On Efficiently Partitioning a Topic in Apache Kafka

Theofanis P. Raptis and Andrea Passarella
Institute of Informatics and Telematics, *National Research Council*, Pisa, Italy. Email: {name.surname}@iit.cnr.it

***Abstract*—Apache Kafka addresses the general problem of delivering extreme high volume event data to diverse consumers via a publish-subscribe messaging system. It uses partitions to scale a topic across many brokers for producers to write data in parallel, and also to facilitate parallel reading of consumers. Even though Apache Kafka provides some out of the box optimizations, it does not strictly define how each topic shall be efficiently distributed into partitions. The well-formulated fine-tuning that is needed in order to improve an Apache Kafka cluster performance is still an open research problem. In this paper, we first model the Apache Kafka topic partitioning process for a given topic. Then, given the set of brokers, constraints and application requirements on throughput, OS load, replication latency and unavailability, we formulate the optimization problem of finding how many partitions are needed and show that it is computationally intractable, being an integer program. Furthermore, we propose two simple, yet efficient heuristics to solve the problem: the first tries to minimize and the second to maximize the number of brokers used in the cluster. Finally, we evaluate its performance via large-scale simulations, considering as benchmarks some Apache Kafka cluster configuration recommendations provided by Microsoft and Confluent. We demonstrate that, unlike the recommendations, the proposed heuristics respect the hard constraints on replication latency and perform better w.r.t. unavailability time and OS load, using the system resources in a more prudent way.**

***Index Terms*—Apache Kafka, publish-subscribe, distributed systems, event-store, stream processing**

I. Introduction

Data stream processing is gaining more and more attention as a new implementation paradigm to manage real-time data-driven applications in numerous smart-* domains, such as cities [1], industry [2], networking [3], and agriculture [4]. This paradigm is more suitable to modern application developers that consist of vertically separated engineering teams, each one managing a loosely coupled sub-system. In comparison to old-fashioned data-driven applications with centralized, shared data management systems, the distributed and non-synchronous nature of stream handling enables stakeholders to construct their sub-systems in an event-driven way which heavily relies to event-data passing [5]. The sub-systems get activated with local, asynchronous state updates in the presence of loosely coupled interactions which would typically need to be centrally managed and synchronized.

Apache Kafka was initially implemented so as to address the generic challenge of delivering extreme high volume event data to different consumers. Apache Kafka handles data as a partitioned write-ahead commit log on persistent storage and implements a pull-based messaging method to enable both real-time consumers such as online services and offline consumers such as Hadoop to consume those data at their own pace. During the last decade, Apache Kafka has become one of the most successful Apache open source products and is widely appreciated in the distributed systems community.

A representative case of Apache Kafka adoption is the largest professional social network on the Internet, LinkedIn. This network produces billions of events that reflect its over-300 million members' social interactions and actions every day and routes them to its products. Product examples include "Who's Viewed My Profile?" which is an info and news feeds, "People You May Know" which is an ML-based social and content recommendation system, and various other back-end monitoring and reporting systems like site health auditing processes as well as account fraud detection tools. Therefore, the ability to deliver high-volume event data in real time to both LinkedIn user-facing products as well as its back-end systems is highly important. With this objective under consideration, Apache Kafka has been developed as LinkedIn's main online data handling framework [6].

Apache Kafka is a publish-subscribe data stream service, where a producer sends data to an Apache Kafka topic (a logical category of data) in a set of data brokers (the Apache Kafka cluster), and a consumer receives the data from the subscribed topic upon request. A topic may be stored in ≥ 1 partitions (the physical storage of data in the Apache Kafka cluster), and all the partitions of the topic are distributed among the brokers of the cluster. Apache Kafka utilizes partitions to distribute a topic across many brokers for producers to send data in parallel, and also to enable parallel reception of consumers. Although Apache Kafka gives the capability to employ 4.000 partitions per broker and 200.000 partitions per cluster [7], it does not strictly define how each topic shall be efficiently divided to partitions or how shall the partitions be efficiently allocated to brokers. Moreover, to the best of our knowledge, there has been no research study in the state-of-the-art to address this emerging issue in a systematic manner, even in the case of a single topic.

A. Novelty of the Current Study

As discuss in detail in section I-B, there has been no study in the state-of-the-art to (i) model the Apache Kafka topic partitioning process in a fine-grained, combinatorial manner (even for a single topic), (ii) identify, formulate and characterize the emerging related optimization problem(s), and, (iii) provide efficient algorithmic solutions by taking into account the application requirements. Some questions arise, for example, how many partitions shall be allocated to a

This work was funded by the European Union's Horizon 2020 research and innovation programme MARVEL under grant agreement No 957337. This publication reflects the authors views only. The European Commission is not responsible for any use that may be made of the information it contains.

topic? How many brokers shall be used? Or, how to effectively respect the application constraints?

In the following sections we address such questions and provide an evaluation of our design. In section II, we provide the Apache Kafka basics. In section III, we model the application constraints and requirements. In section IV, we formulate the problem that emerges from the modeling assumptions. In section V, we design two heuristic algorithms for addressing the problem. In section VI, we conduct a performance evaluation against some configuration recommendations from the state-of-the-art and we highlight the strengths of our heuristics.

B. Related works

Although the related state-of-the-art has not researched the systematic formulation of topic partitioning in Apache Kafka deployments, there has been a notable corpus of works regarding various aspects of Apache Kafka functionalities. We briefly report those works here, so as the reader can grasp the current status of the area.

