Reverse Lecture 1 Report

Performance Tuning of Apache Kafka Rebalance

Mudambi Seshadri Srinivas
Vaishali Koul
Roberto Campbell

CMPE 275
San Jose State University

Table of contents

Abstract

Enterprise integration is one of the key building blocks of an enterprise system architecture, with many open-source offerings available. One of the most widely used messaging systems for integration available today is Apache Kafka. Kafka provides fault tolerance, high resiliency, and low latency for use in real-time applications. By taking advantage of Kafka's distributed architecture, we can integrate a variety of applications with little overhead and through the use of a concise and easy to understand API. However, in an environment with multiple consumers, Kafka has an added overhead when nodes are added or removed, such as in the case of a node failure. To determine the performance effect of rebalancing consumption by consumer groups on a topic, we compare the performance of the Kafka Improvement Plan (KIP) 54 feature StickyAssignor with the RoundRobin rebalancing strategy.

Background

Apache Kafka is a distributed streaming application for real-time data processing. Like earlier messaging queue systems such as RabbitMQ, Kafka provides a publish subscribe API which decouples the message consumption from production by publishing and consuming from topics. However, Kafka differs from messaging queues through its use of an append-only commit log which is replicated across the nodes running Kafka, otherwise known as the brokers. This allows the messages to be replayed or read after being sent. On the consumer side, nodes read messages from topics, which can be split into partitions and divided amongst available consumers. Kafka also guarantees fault tolerance for both the consumers and brokers.

Problem Statement

Coordination and resource sharing has always been a challenge in the distributed computing environment and can be improved with adaptations to the right protocols and techniques. Such scenarios typically arise in a high-traffic environment when the resources have to be balanced amongst the available worker nodes. The optimum way of distributing available resources is a challenge. Kafka provides fault tolerance for consumption by providing automatic rebalancing of consumption nodes when a new node is added to a consumer group or is removed/fails. The disadvantage of this process is that consumers cannot read from topic partitions while rebalancing is happening, since the consumer is being assigned new topic partitions to consume from. This forces consumption to stop, leading to zero throughputs from the affected consumer group.

Instead of rebalancing without considering the prior state, an alternative can use a greedy approach with respect to the previous state and the set of topic partitions to be allocated. Kafka provides in-built support for three reassignment strategies: range-based, round-robin, and sticky assignment.

These attributes make Kafka a great choice for enterprise integration, especially when building applications that scale. However, in order to fit Kafka to a particular use case, it's important to benchmark the performance by simulating a likely, everyday scenario. In this report, we examine the difference between two rebalancing strategies on consumption latency and based on the results discuss what is the optimal strategy for rebalancing consumers in a consumer group.

For instance, if there are 3 consumers in the consumer group (C0, C1 and C3) and 2 topics with 3 partitions each, the assignment will be as follows :

```
C0 = t0p0, t1p0
C1 = t0p1, t1p1
C2 = t0p2, t1p2
```

Rebalancing may occur in four cases here,

1. A new consumer gets added.
2. One of the consumer dies.
3. The group coordinator is unable to receive a heartbeat from one of the consumers.
4. A new partition is added.

When a new topic is added, the partition assignor strategy decides to which the consumer is the newly added topic assigned. The default partition strategy used by rebalancing is range based, however Kafka provides support for changing the partition strategy when adding a new consumer either in the consumer properties using the `partition.assignment.strategy` property or programmatically using the Java API.

Architecture

For our experiments, we use the same architecture in order to compare the different assignment strategies. The strategy makes use of a single consumer group with multiple consumers subscribed to partitions across two topics which are receiving data streams from a java application running locally.

