Future Generation Computer Systems 154 (2024) 173–188

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications

Theofanis P. Raptis, Claudio Cicconetti *, Andrea Passarella

Institute of Informatics and Telematics, National Research Council, Pisa, Italy

ARTICLE INFO

Keywords:
Fog computing
Data streaming
Topic partitioning
Distributed systems

ABSTRACT

Apache Kafka is a widely-used event streaming platform for reliable high-volume real-time data exchange following a producer–consumer pattern. Despite its popularity, Apache Kafka requires expertise and attention to detail, and there are no default guidelines that can be applied to all use cases without careful consideration. In this paper, we propose a novel approach to optimise the number of partitions and brokers in Apache Kafka, which are two key configuration parameters, under the given characteristics and constraints of the target applications. In particular, we consider the distribution of data-intensive real-time flows exchanged between a set of producers and consumers, which is representative of fog computing environments for ML/AI analytics. We introduce a methodology for modelling the topic partitioning process in Apache Kafka and formulate an optimisation problem to determine the optimal number of partitions to satisfy the application requirements and constraints. We propose two efficient heuristics to solve the optimisation problem, considering the trade-off between resource utilisation and application performance. We evaluate the performance of our approach through numerical simulations, and we demonstrate its practicality by implementing a prototype on an Apache Kafka cluster and conducting experiments in three different scenarios focused on mass consumption vs. production and real-time data streaming. To carry out repeatable experiments in controlled conditions, we developed a reusable framework that fully automatises cluster setup and performance assessment, and we make it available to the community as open-source software.

1. Introduction

Fog computing has been adopted in many verticals [1] for processing and analysing data in real-time, especially for applications that require low latency and high bandwidth. It allows for faster processing of data and reduces the need to transfer large amounts of data to a centralised cloud, thereby improving the overall system performance. However, managing resources in such high-volume, real-time fog environments can be challenging. Fog nodes have limited processing power, memory, and storage, which means that resource allocation must be carefully managed to meet the applications' expectations under performance trade-offs. Moreover, the dynamic and heterogeneous nature of fog environments poses additional challenges in terms of resource allocation [2].

To effectively allocate resources in fog environments, several factors need to be considered, including the characteristics of the application, the available resources, and the network conditions [3]. The challenge is to allocate resources in such a way that the application can meet its performance requirements while utilising the available resources efficiently [4]. In this respect, resource allocation is not a one-size-fits-all problem, and different applications may have different requirements and constraints. For instance, some applications may require more processing power, while others may need more memory or storage [5]. Therefore, it is essential to have a flexible and adaptable resource allocation mechanism that can be customised for specific applications and their requirements.

Apache Kafka[1] is an event streaming platform known for its ability to reliably handle high-volume real-time data exchange [6]. It follows a producer–consumer (pub-sub) pattern, where producers generate data and consumers consume it for various purposes. While there are several mature technologies that can fulfil this role, e.g., in [7] the authors have investigated two serverless data pipeline approaches designed with

* Corresponding author.
E-mail addresses: t.raptis@iit.cnr.it (T.P. Raptis), c.cicconetti@iit.cnr.it (C. Cicconetti), a.passarella@iit.cnr.it (A. Passarella).

[1] https://kafka.apache.org/

https://doi.org/10.1016/j.future.2023.12.028
Received 19 September 2023; Received in revised form 16 November 2023; Accepted 23 December 2023
Available online 8 January 2024

Message Queuing Telemetry Transport (MQTT) and Apache NiFi, Apache Kafka is extremely popular due to its ability to handle massive amounts of data in real time, making it an ideal platform for use in fog computing environments [8]. For example, Apache Kafka was used as the core technology for streaming concurrent audio/video flows from sensors deployed in smart cities to ML/AI processors in the edge-fog-cloud continuum, in the project MARVEL[2] funded by under the H2020 programme of the European Commission.

One of the key features of Apache Kafka is its ability to partition data across multiple servers, which is not fixed and can be configured by the application provider [9]. This allows for data to be distributed across multiple nodes in an Apache Kafka cluster, which provides fault tolerance and ensures that data can be processed quickly and efficiently [10]. Additionally, Apache Kafka allows for data replication, which ensures that data are not lost in the event of a node failure. However, all this flexibility, while advantageous, also requires careful optimisation to achieve efficient data distribution: failure to properly configure Apache Kafka can result in degraded performance and even system failures, as the default configuration does not work optimally for all use cases. Therefore, it is essential to optimise the number of partitions and brokers in Apache Kafka based on the unique characteristics and constraints of the target application. In many cases, experts need to identify for each specific use case a suitable methodology to determine the best number of partitions/brokers to meet the application requirements and constraints [11].