- *Apache Kafka fundamentals*. In [8], the authors use formal methods to model the communication between producers and consumers in Apache Kafka. They also apply the model checking tools to verify five properties of the system. In [9], the authors develop a custom made Apache Kafka consumer, which enhances Apache Kafka with robust, scalable and smart filtering and queuing mechanisms to achieve effective rate adjustment. In [10], the authors promote the improvement of fault tolerance in the Apache Kafka pipeline architecture by defining optimal checkpoint interval values which impact the data recovery of the original Apache Kafka pipeline. The design is proved to reduce the number of lost data by using the optimal checkpoint interval time. In [11], the authors analyze the structure and workflow of Apache Kafka and propose a queuing based packet flow model to predict performance metrics of Apache Kafka cloud services.
- *Apache Kafka applications.* In [6] the authors replicate Apache Kafka logs for various distributed data-driven systems at LinkedIn, including source-of-truth data storage and stream processing. In [12], the authors design a distributed cluster processing model based on Apache Kafka data queues, to optimize the inbound efficiency of seismic waveform data. In [13], the authors extend Apache Kafka by building an in-memory distributed complex event recognition engine built on top of Apache Kafka streams. In [14], the authors design a simulation platform enabling evaluations of future mobility scenarios, based on an Apache Kafka architecture. In [15], the authors break the streaming pipeline into two distinct phases and evaluate percentile latencies for two different networks, namely 40GbE and InfiniBand EDR (100Gbps), to determine if a typical streaming application is network intensive enough to benefit from a faster interconnect. In [16], the authors propose a distributed framework for the application of stream processing on heterogeneous environmental data, which addresses the challenges of data heterogeneity from heterogeneous systems and offers real-time processing of huge environmental datasets through a publish/subscribe method via a unified data pipeline with the application of Apache Kafka for real-time analytics. In [17], the authors find that filtering on large datasets is best done in a common upstream point instead of being pushed to, and repeated, in downstream components. To demonstrate the advantages of such an approach, they modify Apache Kafka to perform limited native data transformation and filtering, relieving the downstream Spark application from doing this.
- *Apache Kafka performance evaluation*. In [18], the authors propose an approach to track a specific vehicle over the video streams published by the collaborating traffic surveillance cameras. In [19], the authors make an evaluation of several configurations and performance metrics of Apache Kafka in order to allow users avoid bottlenecks and eventually leverage some good practice for efficient stream processing. In [20], a performance study shows that estimates regarding the performance impact of different Apache Kafka configurations, based on experiences and perceptions, are not always true. In [21], the authors propose an Apache Kafka-based load shedding engine which operates when the latency exceeded the threshold, and discards data in the Apache Kafka producer for quick overload control. In [22], the authors conduct an analysis of Apache Kafka's network fault tolerance capabilities. Across different Apache Kafka configurations, they observe that Apache Kafka is fault-tolerant towards network faults to some degree, and they report observations of its shortcomings. In [23], the authors compare RabbitMQ with Apache Kafka based on the core functionalities of pub/sub systems and they venture into a qualitative and empirical comparison of the common features of the two systems.
- *Apache Kafka latency and reliability*. In [24], the authors test the impact of numerous configuration parameters on the reliability of Apache Kafka, including retry strategies and replications of partitions for fault tolerance. In [25], the authors build an Apache Kafka testbed using Docker containers to analyze the distribution of Apache Kafka's end-to-end latency. In [26], the authors find that changing configuration parameters can significantly impact the guarantee of data delivery in Apache Kafka. In [27], the authors introduce a tool for testing the reliability of Apache Kafka so as to study different data delivery semantics in Apache Kafka and compare their reliability under poor network quality.

II. Apache Kafka basics

Apache Kafka provides a publish-subscribe data streaming service, in which p producers send data to an Apache Kafka topic τ (a logical category of data) stored in a set of b data brokers (the Apache Kafka cluster), and c consumers read data

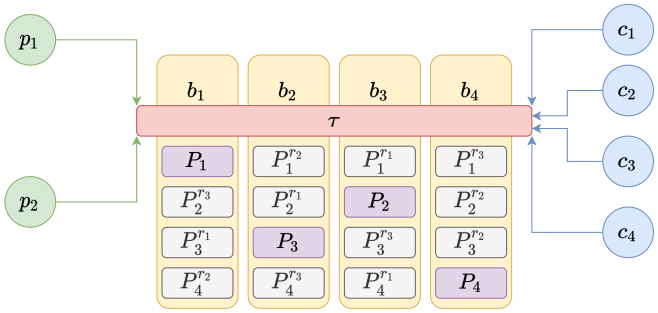


Fig. 1: An example of an Apache Kafka configuration for a given topic τ, with $p = 2, c = 4, b = 4, P = 4, r = 3$. Leader partitions colored in purple, replicas in gray.

from the topic. The topic is stored in $P \geq 1$ partitions (the physical storage of data in the Apache Kafka cluster), and all the partitions of the same topic are distributed among the brokers. Apache Kafka uses partitions to scale a topic across many brokers for producers to write data in parallel, and to further enable parallel reading of consumers. Every partition is able to be replicated across the Apache Kafka brokers for purposes of fault tolerance with a replication factor r. The replication helps in maintaining high availability in case one of the brokers goes down and is unavailable to serve the requests. A partition can be replicated across multiple replicas, each one stored on a different broker. One of the replicas is elected as leader and the remaining replicas are labeled as followers. Apache Kafka administers all the replicas automatically and ensures that they are kept in sync. The requests of both a producer and a consumer to a partition are served through leader replica. It is just the leader partition that manages all the data reads and writes of the interested producers and consumers, in a FIFO manner. An example of an Apache Kafka configuration is shown in Fig. 1.