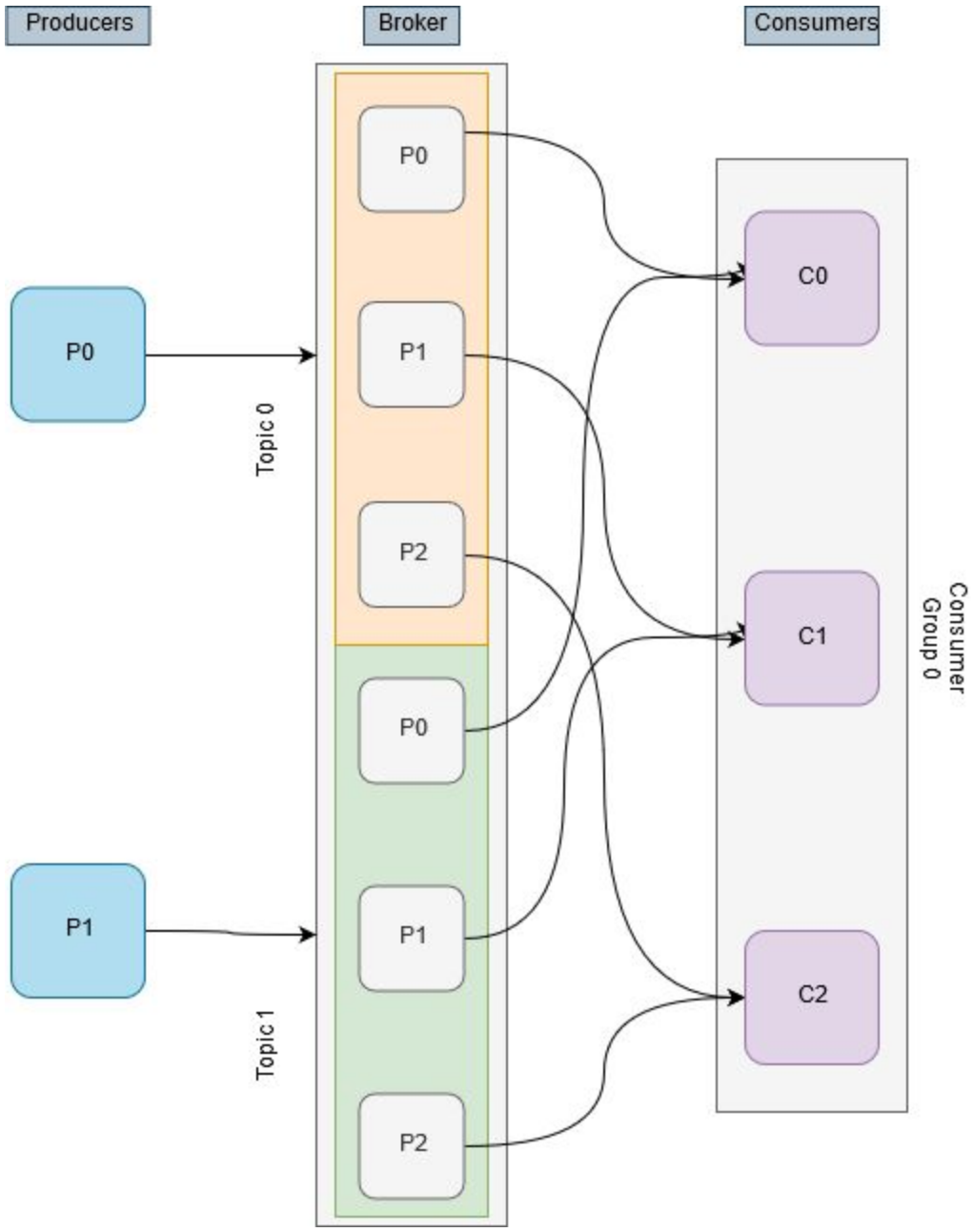

Figure 1. Initial Configuration of Kafka Cluster with Producers and Consumers

Benchmarking

Start Zookeeper :

```
>> zookeeper-server-start <path_to_config>/zookeeper.properties
Zookeeper is the central hub which stores the configuration metadata required for maintaining the cluster
This command looks for zookeeper-server-start file and the takes the zookeeper configuration file for the zookeeper properties.
```

Create topics :

```
>> kafka-topics --zookeeper localhost:2181 --create --topic t1 --replication-factor 3 --partitions 3
>> kafka-topics --zookeeper localhost:2181 --create --topic t2 --replication-factor 3 --partitions 3
>> kafka-topics --zookeeper localhost:2181 --create --topic t3 --replication-factor 3 --partitions 3
A topic is a feed name. It’s like the tables in sql except it’s no database.
```

create : it creates a new topic.
delete : it is used to delete a topic.
replication-factor : creates certain replicas of the topic.
Partitions : Topics are divided into partitions which divide the data across various brokers.

Startup brokers :

```
>> kafka-server-start.sh config/server.properties
>> kafka-server-start.sh config/server-1.properties
>> kafka-server-start.sh config/server-2.properties
```

For the experiment, 3 kafka servers were used.

