Effectiveness of Rebalancing a Kafka Cluster

Interim Report by Bhubanesh Mishra

MSc Cloud Computing, Newcastle University

1 Introduction

In the modern era, almost every organization generates massive amounts of data every day. These data can vary from simple user end actions like logins, likes, comments, clicks and search, to more operational and system metrics data such as latency, errors, CPU usage, memory, network usage, to name a few [1]. The production of these data and the requirement for real-time analysis on this data as per the organizational needs creates a significant challenge for the systems to cope with these sheer volumes and provide optimal performance daily.

One of the well-known methods to manage these massive data in a distributed approach is to use the publish/subscribe messaging interaction paradigm [2]. In this publish/subscribe messaging system, the publisher categorizes the message somehow, and the recipient (subscriber) signs up to receive specific types of this communications. A broker, or a central location that broadcasts the messages, is standard in publish/subscribe systems [3]. Apache Kafka and RabbitMQ are the two most commercial open-source publish/subscribe systems used widely for managing the vast amount of data in the industries for the last couple of years [2]. Between these two, Apache Kafka showed superior performance when experimentally verified [1] and can even process millions of messages per second [4].

Apache Kafka is a distributed messaging platform capable of very high performance and horizontally scalable [5]. In Kafka, Topics stores the messages that get replicated and distributed across the Brokers, forming the cluster [2]. In an ideal scenario, for a balanced cluster, the Topics are evenly distributed amongst the Brokers. However, in practicality, due to faulty design of the system architecture, unpredictable workloads or addition/removal of brokers, the cluster could become unbalanced. Tools exist to rebalance a Kafka cluster by switching which Topic replica is the master or moving replicas to under-utilised Broker [6]. The latter operation of moving replicas can be expensive and take hours to migrate data between brokers [7]. The cluster operates at reduced capacity because the rebalance utilises the I/O and network bandwidth [7].

2 Aim and objectives

2.1 Aim

The high-level aim of the project is to investigate two things on Kafka which will be running on Kubernetes. Firstly, what are the optimal values for tuning a rebalance of a Kafka cluster to trade off speed versus impact? Secondly, a stretch goal is to look at the effects of rebalancing on a running cluster. For example, how long will it take? Will this have an impact on the current workload? What are the financial ramifications of charging for some data transfers but not others?

2.2 Objectives

These are the objectives of the project to achieve the aims mentioned above.

2.2.1 Literature Review

Extensive research work will be carried out to understand Kafka and previous works in rebalancing a Kafka cluster and explore automatic tools to achieve effectiveness.

2.2.2 Lab Set-up

A laboratory will host the test rig comprising the infrastructure, libraries, installable, and monitoring tools to carry out the experimental study.

2.2.3 Basic Metrics Gathering

Following the lab set-up, a balanced Kafka cluster will be created, and basic system and application metrics will be extracted from the monitoring tools. These metrics shall be the baseline for the entire experiment.

2.2.4 Unbalanced Cluster Creation

The Kafka cluster will be purposedly unbalanced, and the resultant metrics will be collected for comparing it with the metrics gathered from the balanced cluster.

2.2.5 Implementation of Cruise Control and Tuning

Various Cruise Control configuration execution on top of an unbalanced Kafka cluster will be performed to rebalance it and obtain the corresponding metrics. Furthermore, the rebalance will be fine-tuned by adjusting the various parameters, and the best settings will be obtained to help estimate and calculate trade-offs as per the first aim in 2.1.

2.2.6 Data Analysis

Data Analysis using Python, Pandas, and SciKit-Learn will be carried out by gathering all the system and application metrics data such as CPU, Memory, Network I/O, Consumer Lag and Consumer Group Lag, to name a few, in the baseline, unbalanced, cruise control and tuning stages. Moreover, using the results of this data analysis, estimations and conclusions will be made about the second aim, as mentioned above in 2.1. Therefore, finally helping in the critical evaluation of the project.

3 Progress Overview

Since this project work is being carried out as part of the Industry MSc project for Red Hat, the supervisors at Red Hat suggested few reading materials for getting acquainted with the world of Apache Kafka. The start date for this project activity was 19th April 2021.

The literature review was undertaken to understand the concepts of Apache Kafka. Firstly, why was Kafka needed and what business challenges made the creation of Kafka a necessity was explored. Following this, the salient concepts of Kafka such as installation requirements, Kafka Producers, Kafka Consumers, Kafka Brokers, language in which Kafka is written and general administration of a Kafka cluster was explored. Furthermore, the prime concepts of reliable data delivery and how to achieve it in Kafka were also researched. One of the principal resources helping in this research was the official Kafka guide [3], written by the creators of Kafka at LinkedIn themselves, which provided the complete understanding of Kafka, from the basics to the advanced level. Moreover, the utility of Zookeeper [8] in running a Kafka cluster was also probed.

Besides the understanding of Kafka, the impact of real-time workload and the resultant rebalancing mechanism was also scrutinised. Various methods used for rebalancing were studied [6], out of which the most appropriate method- the Cruise Control created by LinkedIn [7] was selected for our experimentation purpose in the project. Additionally, the Linux foundation entity Strimzi [9], using which the entire project activity was to be carried out as suggested by the Red Hat team, was also investigated.