When a consumer group is subscribed to a topic, each consumer instance in the group reads data from a different subset of the partitions in the topic in parallel. A consumer instance can read data from multiple partitions, while one partition has to be read by only one consumer instance within the same consumer group. Different consumer groups can independently consume the same data and no coordination among consumer groups is deemed necessary. Consequently, the number of partitions defines the maximum extent of parallelism within a consumer group.

Producers and consumers can produce and consume increased volumes of data or generate requests at a very high rate and thus reserve broker resources, trigger network saturation and reduce the QoS of other consumers and brokers. Although Apache Kafka clusters can artificially apply quotas on requests in order to administer the broker resources utilized by producers and consumers, the quota functionality can not be applied in some application verticals that require high network bandwidth or high request rates.

The data topic partitioning reflects the extent of parallelism in Apache Kafka [28]. Latency significantly depends on the application design for handling data, and therefore optimizing it involves tuning the operational logic. On the other hand, the cluster throughput depends on the cluster parameters, and therefore, the broker and partition optimization can potentially increase efficiency. As the amounts of produced data increases, the number of partitions typically needs to scale up so as to prevent bottlenecks. High latency in the data pipeline can result in tremendous lag on the consumer side. The absence of an adequate number of partitions to achieve the required throughput can lead to time outs on the requests from producers to brokers and corresponding data loss. This major risk necessitates a balanced throughput and latency in the Apache Kafka cluster.

III. Modelling application constraints and requirements

A. Data throughput

In principle, as the number of partitions in an Apache Kafka cluster increases, so does also the throughput one can achieve. Therefore, a viable objective would be to maximize the number of partitions within the cluster:

$$\max P \tag{1}$$

Writes to different partitions can be performed completely in parallel on both the producer and the broker side. As mentioned beforehands, in the consumer side, within a consumer group, Apache Kafka always maps the data of a single partition to a single consumer. Consumers can select both the topic and partition from which they want to read data. If consumers have not specified any partition, Apache Kafka can automatically choose more than one, based on the number of available partitions. A topic, is not able to support more consumers than partitions. Consequently, the extent of parallelism in the consumer side is bounded by the number of partitions able to be consumed:

$$P \geq c \tag{2}$$

We denote the target Apache Kafka cluster throughput as T. If the throughput that can be achieved on a single partition for production and consumption (T_p and T_c respectively) is known (through measurements), then we need to have at least the following number of partitions:

$$P \geq \max\left(\frac{T}{T_p}, \frac{T}{T_c}\right) \tag{3}$$

Except for throughput, there are a few other factors that have to be considered when optimizing the number of partitions. In some cases, having too many partitions may also have negative impact [29], which we highlight in the following sections.

B. OS load

Each partition corresponds to a directory in the broker's file system. In this directory, there are typically two files, one for the index and another for the data per log entry. Each broker maintains a single file handle for both the index and the data file of every log segment. Therefore, as the number of partitions increases, the open file handle limit in the OS

must increase as well. Based on the above, if $H_{\max}$ denotes the maximum number of open file handles that a broker can tolerate, and b the number of brokers, then the number of partitions in the Apache Kafka cluster is bounded by

$$P \cdot r \leq b \cdot H_{\max} \tag{4}$$

C. Replication latency

We consider as end-to-end latency in Apache Kafka the time required for the data to be published by the producer until the time that the data are read by the consumer. Apache Kafka provides access of data to a consumer after the point of time when the data has been replicated to all the in-sync replicas. Therefore, a significant portion of the end-to-end latency is considered the time to commit data. Typically, an Apache Kafka broker uses a single thread to replicate data from another broker for all partitions that they mutually share. Assuming a given replication factor r, we denote the replication latency as l_r.

We can improve replication latency via building larger Apache Kafka clusters. For example, suppose that x partition leaders reside on a broker and that the replication process necessitates in this case y ms of time. If we add z brokers in the same Apache Kafka cluster ($x > z$), each of the z brokers needs to fetch only x/z partitions on average from the initial broker. Therefore, the added latency of committing data will be in the order of just y/z ms in this case, instead of y ms. Depending on the application area, the application owner might enforce a requirement of a specific (end-to-end, and therefore) replication latency threshold, which we denote as L. Therefore, this threshold bounds the related system configurations as follows:

$$\frac{P \cdot r}{b} \cdot l_r \leq L \tag{5}$$

D. Unavailability

The intra-cluster replication of Apache Kafka leads to higher availability and durability. However, when a broker fails, partitions with a leader on that broker become temporarily unavailable. Apache Kafka then automatically assign the leader role of those unavailable partitions to other replicas that continue serving the consumers. This is performed vis one of the Apache Kafka brokers designated as the controller, typically in serial fashion.

