KIP-615: add ConstrainedCooperativeStickyAssignor

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: *Rejected (modified existing assignor algorithm instead of adding new one, so no public APIs were changed)*

Discussion thread: *here*

JIRA: *KAFKA-9987*

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

Motivation

In KIP-429 we added the new CooperativeStickyAssignor which leverages on the underlying sticky assignment algorithm of the existing StickyAssignor (moved to AbstractStickyAssignor). The algorithm is fairly complex as it tries to optimize stickiness while satisfying perfect balance *in the case individual consumers may be subscribed to different subsets of the topics.* While it does a pretty good job at what it promises to do, it doesn't scale well with large numbers of consumers and partitions. To give a concrete example, users have reported that it takes 2.5 minutes for the assignment to complete with just 2100 consumers reading from 2100 partitions.

Since partitions revoked during the first of two cooperative rebalances will remain unassigned until the end of the second rebalance, it's important for the rebalance to be as fast as possible.

If we can constrain the problem a bit, we can simplify the algorithm greatly. In many cases the individual consumers won't be subscribed to some random subset of the total subscription, they will all be subscribed to the same set of topics and rely on the assignor to balance the partition workload. It would be nice to provide an additional cooperative assignor OOTB that performs efficiently for the many use cases that satisfy this constraint.

Public Interfaces

We will add a new assignor implementing the `ConsumerPartitionAssignor` interface, which can then be plugged in to get efficient cooperative rebalancing OOTB.

ConstrainedCooperativeAssignor

```
/**
 * A cooperative assignor that is optimized for the case with all consumers subscribed to the same set of
topics.
 * If your group does not satisfy this constraint, you should use the {@link CooperativeStickyAssignor} instead.
 * <p>
 * This assignor guarantees a balanced partition assignment such that the number of assigned partitions will
 * differ by at most one across all members of the group. It aims to be as "sticky" as possible without
violating
 * balance such that the same assignment will always be generated when there are no membership or metadata
changes.
 * Note that as a cooperative assignor, any partitions that are transferring ownership will be removed from the
 * assignment until they have been safely revoked. A followup rebalance will automatically be triggered to
assign
 * such partitions to their new owner according to the cooperative rebalancing protocol.
 * <p>
 * IMPORTANT: if upgrading from 2.3 or earlier, you must follow a specific upgrade path in order to safely turn
on
 * cooperative rebalancing. See the <a href="https://kafka.apache.org/documentation/#upgrade_240_notable"
>upgrade guide</a>
 * for details.
 */
public class ConstrainedCooperativeStickyAssignor extends AbstractStickyAssignor {

    @Override
    public String name() {
        return "constrained-cooperative-sticky";
    }

    @Override
    public List<RebalanceProtocol> supportedProtocols() {
        return Arrays.asList(RebalanceProtocol.COOPERATIVE, RebalanceProtocol.EAGER);
    }
}
```

Proposed Changes

The constrained sticky algorithm is as follows.

Let N be the number of consumers participating in the rebalance, and P be the total number of partitions summed across all subscribed topics. Since we assume that all consumers are subscribed to all topics, we can assert that in the balanced assignment each consumer will have between $(P/N)_{floor}$ and $(P/N)_{ceil}$ partitions.

We can do the assignment in just a few linear passes. First define the following sets:

```
C_f := (P/N)_floor, the floor capacity
C_c := (P/N)_ceil, the ceiling capacity

members := the sorted set of all consumers
partitions := the set of all partitions
unassigned_partitions := the set of partitions not yet assigned, initialized to be all partitions
unfilled_members := the set of consumers not yet at capacity, initialized to empty
max_capacity_members := the set of members with exactly C_c partitions assigned, initialized to empty
member.owned_partitions := the set of previously owned partitions encoded in the Subscription

// Reassign as many previously owned partitions as possible
for member : members
                remove any partitions that are no longer in the subscription from its owned partitions
                remove all owned_partitions if the generation is old
                if member.owned_partitions.size < C_f
                        assign all owned partitions to member and remove from unassigned_partitions
                        add member to unfilled_members
                else if member.owned_partitions.size == C_f
                        assign first C_f owned_partitions to member and remove from unassigned_partitions
                else
                        assign first C_c owned_partitions to member and remove from unassigned_partitions
                        add member to max_capacity_members

sort unassigned_partitions in partition order, ie t0_p0, t1_p0, t2_p0, t0_p1, t1_p0 .... (for data parallelism)
sort unfilled_members by memberId (for determinism)

// Fill remaining members up to C_f
for member : unfilled_members
        compute the remaining capacity as C = C_f - num_assigned_partitions
        pop the first C partitions from unassigned_partitions and assign to member

// Steal partitions from members with max_capacity if necessary
if we run out of partitions before getting to the end of unfilled members:
        for member : unfilled_members
                poll for first member in max_capacity_members and remove one partition
                assign this partition to the unfilled member

// Distribute remaining partitions, one per consumer, to fill some up to C_c if necessary
if we run out of unfilled_members before assigning all partitions:
        for partition : unassigned_partitions
                assign to next member in members that is not in max_capacity_members (then add member to
max_capacity_members)
```

Compatibility, Deprecation, and Migration Plan

N/A

Rejected Alternatives

Improving the existing sticky algorithm

> As mentioned above, the existing algorithm is good at what it aims to do and is considerably more flexible than the proposed assignor here. It seems best to add a new assignor to meet the needs of some without preventing others from using the only current cooperative assignor that may meet their needs