Start console consumers :

```
>> kafka-console-consumer --bootstrap-server localhost:9092 --whitelist '<topic_name1>|<topic_name2>|<topic_name3>|<topic_name4>' --consumer-property group.id=<group_id>
```

--whitelist - Takes as an argument more than one topic. We used one consumer group and all the consumers had equal subscriptions so while creating each consumer process, all the topics were passed as an argument.

Confluent-control-center :

```
>> control-center-start ../etc/confluent-control-center/control-center.properties
Starts the confluent control center on port 9021.
```

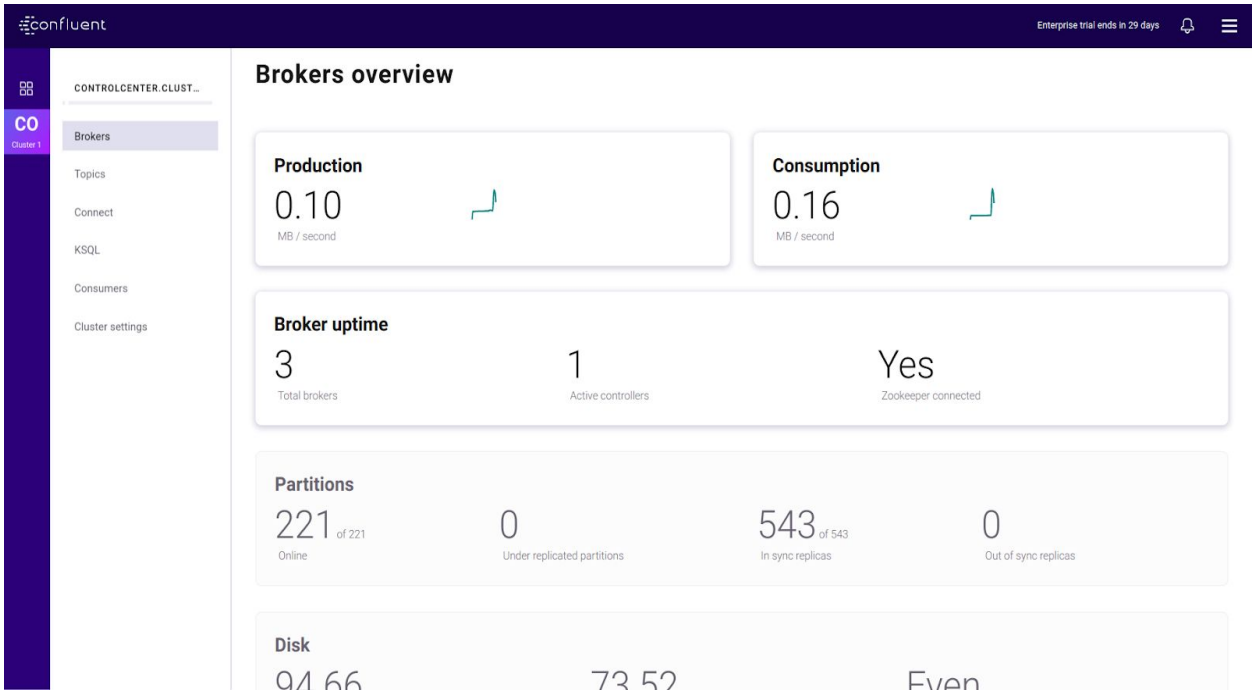

Fig 2. A confluent-center dashboard displaying the kafka cluster configuration metrics

Round-Robin assignment :