In specific cases, the observed unavailability can be proportional to the number of partitions. Indicatively, for the case of a broker that has a total of x partitions, each with y replicas: Roughly, this broker will be the leader for about x/y partitions. If this broker fails, the x/y partitions become unavailable at the same time. If z ms time is required to elect a new leader for one partition, then it will require up to zx/y ms to elect a new leader for all x/y partitions. Therefore, for several partitions, the observed unavailability can be zx/y ms plus the time taken to detect the failure. Therefore, given an observed unavailability time u during a broker failure, the application unavailability requirement threshold U shall bound the configuration as follows:

$$\frac{P}{b} \cdot u \leq U \tag{6}$$

IV. The problem

Based on the descriptions of section III, there are contrasting forces defining the number of partitions P to be used, so choosing the "right" number of partitions per topic requires knowledge of many factors. Increasing the partition density (the number of partitions per broker) adds an overhead related to metadata operations and per partition request/response between the partition leader and its followers. Increasing the number of partitions in a cluster though will lead to increased parallelism of data consumption, which in turn improves the throughput of an Apache Kafka cluster; however, the time required to replicate data across replica sets will also increase [30].

Even though Apache Kafka provides some out of the box optimizations, the well-formulated fine-tuning that is needed in order to improve cluster performance is still an open research problem. Based on equations/inequalities 1-6, we formulate the problem as the following program (the program variables are marked in bold font):

Objective:

$$\max \quad \boldsymbol{P} \tag{7}$$

Constraints:

$$\boldsymbol{P} - \max\left(\frac{T}{T_p}, \frac{T}{T_c}, c\right) \geq 0 \qquad \text{(throughput)} \tag{8}$$

$$\boldsymbol{P} \cdot r - \boldsymbol{b} \cdot H_{\max} \leq 0 \qquad \text{(OS load)} \tag{9}$$

$$\boldsymbol{P} \cdot r \cdot l_r - \boldsymbol{b} \cdot L \leq 0 \qquad \text{(replication latency)} \tag{10}$$

$$\boldsymbol{P} \cdot u - \boldsymbol{b} \cdot U \leq 0 \qquad \text{(unavailability)} \tag{11}$$

$$r \leq \boldsymbol{b} \leq B \tag{12}$$

$$\boldsymbol{P}, \boldsymbol{b} \in \mathbb{Z}^+ \tag{13}$$

The objective function (7) maximizes the number of partitions in the cluster. Constraint (8) guarantees that the selected number of partitions renders the resulting throughput viable. Constraint (9) guarantees that the OS load in terms of open file handles can be supported by the system in place. Constraint (10) guarantees that the replication latency does not exceed the maximum latency threshold provided by the operator. Constraint (11) guarantees that potential broker unavailability do not exceed the maximum unavailability threshold provided by the operator. Constraint (12) guarantees that the number of selected brokers does not exceed the actual number B of available brokers in the cluster.

The problem is computationally intractable, since it is formulated as an integer program (all of the variables are restricted to be nonzero positive integers, through constraints (13)). This means that we are not able to optimally maximize the exact number of partitions needed to satisfy all the system constraints for any given instance of the problem in polynomial time.

Algorithm 1: BroMin

Parametrical input: T, c, r, L, U, B
Measured input: $T_p, T_c, H_{\max}, l_r, u$
Output: P, b

1 **for** $b = r;\ b \leq B;\ b++$ **do**
2 **for** $P = \left\lfloor \frac{b \cdot H_{\max}}{r} \right\rfloor;\ P \geq \max\left(\frac{T}{T_p}, \frac{T}{T_c}, c\right);\ P--$ **do**
3 **if** $P \cdot r \cdot l_r \leq b \cdot L$ ***and*** $P \cdot u \leq b \cdot U$ **then**
4 return P, b;
5 return *"No feasible solution found."*

Algorithm 2: BroMax

Parametrical input: T, c, r, L, U, B
Measured input: $T_p, T_c, H_{\max}, l_r, u$
Output: P, b

1 **for** $b = B;\ b \geq r;\ b--$ **do**
2 **for** $P = \left\lfloor \frac{b \cdot H_{\max}}{r} \right\rfloor;\ P \geq \max\left(\frac{T}{T_p}, \frac{T}{T_c}, c\right);\ P--$ **do**
3 **if** $P \cdot r \cdot l_r \leq b \cdot L$ ***and*** $P \cdot u \leq b \cdot U$ **then**
4 return P, b;
5 return *"No feasible solution found."*

V. Maximizing the number of partitions

The naive way to solve the problem is to simply remove the constraint that P and b are integers (therefore enforce $b, P \in \mathbb{R}$ instead of constraints (13)), solve the corresponding linear relaxation of the integer program, and then round the entries of the solution to the linear relaxation. But, not only may this solution not be optimal, it may not even be feasible, as it could potentially violate some constraint.

In order to address this challenge, we designed Algorithms 1 and 2 for a topic partitioning with ensuring adequate throughput, low OS load, low replication latency and high availability. Each of the two algorithms targets at fully exploiting a given broker's resources and at minimizing (Algorithm 1 - BroMin) or maximizing (Algorithm 2 - BroMax) the number of brokers used. Although the two approaches seem to follow an opposite line of though, they both try to maximize the number of partitions needed in the Apache Kafka cluster to satisfy the constraints, but with a different side-objective: use as few HW resources as possible in the first case and take advantage of all the available HW resources in the second case.