Despite the popularity and extensive use of Apache Kafka in production, the research on data topic partitioning remains limited. Although a considerable amount of work has been presented on optimising other aspects of Apache Kafka, such as data stream processing, the topic partitioning problem has not been extensively researched. This is mainly due to the fact that topic partitioning is highly dependent on the application-specific requirements, and there is no one-size-fits-all solution [12]. On the one hand, most use case owners tend to focus on significantly fine-grained optimisations that cater to their specific requirements, rather than considering a more generalised approach [13]. As a result, the literature lacks research on how to optimise the number of partitions and brokers, which are two key configuration parameters that significantly affect Apache Kafka's performance. On the other hand, although industrial corporate companies offer some recommendations for Apache Kafka configuration, these recommendations may not always lead to optimal performance. This is because the recommendations are typically derived from a general set of assumptions, and do not consider the specific characteristics and requirements of individual applications [14]. Consequently, a well-designed approach that is tailored to the characteristics of the target application may have the potential to outperform these recommendations significantly.

This work addresses the aforementioned limitations of the state-of-the-art by proposing a novel approach to configure efficient Apache Kafka topic partitioning for applications with real-time data streaming with high-reliability requirements. Our contribution is two-fold.[3] On the one hand, we formulate an optimisation problem to determine the optimal number of partitions to satisfy the application requirements and constraints, for which we propose two heuristic algorithms that are representative of the trade-off between resource utilisation and application performance. The problem formulation takes into account inherent application requirements, and therefore, the algorithms, which receive the values of those requirements as input, lead to application-driven solutions. Indeed, our approach is based on the idea of achieving a balanced allocation of resources while avoiding under- or over-utilisation of Apache Kafka resources. The heuristics are evaluated through numerical analysis in scenarios with a large number of producers/consumers and a broad set of configurations. On the other hand, we demonstrate the practicality of our approach by implementing a prototype on an Apache Kafka cluster and conducting real experiments, using a reusable framework that fully automates cluster setup and performance assessment and allows the evaluation of throughput or end-to-end latency metrics.

The structure of the paper is as follows. In Section 2 we illustrate the Apache Kafka fundamentals and review the state of the art. The effective design of Apache Kafka topic partitioning for reliable data streaming applications is elaborated in Section 3, which describes the system model and provides a mathematical formulation of the related optimisation problem, for which two heuristics are proposed and evaluated through numerical simulations. Furthermore, in Section 4 we present a reusable framework for the fully automated execution of prototype experiments under controlled and repeatable conditions, which is then used to verify the effectiveness of the topic partitioning model and techniques in a realistic setting. Section 5 concludes the paper.

2. Background

In this section, we provide the reader with the background necessary to fully appreciate and position our work. In Section 2.1 we illustrate the key design principles of Apache Kafka, while the more relevant works in the literature are reviewed in Section 2.2. Further to this material, for a deeper understanding of the Apache Kafka inner mechanisms, the reader can refer to [16].

2.1. Apache Kafka fundamentals

A pub-sub data streaming service is offered by Apache Kafka, in which data are sent by p producers to a topic (a logical category of data) τ stored in a collection of b data brokers (the Apache Kafka cluster), and read by c consumers. In the Apache Kafka cluster, the topic is physically stored in $P \geq 1$ partitions, and all of the partitions of the same topic are spread across the brokers. For producers to publish data in parallel across several brokers and for consumers to read data in parallel, Apache Kafka employs partitions. With a replication factor of r, each partition may be duplicated across Apache Kafka brokers for fault tolerance. If one of the brokers fails and cannot fulfil requests, the replication aids in ensuring high availability. Multiple clones of a partition may be created, each of which is kept on a distinct broker. The remaining replicas are designated as followers, and one of the replicas is chosen to be the leader. All of the replicas are automatically maintained and kept in sync using Apache Kafka. The leader replica fulfils the demands made of a partition by both a producer and a consumer. Only the leader partition controls all of the interested producers' and consumers' FIFO data reads and writes. Fig. 1 displays an illustration of an Apache Kafka configuration.