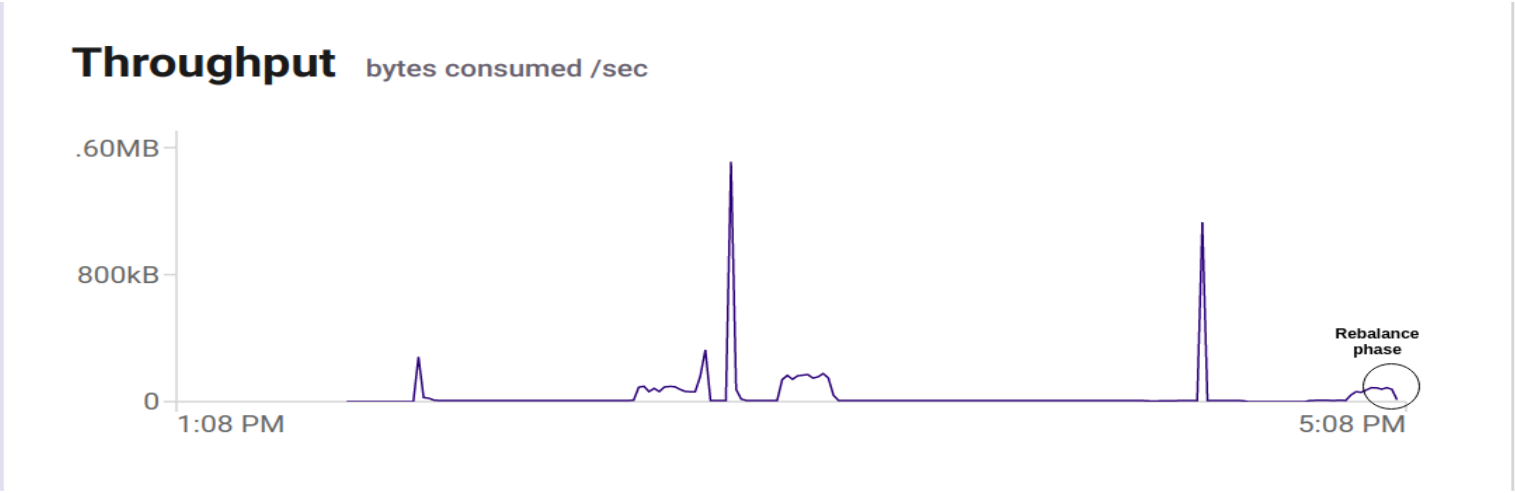

Figure 3. Rebalance throughput visualization

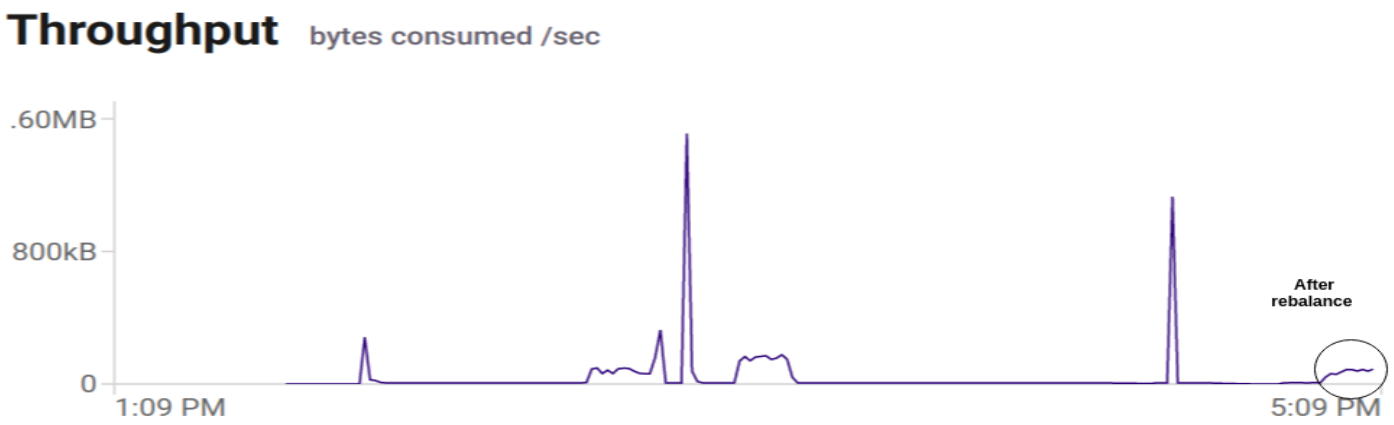

Figure 3. After Rebalance throughput visualization

Sticky assignment:

Figure 4. Rebalance phase throughput visualization

Improvements

Suppose there are two consumers C0, C1,C3 two topics t1, t2, and each topic has 3 partitions, resulting in partitions

```
t1p0, t1p1, t1p2,
t2p0, t2p1, t2p2.
```

Each consumer is subscribed to all the topics.

The assignment with both sticky and round-robin

Consumer	Topic - Partition Assignment
Consumer-1 (C0)	t1p0, t2p0
Consumer-2 (C1)	t1p1, t2p1
Consumer-3 (C2)	t1p2, t2p2

Suppose if a “consumer 2” is removed or loses connection from the consumer group, there is a reassignment of topic-partitions between the available consumers.