More specifically, both algorithms receive a set of parametrical inputs ($T, c, r, H_{\max}, L, U, B$) and a set of measured inputs (T_p, T_c, l_r, u). Each algorithm starts gradually increasing or decreasing correspondingly the number of brokers to be investigated, with a lowest number equal to the replication factor r and highest equal to the number of available brokers B (line 1). Then, for the selected broker number, both algorithms start decreasing the number of investigated partitions P, starting from the maximum allowed number per broker w.r.t. open file handles, and finishing at the minimum allowed number w.r.t. the desired cluster throughput and consumer support (line 2). The algorithms then return as a solution, the first occurence of P, b combination that satisfies the replication latency and unavailability time constraints (line 3-4). As usual in the case of heuristic approaches, the inherent disadvantage is that if the algorithm fails to find a solution (line 5), it cannot be determined whether it is because there is no feasible solution or whether the algorithm simply was unable to find one. Further, it is usually impossible to quantify how close to optimal a solution returned by these methods are. In order to address the last concern, we perform a large scale simulation study in section VI.

VI. Performance evaluation

In this section we perform an extensive simulation evaluation of our algorithms in Octave 6.2.0.

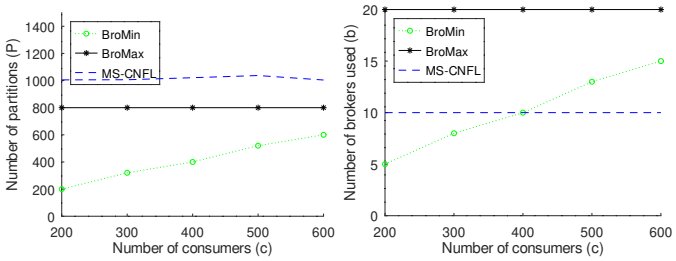
A. Parameters' configuration

We assign values to the algorithm measured inputs according to verified state-of-the-art measurements. For example, the per-partition throughput T_p that one can achieve on the producer depends on configurations such as the batching size, compression codec, type of acknowledgement, replication factor, etc. However, as shown in [31], in general, one can produce at $T_p = 10$ MB/sec on just a single partition. The consumer throughput T_c is often application dependent since it corresponds to how fast the consumer logic can process data. In our case we set it indicatively to $T_c = 20$ MB/sec. Furthermore, in [28], it is noted that it can take around 5 ms to elect a new leader for a single partition, and therefore set the equivalent unavailability time to $u = 5$ ms. The same report, also claims that production Apache Kafka clusters can run with > 30.000 open file handles per broker. In our case, in order to be able to capture also weaker settings, we set $H_{\max} = 10.000$. Next, from the report's contents, it can be assumed that the replication latency for a single partition (without considering other partitions) could be as low as $l_r = 1$ ms. Last but not least, the report states that leader election for a single partition can take 5 ms, and therefore we set the unavailability time as $u = 5$ ms. The parametrical inputs T, c, r, L, U and B, depend on the application requirements. Different applications might come with completely different values of those parameters. In our simulations, we variate the values of c, B and r, and we set $T = 100$ MB/s, $L = 200$ ms and $U = 2000$ ms.

B. Performance metrics and benchmark

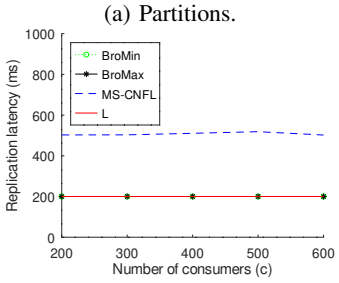
According to [30], there are three main metrics that can reflect the performance of an Apache Kafka cluster:

1) System throughput: In our case, this is captured by the ultimate number of partitions selected by each algorithm; as mentioned earlier, the number of partitions is the unit of parallelism in Apache Kafka. In our two algorithms, the desired throughput T is also taken into account as a hard constraint.

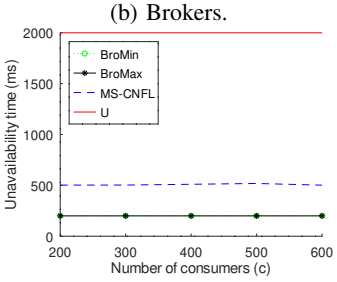


(a) Partitions.

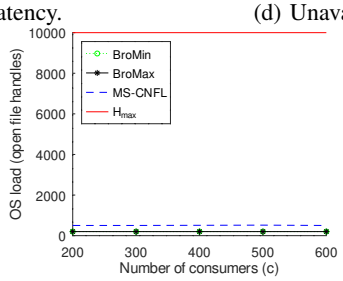
(b) Brokers.



(c) Latency.

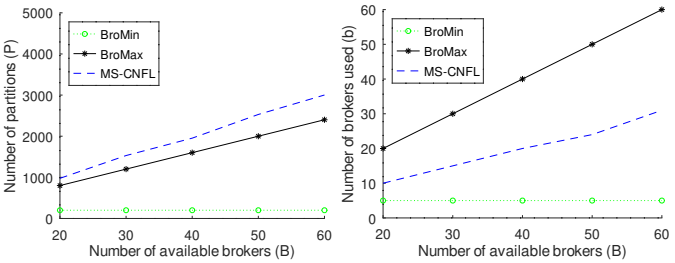


(d) Unavailability.



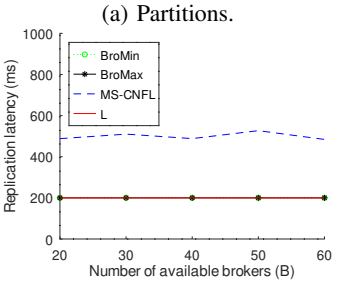
(e) OS load.

Fig. 2: Performance for variable numbers of consumers.

