# KIP-70: Revise Partition Assignment Semantics on New Consumer's Subscription Change



# Status

**Current state**: *Committed*

**Discussion thread**: *here*

**JIRA**: *KAFKA-4033*

**Released:** 0.10.1.0

This KIP was co-authored with Jason Gustafson and Ewen Cheslack-Postava.

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

# Motivation

The idea behind this KIP was initiated as a result of the discussion on the pull request for KAFKA-3664. The original issue reported in the JIRA was about offsets of partitions not being committed when a consumer unsubscribes from them. Specifically, when users are using group management, if they call `consumer.subscribe()` or `consumer.unsubscribe()` to change the subscription, the removed subscriptions will be immediately removed and their offset will not be committed. The fix provided in the corresponding pull request includes performing a `commitAsync()` in `subscribe()` and `unsubscribe()` methods to trigger an offset commit only when auto commit is enabled for the consumer. This solution maintains the current invariants as far as consistency between the assignment and offset commits, and it addresses the main problem from the JIRA, which is basically that users will see duplicates when they change subscriptions with auto commit enabled. For users who are using manual commit, they will have to call `commitSync()` prior to changing their subscription, but that seems reasonable.

## But if we consider the issue reported in the JIRA more carefully, we conclude that the root cause is Consumer `assignment` that is currently updated immediately upon subscription changes (e.g. through this call, then this, and finally this). This behavior is something that can be improved due to following reasons (reference):

- There is no known pattern for application development that would rely on the current behavior where the assignment is filtered immediately.
- The current behavior seems error prone with respect to resource cleanup since it is not unreasonable for a user to expect symmetric calls to `onPartitionsAssigned` and `onPartitionsRevoked`.
- The behavior today is inconsistent between subscriptions that are a list of topics and regex subscriptions: when you set a regex subscription, the assignment is *not* filtered immediately after the subscription change.
- Very few people probably rely on the intersection of functionality that would be affected: they would have to both pass in a `ConsumerRebalanceListener`, make subscription changes, and rely on the callback returning the filtered list.

Fixing this behavior would likely change the solution provided in the JIRA's pull request.

Examples below should further clarify the problem, and the suggested semantics change.

# Public Interfaces

There are no new public interfaces introduced by this KIP, but it does make a subtle change to the semantics of the consumer's `subscribe` API as discussed below.

# Proposed Changes

The new consumer's implementation of topics subscribe and unsubscribe interfaces are modified. In the case of topic subscribe, it does not cause an immediate assignment update (this is how the regex subscribe interface is implemented); instead, the assignment remains valid until it has been revoked in the next rebalance. In the case of unsubscribe, consumed offsets are committed immediately before clearing the assignment. This is mostly about fixing an inconsistent behavior. The examples below show this inconsistency and how this KIP proposes to resolve it.

## Examples

In the following examples assume the cluster contains only two topics `foo` and `bar`, each with a single partition.

### Topic Subscription

#### Existing Semantics

```
consumer.subscribe(Arrays.asList("foo", "bar"))
System.out.println(consumer.assignment()); // prints []

consumer.poll(0)
  --> onPartitionsRevoked([])
  --> onPartitionsAssigned([(foo, 0), (bar,0)])
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)]

consumer.subscribe(Arrays.asList("foo"))
System.out.println(consumer.assignment()); // prints [(foo, 0)]

consumer.poll(0)
  --> onPartitionsRevoked([(foo, 0)])
  --> onPartitionsAssigned([(foo, 0)])
System.out.println(consumer.assignment()); // prints [(foo, 0)]
```

#### Suggested Semantics

```
consumer.subscribe(Arrays.asList("foo", "bar"))
System.out.println(consumer.assignment()); // prints []

consumer.poll(0)
  --> onPartitionsRevoked([])
  --> onPartitionsAssigned([(foo, 0), (bar,0)])
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)]

consumer.subscribe(Arrays.asList("foo"))
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)] # notice the change

consumer.poll(0)
  --> onPartitionsRevoked([(foo, 0), (bar, 0)]) // # notice the change
  --> onPartitionsAssigned([(foo, 0)])
System.out.println(consumer.assignment()); // prints [(foo, 0)]
```

### Regex Subscription

#### Existing Semantics

```
consumer.subscribe(Pattern.compile("..."))
System.out.println(consumer.assignment()); // prints []

consumer.poll(0)
  --> onPartitionsRevoked([])
  --> onPartitionsAssigned([(foo, 0), (bar,0)])
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)]

consumer.subscribe(Pattern.compile("f.."))
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)]

consumer.poll(0)
  --> onPartitionsRevoked([(foo, 0)])
  --> onPartitionsAssigned([(foo, 0)])
System.out.println(consumer.assignment()); // prints [(foo, 0)]
```

**Suggested Semantics**

```
consumer.subscribe(Pattern.compile("..."))
System.out.println(consumer.assignment()); // prints []

consumer.poll(0)
  --> onPartitionsRevoked([])
  --> onPartitionsAssigned([(foo, 0), (bar,0)])
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)]

consumer.subscribe(Pattern.compile("f.."))
System.out.println(consumer.assignment()); // prints [(foo, 0), (bar, 0)]

consumer.poll(0)
  --> onPartitionsRevoked([(foo, 0), (bar, 0)]) // # notice the change
  --> onPartitionsAssigned([(foo, 0)])
System.out.println(consumer.assignment()); // prints [(foo, 0)]
```

# Compatibility, Deprecation, and Migration Plan

- Impacted users: Only those who currently rely on passing a `ConsumerRebalanceListener`, making a call to consumer's topics subscribe interface, and expecting an updated assignment in the callback are impacted. The number of users who rely on this specific use case is expected to be minimal to zero.

# Test Plan

*Describe in few sentences how the KIP will be tested. We are mostly interested in system tests (since unit-tests are specific to implementation details). How will we know that the implementation works as expected? How will we know nothing broke?*

# 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.*