Round robin result :

Consumer	Topic - Partition Assignment
Consumer-1	t1p0, t1p2, t2p1
Consumer-3	t1p1, t2p0, t2p2

Changed assignments = 4
Preserved assignments = 2

Sticky assignor result :

Consumer	Topic - Partition Assignment
Consumer-1	t1p0, t2p0, t1p1
Consumer-3	t1p2, t2p2, t2p1

Changed assignments = 2
Preserved assignments = 4

Sticky Assignment rebalances states :

One Consumer :

Consumer Id	Topic Partition Topic	Partition	Lag Messages behind	Current offset	End offset
consumer-1-11663d4a-3615-4148-aa5c-c...	t1	1	0	105000	105000
consumer-1-11663d4a-3615-4148-aa5c-c...	t1	2	0	2000	2000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	0	0	303000	303000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	1	0	101000	101000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	2	0	204000	204000

Two Consumers :

Consumer Id	Topic Partition Topic	Partition	Lag Messages behind	Current offset	End offset
consumer-1-11663d4a-3615-4148-aa5c-c...	t1	1	0	105000	105000
consumer-1-ddbdea8b-e6c2-4105-a675-e...	t1	2	0	2000	2000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	0	0	303000	303000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	1	0	101000	101000
consumer-1-ddbdea8b-e6c2-4105-a675-e...	t2	2	0	204000	204000

Three Consumers:

Consumer Id	Topic Partition Topic	Partition	Lag Messages behind	Current offset	End offset
consumer-1-11663d4a-3615-4148-aa5c-c...	t1	0	0	201000	201000
consumer-1-ddbdea8b-e6c2-4105-a675-e...	t1	2	0	2000	2000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	0	0	303000	303000
consumer-1-9c4aef56-96c1-4173-b610-5...	t2	1	0	101000	101000
consumer-1-ddbdea8b-e6c2-4105-a675-e...	t2	2	0	204000	204000

Rebalance of Partitions:

Consumer Id	Topic Partition Topic	Partition	Lag Messages behind	Current offset	End offset
consumer-1-11663d4a-3615-4148-aa5c-c...	t1	0	0	201000	201000
consumer-1-11663d4a-3615-4148-aa5c-c...	t1	1	0	105000	105000
consumer-1-9c4aef56-96c1-4173-b610-5...	t1	2	0	2000	2000
consumer-1-9c4aef56-96c1-4173-b610-5...	t2	2	0	204000	204000
consumer-1-11663d4a-3615-4148-aa5c-c...	t2	0	0	303000	303000

Round Robin assignment states : (Too many shuffles)

Consumer Id	Topic Partition Topic	Partition	Lag Messages behind	Current offset	End offset
consumer-1-8d947cc9-ed1a-40f2-b094-8...	t4	2	0	0	0
consumer-1-32b55bab-1f7d-4d7f-81b9-6...	t4	0	0	0	0
consumer-1-876ab095-b8f9-46d1-9769-f...	t4	1	0	0	0
	test	0	0	0	0
consumer-1-8d947cc9-ed1a-40f2-b094-8...	t1	2	0	2312833	2312833

Consumer	Topic Partition		Lag		
Id	Topic	Partition	Messages behind	Current offset	End offset
consumer-1-8d947cc9-ed1a-40f2-b094-8...	t1	2	0	2312833	2312833
consumer-1-876ab095-b8f9-46d1-9769-f...	t1	1	0	4102859	4102859
consumer-1-32b55bab-1f7d-4d7f-81b9-6...	t1	0	0	3059151	3059151
consumer-1-8d947cc9-ed1a-40f2-b094-8...	t2	2	0	2902057	2902057
consumer-1-876ab095-b8f9-46d1-9769-f...	t2	1	0	2364345	2364345

Consumer	Topic Partition		Lag		
Id	Topic	Partition	Messages behind	Current offset	End offset
consumer-1-876ab095-b8f9-46d1-9769-f...	t2	1	0	2364345	2364345
consumer-1-32b55bab-1f7d-4d7f-81b9-6...	t2	0	0	3228644	3228644
consumer-1-8d947cc9-ed1a-40f2-b094-8...	t3	2	0	0	0
consumer-1-32b55bab-1f7d-4d7f-81b9-6...	t3	0	0	214348	214348
consumer-1-876ab095-b8f9-46d1-9769-f...	t3	1	0	257057	257057