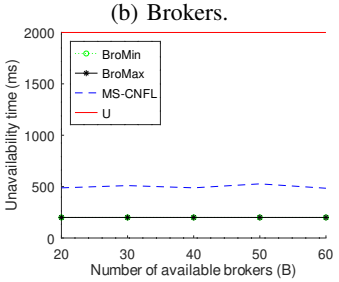


(a) Partitions.

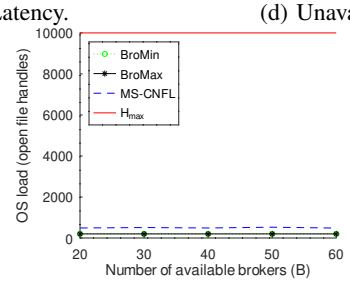
(b) Brokers.



(c) Latency.



(d) Unavailability.



(e) OS load.

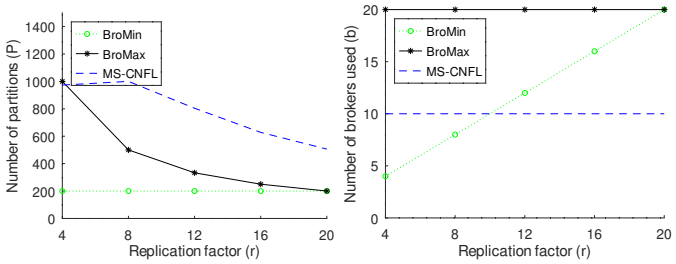
Fig. 3: Performance for variable numbers of available brokers.

2) Latency: the amount of time that is needed to process data, in the sense of time required for data to be stored or retrieved. The dominant factor in this procedure is the replication time of the data. We capture this metric by appropriately measuring the l_r, but also through a hard constraint on L.
3) The numbers or costs of the application's infrastructure: In our case, that would be the number of brokers used in the Apache Kafka cluster. This metric explains the diversity of the two introduced algorithms.

We additionally measure the OS load metric via the open file handles and the unavailability metric via the unavailability time. Given the absence in the literature of a well-defined similar method which targets at efficiently partitioning an Apache Kafka topic, in order to adequately benchmark the performance of our algorithms, we carefully studied the commercial best practices in the related industry. We found out specific Apache Kafka configuration recommendations used by service well known providers. Specifically, Microsoft, is stating as a coarse rule in [32] that it is generally recommended to not exceed 1.000 partitions per broker (including replicas). Also, Confluent, as a rule of thumb, states in [28], that if latency is a core objective, it would be probably efficient to limit the number of partitions P per broker b to $100 \cdot B$ ($\cdot r$ if we include the replicas). Therefore, taking into account those two sets of configuration recommendations, we arrive at the following benchmark method, which we call MS-CNFL: $P = \min\left(P \in_R [1...\frac{1.000 \cdot B}{r}], P \in_R [1...100 \cdot B]\right)$ and $b \in_R [1...B]$, where $\in_R$ denotes uniformly random selection.

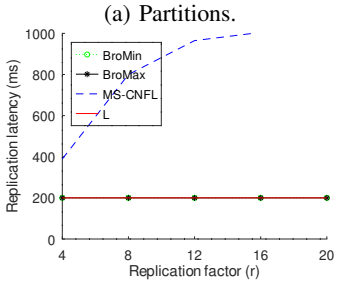
C. Performance results

Figure 2 displays the four selected metrics for a variable number of consumers. BroMax and MS-CNFL maintain a constant number of partitions, regardless of the number of consumers in the system (Fig. 2a), with MS-CNFL maintaining a larger number of partitions. On the contrary, BroMin is gradually increasing the number of partitions according to the number of the consumers in the system. A similar behavior is observed in the number of brokers used by each method (Fig. 2b). BroMax and MS-CNFL maintain a constant number of brokers, regardless of the number of consumers in the system, and BroMin is gradually increasing the number of brokers according to the number of the available consumers in the system. In this case, we can see that BroMin is choosing to use more brokers than MS-CNFL for more than 400 consumers in the system. Regarding the hard constraints of latency, unavailability and OS load (Fig. 2c, Fig. 2d and Fig. 2e respectively), we can see that while BroMin and BroMax respect both application constraints, MS-CNFL is violating the latency constraint. This performance can be explained by simultaneously observing Fig. 2a and Fig. 2b: MS-CNFL is maintaining more partitions than BroMin and BroMax, but at

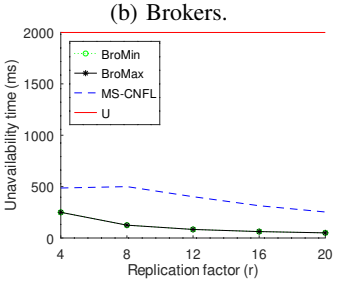


(a) Partitions.

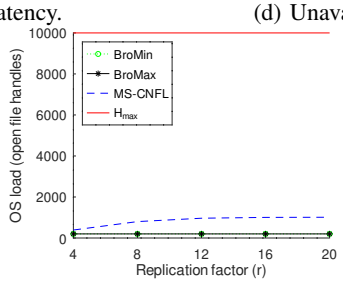
(b) Brokers.



(c) Latency.



(d) Unavailability.



(e) OS load.