Once the initial research was completed for Kafka, the next stage of the project work was undertaken: the set-up of Lab for experimenting. At first, an AWS cloud EC2 instance was selected to set-up the infrastructure for the test rig. The required installation files, python libraries were installed in that instance. Moreover, as it was suggested by the Red Hat team to make use of Strimzi – Kafka on Kubernetes concept to experiment; therefore, a Minikube [10] was installed in the instance. Inside this Minikube, a Kafka cluster was set up comprising a broker, bootstrap server, zookeeper, entity operator, and cluster operator by following the official Strimzi documentation [9] of Kafka.

Due to Pythons easy readability and coding capability, it was decided as the programming language for the application development for this project. Initially, a primary hello world type Python application was created to understand the flow of the producer-consumer model of Kafka on the freshly installed cluster inside Minikube. Following the successful understanding of this basic flow, a more complex pizza ordering python application was created, creating unlimited orders for pizza across various outlets and sending the respective order information from the producer to the consumer via a broker for consumption. Furthermore, this application was converted using the docker containerisation technique to include the multiple producers and consumers, for sending and receiving messages continuously. Therefore, creating a real-world data flow mechanism as desired in a balanced Kafka cluster.

Moreover, for gathering the metrics mentioned in objectives (2.2.3-2.2.5) for different project activities, the Prometheus-Grafana monitoring solution was also installed inside the Minikube and configured with the Kafka cluster. This activity concluded the Lab set-up work for the project. Below mentioned Figure-1 shows the sample screenshots of the successful set-up of the Lab and the primary metrics observed from the test rig.

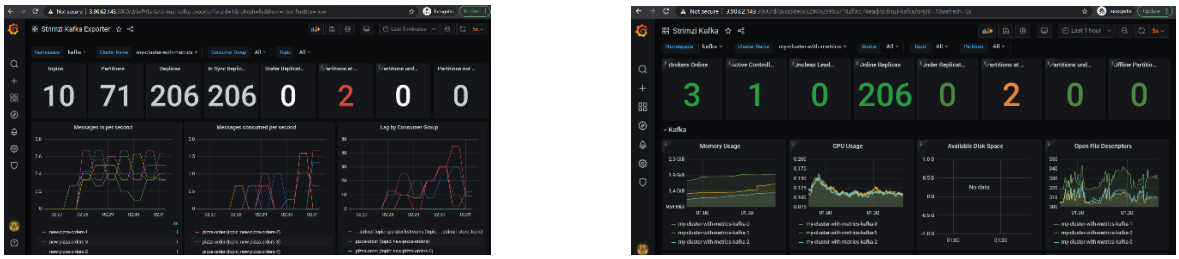

Figure-1: Screenshots of Kafka-Exporter and Kafka Metrics from Grafana

Currently, work is in progress to introduce scalability to this containerised application to proceed to the next stage of the project, gathering metrics from a balanced cluster involving multiple producers and consumers and create a baseline for the experiment. Besides, to observe the progress made till now, code developed and in-progress activities GitHub link is provided in Appendix 2.

4 Project Plan

The following Figure-2 shows the complete Project Plan in a Gantt-Chart format. The start date for the project was 19th April 2021, and the completion date is 20th August 2021.
Additionally, a comprehensive breakdown of the Gantt-Chart can also be viewed via the public URL link as provided in Appendix 1.

Task	Time in Weeks															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Lit Review																
Lab Setup																
Code Development																
Code Testing																
Data Analysis																
Summarising																
Report Writing and Presentation																
Project Buffer																
Submission																

Figure-2: Project Plan in Gantt Chart

References

[1] Kreps, J., Narkhede, N. and Rao, J., 2011, June. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (Vol. 11, pp. 1-7).

[2] Dobbelaere, P. and Esmaili, K.S., 2017, June. Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. In Proceedings of the 11th ACM international conference on distributed and event-based systems (pp. 227-238).

[3] Narkhede, N., Shapira, G. and Palino, T., 2017. Kafka: the definitive guide: real-time data and stream processing at scale. " O'Reilly Media, Inc.".

[4] Goodhope, K., Koshy, J., Kreps, J., Narkhede, N., Park, R., Rao, J. and Ye, V.Y., 2012. Building LinkedIn's Real-time Activity Data Pipeline. IEEE Data Eng. Bull., 35(2), pp.33-45.

[5] Thein, K.M.M., 2014. Apache kafka: Next generation distributed messaging system. International Journal of Scientific Engineering and Technology Research, 3(47), pp.9478-9483.

[6] Kumar, I., 2018. Autonomous Workload Rebalancing in Kafka.

[7] Qin, 2016, 'Introduction to Kafka Cruise Control', Stream Processing Meetup, LinkedIn, Mountain View, CA, 2 November, 2016, viewed 10 June, 2021, https://www.slideshare.net/JiangjieQin/introduction-to-kafka-cruise-control-68180931.

[8] Hunt, P., Konar, M., Junqueira, F.P. and Reed, B., 2010, June. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In USENIX annual technical conference (Vol. 8, No. 9).

[9] Strimzi Documentation v0.22, Retrieved 10 June, 2021, from https://strimzi.io/docs/operators/0.22.0/overview.html

[10] Moilanen, M., 2018. Deploying an application using Docker and Kubernetes.

Appendix

1 https://sharing.clickup.com/g/h/69qng-120/06dbf6b36741b4a
2 https://github.com/bhuvigithub/Kafka-Project