. When
  // it encounters a newer version, it parses it using the current format. This basically means
  // that new versions cannot remove or reorder any of the existing fields.
  //
  // Version 1 adds owned partitions.
  // Version 2 adds generationId (KIP-792).
  // Version 3 adds rack id to enable rack-aware assignment. <== NEW
  "validVersions": "0-3",
  "flexibleVersions": "none",
  "fields": [
    { "name": "Topics", "type": "[]string", "versions": "0+" },
    { "name": "UserData", "type": "bytes", "versions": "0+", "nullableVersions": "0+",
      "default": "null", "zeroCopy": true },
    { "name": "OwnedPartitions", "type": "[]TopicPartition", "versions": "1+", "ignorable": true,
      "fields": [
        { "name": "Topic", "type": "string", "mapKey": true, "versions": "1+", "entityType": "topicName" },
        { "name": "Partitions", "type": "[]int32", "versions": "1+"}
      ]
    },
    { "name": "GenerationId", "type": "int32", "versions": "2+", "default": "-1"},
    { "name": "RackId", "type": "string", "versions": "2+", "nullableVersions": "2+", "default": "null",
"ignorable": true } <== NEW
  ]
}
```

Rack id will be included in `ConsumerPartitionAssignor.Subscription` for each member's subscription metadata in `ConsumerPartitionAssignor.GroupSubscription` so that partition assignors can match the rack id of members with rack id of partition replicas.

Changes to ConsumerPartitionAssignor.Subscription

```
final class Subscription {
    ....
    private final Optional<String> rackId;

    public Subscription(List<String> topics, ByteBuffer userData, List<TopicPartition> ownedPartitions, int
generationId, Optional<String> rackId) {
        this.topics = topics;
        this.userData = userData;
        this.ownedPartitions = ownedPartitions;
        this.generationId = generationId;
        this.rackId = rackId;
        this.groupInstanceId = Optional.empty();
    }

    public Subscription(List<String> topics, ByteBuffer userData, List<TopicPartition> ownedPartitions) {
        this(topics, userData, ownedPartitions, Optional.empty());
    }

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

    ....
}
```

We also propose to update range assignor and cooperative sticky assignor to use rack-aware algorithm if `client.rack` is configured. Rack-aware assignors will match the racks of consumers and replicas on a best-effort basis and attempt to improve locality for consumer partition assignment.

Proposed Changes

For consumers which specify `client.rack` , the rack id be will added to `ConsumerProtocolSubscription` message using the protocol described above. This propagates the rack id of all members to the consumer that performs partition assignment.

Partition assignors already have access to cluster metadata when performing the assignment:

Existing partition assignor interface
`GroupAssignment assign(Cluster clusterMetadata, GroupSubscription groupSubscription);`

The `clusterMetadata` instance used by partition assignors contains replica information for every partition, where each replica's rack is included in their `Node` if the broker was configured with `broker.rack`. This KIP also adds rack id for each member's `Subscription` instance in `GroupSubscription`. So a rack-aware partition assignor can match the rack id of the members with the rack id of the replicas to ensure that consumers are assigned partitions in the same rack if possible. In some cases, this may not be possible, for example, if there is a single consumer and one partition which doesn't have a replica in the same rack. In this case the partition is assigned with mismatched racks and will result in cross-rack traffic. The built-in assignors will prioritize balancing partitions over improving locality, so in some cases, partitions may be allocated to a consumer in a different rack if there aren't sufficient partitions in the same rack as the consumer. The goal will be to improve locality for cases where load is uniformly distributed across a large number of partitions.

Rebalance to Improve Locality After Reassignments

Rack-aware partition assignment will use racks of all partition replicas including those marked offline or not in the ISR to ensure that transient states don't result in sub-optimal assignments. But replica racks may change due to reassignments when replicas are added or removed. In this case, the existing assignment may no longer be optimal and the next rebalance may not happen for a long time. To improve locality in this case, leader will trigger rebalance whenever it detects that the set of racks of partition replicas have changed in the metadata. This rebalance will be triggered only if the leader has `client.rack` configured. Since reassignments that change the set of replica racks of a partition are rare typically, this shouldn't result in frequent rebalances.

Compatibility, Deprecation, and Migration Plan

The new optional rack field for `ConsumerProtocolSubscription` will be added at the end of the structure, so it will not affect consumers with older versions.

Consumers without rack ids and/or partitions with replicas without rack ids are assigned partitions using the non-rack-aware algorithm by all assignors, ensuring that consumers with different versions are supported. Range assignor and cooperative sticky assignor will use rack-aware algorithm if `client.rack` is provided and both the client and brokers use version 3.4 and above. If either is at a lower version, existing non-rack-aware algorithm will be used. In the case where replication factor is greater than or equal to the number of racks, all racks will have replicas of all partitions and hence the assignors will retain very similar assignment as before. If replication factor is lower and `client.rack` is provided, the updated assignors will use the new rack-aware logic, but there is no other compatibility impact.

Test Plan

- Unit tests will be added for the protocol change and the new assignors.
- Existing integration tests will be used to ensure that clients using existing assignors work with and without rack ids.
- New integration tests will be added to verify rack-aware partition assignment.
- Existing system tests will be used to verify compatibility with older versions.

Rejected Alternatives

Use client tags similar to Kafka Streams instead of using rack configuration

Kafka Streams introduced rack-aware rack assignment in KIP-708. Flexible client tags were introduced to implement rack-awareness along with a rack-aware assignor for standby tasks. Tags are more flexible, but since we want to match existing rack configuration in the broker and consumers, it seems better to use rack id directly instead of adding prefixed tags. The next generation consumer group protocol proposed in KIP-848 also uses racks in the protocol.

Propagate rack in userData field

Kafka Streams currently propagates tags in the `userData` bytes field that is populated by its task assignor. We could do the same and populate rack only in a rack-aware partition assignor in the `userData` field that is managed by assignors. Since `client.rack` is a standard consumer configuration option and is used for follower fetching, it seems better to include this at the top level rather than in assignor-specific `userData` structure. This will allow any of the consumer partition assignors to take advantage of rack-based locality in future.

Implement new assignors for rack-aware assignment instead of updating existing assignors

Rack-awareness is not enabled by default in Kafka clients and brokers. For example, brokers use rack-aware replica selector for follower fetching, only if brokers are explicitly configured with `replica.selector.class`. We could retain existing assignors in consumers and implement new assignor classes for rack-aware assignment. But that requires consumers to be explicitly configured with new assignors to benefit from this KIP. Since consumers are configured with `client.rack` only to benefit from locality with follower fetching, it seems reasonable to update existing assignors rather than require a configuration change. In scenarios where all racks have replicas of all partitions, we can retain the existing logic, so there will be no impact of this change in this case. In scenarios where clients have configured `client.rack` to benefit from locality, but racks have a subset of replicas, it seems reasonable to make existing assignors rack-aware to benefit from improved locality without additional configuration changes.