Fig. 4: Performance for variable replication factor values.

the same time, tries to distribute them to a non-optimized, insufficient number of brokers. This choice is especially aggravated for more than 400 consumers in the system, where even BroMin (which tries to minimize the number of brokers required to perform well) uses more brokers than MS-CNFL. But even in the cases where the hard constraints are not violated by MS-CNFL (OS load and unavailability), BroMin and BroMax achieve a better performance with a lower number of partitions.

Figure 3 displays the four selected metrics for a variable number of available brokers in the cluster. Unlike the previous case of variable consumers, BroMax and MS-CNFL are gradually increasing the number of partitions according to the number of the available brokers in the cluster (Fig. 3a), with MS-CNFL maintaining a larger number of partitions. On the contrary, BroMin maintains a constant number of partitions regardless of the number of the available brokers in the cluster. A similar behavior is observed in the number of brokers used by each method (Fig. 3b). BroMax and MS-CNFL are gradually increasing the number of brokers according to the number of the available brokers in the cluster, and BroMin maintains a constant number of partitions regardless of the number of the available brokers in the cluster. Regarding the hard constraints of latency, unavailability and OS load (Fig. 3c, Fig. 3d and Fig. 3e respectively), we can see that, similar to the previous case, while BroMin and BroMax respect both application constraints, MS-CNFL is violating the latency constraint. This performance can be again explained by simultaneously observing Fig. 3a and Fig. 3b: MS-CNFL is maintaining more partitions than BroMin and BroMax, but at the same time, tries to distribute them to a non-optimized, insufficient number of brokers. Even in the cases where the hard constraints are not violated by MS-CNFL (OS load and unavailability), BroMin and BroMax achieve a better performance with a lower number of partitions.

Figure 4 displays the four selected metrics for a variable replication factor. BroMax and MS-CNFL are gradually decreasing the number of partitions according to the replication factor (Fig. 4a), with MS-CNFL maintaining a larger number of partitions. On the contrary, BroMin maintains a constant number of partitions regardless of the replication factor. A different behavior is observed in the number of brokers used by each method (Fig. 4b). BroMax and MS-CNFL maintain a constant number of brokers, regardless of the replication factor, and BroMin is gradually increasing the number of brokers according to the replication factor. In this case, we can see that BroMin is choosing to use more brokers than MS-CNFL for a replication factor value of more than 10. Regarding the hard constraints of latency, unavailability and OS load (Fig. 4c, Fig. 4d and Fig. 4e respectively), we can see that, in this case, while BroMin and BroMax respect both application constraints, MS-CNFL is not only violating the latency constraint but the violation extent is significantly aggravated for higher replication factor values. This performance can be explained by simultaneously observing Fig. 4a and Fig. 4b: MS-CNFL is maintaining more partitions than BroMin and BroMax, but at the same time, tries to distribute them to a non-optimized, insufficient number of brokers. This choice is especially aggravated for a replication factor of more than 10, where even BroMin (which tries to minimize the number of brokers required to perform well) uses more brokers than MS-CNFL. But even in the cases where the hard constraints are not violated by MS-CNFL (OS load and unavailability), BroMin and BroMax achieve a better performance with a lower number of partitions.

By combining the findings displayed in each subfigure, we are led to the following conclusions: Both BroMin and BroMax try to use the system resources in a more prudent way than the MS-CNFL configuration recommendations, in the sense that, although MS-CNFL eventually distributes the topic into more partitions, it does not efficiently address the application constraints. In the case of latency constraints, MS-CNFL even violates the given thresholds, especially when the replication factor r becomes higher and higher. On the other hand, BroMin tries to find a delicate balance between the number of partitions and the number of brokers used in the cluster (so as to also enhance efficient resource utilization), and BroMax, using the maximum number of brokers that can be used, tries to maximize the number of partitions by taking into account the application constraints.

VII. Conclusions and future work

In this paper, after taking into account the Apache Kafka basics, we provided a modeling approach that considers the most important application constraints and requirements. For the first time in the state-of-the-art, given an Apache Kafka topic, we formulated the topic partitioning the problem and we showed its computational intractability, being expressed as an integer program with an objective function of maximizing the topic's partitions. Then, we designed two heuristic algorithms for addressing the problem and we conducted a performance evaluation against Apache Kafka partitioning configuration recommendations provided by Microsoft and Confluent. We demonstrated that both our heuristics use the system resources in a more prudent way than the benchmark method, and address well the constraints of replication latency, resource usage, OS load and unavailability. Our simulation-based study led us to the conclusion that systematic topic partitioning makes sense in Apache Kafka from the point of view of efficiency. As future work, we are planning to perform real-world tests on the Apache Kafka infrastructure that is under development in the EU H2020 MARVEL project.

References

[1] D. Bajovic *et al.*, "Marvel: Multimodal extreme scale data analytics for smart cities environments," in *2021 International Balkan Conference on Communications and Networking (BalkanCom)*, 2021, pp. 143–147.

[2] T. P. Raptis, A. Passarella, and M. Conti, "Distributed data access in industrial edge networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 915–927, 2020.

[3] ——, "Performance analysis of latency-aware data management in industrial iot networks," *Sensors*, vol. 18, no. 8, 2018.

[4] C. M. Angelopoulos, G. Filios, S. Nikoletseas, and T. P. Raptis, "Keeping data at the edge of smart irrigation networks: A case study in strawberry greenhouses," *Computer Networks*, vol. 167, p. 107039, 2020.

[5] G. Wang *et al.*, *Consistency and Completeness: Rethinking Distributed Stream Processing in Apache Kafka*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2602–2613.

[6] ——, "Building a replicated logging system with apache kafka," *Proc. VLDB Endow.*, vol. 8, no. 12, p. 1654–1655, aug 2015.

[7] J. Rao, "Apache kafka supports 200k partitions per cluster," November 2018, [Posted 08-November-2018, Accessed 06-April-2022]. [Online]. Available: https://www.confluent.io/blog/apache-kafka-supports-200k-partitions-per-cluster/

[8] J. Xu, J. Yin, H. Zhu, and L. Xiao, "Modeling and verifying producer-consumer communication in kafka using csp," in *7th Conference on the Engineering of Computer Based Systems*, ser. ECBS 2021. New York, NY, USA: Association for Computing Machinery, 2021.

[9] M. Tsenos, N. Zacheilas, and V. Kalogeraki, "Dynamic rate control in the kafka system," in *24th Pan-Hellenic Conference on Informatics*, ser. PCI 2020, 2020, p. 96–98.

[10] T. Aung, H. Y. Min, and A. H. Maw, "Enhancement of fault tolerance in kafka pipeline architecture," in *Proceedings of the 11th International Conference on Advances in Information Technology*, ser. IAIT2020. New York, NY, USA: Association for Computing Machinery, 2020.

[11] H. Wu, Z. Shang, and K. Wolter, "Performance prediction for the apache kafka messaging system," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 154–161.

[12] X.-C. Chai *et al.*, "Research on a distributed processing model based on kafka for large-scale seismic waveform data," *IEEE Access*, vol. 8, pp. 39 971–39 981, 2020.

[13] S. Langhi, R. Tommasini, and E. D. Valle, "Extending kafka streams for complex event recognition," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 2190–2197.

[14] M. Gütlein and A. Djanatliev, "On-demand simulation of future mobility based on apache kafka," in *Simulation and Modeling Methodologies, Technologies and Applications*, M. S. Obaidat, T. Oren, and F. D. Rango, Eds. Cham: Springer International Publishing, 2022, pp. 18–41.

[15] M. H. Javed, X. Lu, and D. K. D. Panda, "Characterization of big data stream processing pipeline: A case study using flink and kafka," in *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, ser. BDCAT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–10.

[16] A. Akanbi, "Estemd: A distributed processing framework for environmental monitoring based on apache kafka streaming engine," in *2020 the 4th International Conference on Big Data Research (ICBDR'20)*, ser. ICBDR 2020. ACM, 2020, p. 18–25.

[17] E. Falk, V. K. Gurbani, and R. State, "Query-able kafka: An agile data analytics pipeline for mobile wireless networks," *Proc. VLDB Endow.*, vol. 10, no. 12, p. 1646–1657, aug 2017.

[18] S. Kul, I. Tashiev, A. Şentaş, and A. Sayar, "Event-based microservices with apache kafka streams: A real-time vehicle detection system based on type, color, and speed attributes," *IEEE Access*, vol. 9, pp. 83 137–83 148, 2021.

[19] P. Le Noac'h, A. Costan, and L. Bougé, "A performance evaluation of apache kafka in support of big data streaming applications," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 4803–4806.

[20] G. Hesse, C. Matthies, and M. Uflacker, "How fast can we insert? an empirical performance evaluation of apache kafka," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, 2020, pp. 641–648.

[21] H. Kim *et al.*, "Message latency-based load shedding mechanism in apache kafka," in *Euro-Par 2019: Parallel Processing Workshops*, U. Schwardmann *et al.*, Eds. Cham: Springer International Publishing, 2020, pp. 731–736.

[22] M. Raza *et al.*, "Benchmarking apache kafka under network faults," in *Proceedings of the 22nd International Middleware Conference: Demos and Posters*, ser. Middleware '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 5–7.

[23] P. Dobbelaere and K. S. Esmaili, "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*, 2017, p. 227–238.

[24] H. Wu, "Research proposal: Reliability evaluation of the apache kafka streaming system," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 112–113.

[25] H. Wu, Z. Shang, G. Peng, and K. Wolter, "A reactive batching strategy of apache kafka for reliable stream processing in real-time," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 207–217.

[26] H. Wu, Z. Shang, and K. Wolter, "Learning to reliably deliver streaming data with apache kafka," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 564–571.

[27] ——, "Trak: A testing tool for studying the reliability of data delivery in apache kafka," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 394–397.

[28] J. Rao, "How to choose the number of topics/partitions in a kafka cluster?" March 2015, [Posted 12-March-2015, Accessed 06-April-2022]. [Online]. Available: https://www.confluent.io/blog/how-choose-number-topics-partitions-kafka-cluster/

[29] A. Povzner and S. Hendricks, "99th percentile latency at scale with apache kafka," February 2020, [Posted 25-February-2020, Accessed 06-April-2022]. [Online]. Available: https://www.confluent.io/blog/configure-kafka-to-minimize-latency/

[30] N. Salinger, "Optimizing kafka performance," [Accessed 06-April-2022]. [Online]. Available: https://granulate.io/optimizing-kafka-performance/

[31] J. Kreps, "Benchmarking apache kafka: 2 million writes per second (on three cheap machines)," April 2014, [Posted 27-April-2014, Accessed 06-April-2022]. [Online]. Available: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines/

[32] "Performance optimization for apache kafka hdinsight clusters," December 2021, [Posted 13-December-2021, Accessed 06-April-2022]. [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/kafka/apache-kafka-performance-tuning/