Each consumer instance in a group of subscribers to a topic consumes data concurrently from a separate subset of the topic's partitions. While one partition must be read by only one consumer instance within the same consumer group, a consumer instance can read data from many partitions. The same data may be used independently by various consumer groups, and no coordination is required. As a result, the quantity of partitions determines the degree of parallelism that may be

[2] https://www.marvel-project.eu/

[3] The current version of our paper expands on a previous conference version [15] in several ways. First, we have introduced a new methodology for the automated performance evaluation of Apache Kafka clusters. This methodology allowed us to run experiments in a testbed prototype to compare BroMax and BroMin with different message sizes, replication factors, and numbers of consumers. By doing so, we have been able to obtain real-life data and cross-check our numerical simulation results. Second, we have included the results of these experiments, which have highlighted some qualitative differences with respect to the conclusions obtained with the numerical simulations alone. These differences emphasise the importance of cross-checking analysis with real-life data before production deployment. Lastly, we have made our evaluation framework available to the community so that others can use it for their own performance evaluations.

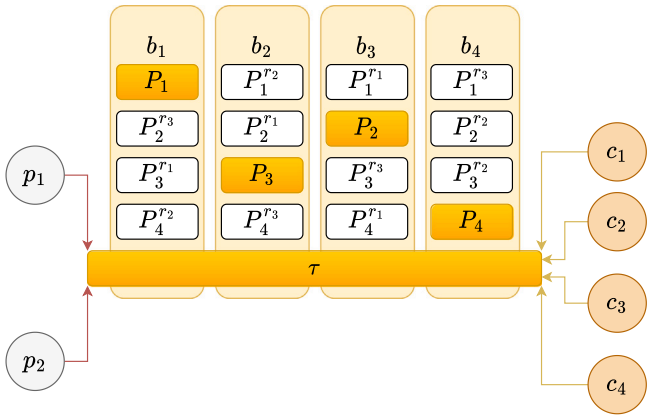

Fig. 1. An indicative illustration of an Apache Kafka cluster for a topic τ, with $p = 2, c = 4, b = 4, P = 4, r = 3$. Leader partitions are coloured in orange, and replicas coloured in white.

used by a consumer group. We observe that the number of instances of consumers in the group should not exceed the number of partitions in the subscribed topic, otherwise, there will be idle consumers. In our analysis, we will consider that all the available consumer instances belong to a single consumer group.[4]

High-volume data producers and consumers might reserve broker resources, cause network saturation, and degrade the QoS of other consumers and brokers by generating requests at a very high rate. Although Apache Kafka clusters can impose artificial quotas on requests to manage the broker resources used by producers and consumers, the quota capability cannot be used in specific application domains that need a significant amount of network bandwidth or a very high request rate. The degree of parallelism in Apache Kafka is reflected in the data topic partitioning [17]. Tuning the operational logic is necessary to reduce latency since it heavily depends on how an application is designed to handle data. However, because the throughput of the cluster is dependent on the cluster characteristics, broker and partition optimisations may boost efficiency. The number of partitions often has to grow up as the amount of data produced increases to avoid bottlenecks. Significant lags on the consumer side can be caused by high pipeline latency. Timeouts on the requests from producers to brokers and related data loss might result from not having enough partitions to handle the entire throughput.

2.2. Related works and novelty of the current study

The combinatorial aspects of Apache Kafka form a significant part of its algorithmic foundations, providing powerful approaches for optimising large-scale data processing and streaming. Combinatorial optimisation is concerned with finding the best solutions from a finite set of possibilities. In the context of Apache Kafka, this involves maximising the throughput of data streams, reducing latency, and minimising resource usage. The literature offers only a few rigorous works on optimisation techniques used in Apache Kafka, such as batch processing, compression, and data transfer.

In [16], the communication in Apache Kafka between producers and consumers is modelled using formal methods. Selected system characteristics are verified using the model testing tools. The verification findings demonstrate that the Apache Kafka data transfer model adheres to its specifications, which leads to the conclusion that the system is trustworthy. In [18], in order to forecast performance measures of Apache Kafka cloud services, the authors analyse the structure and workflow of Apache Kafka and suggest a queuing-based data transfer model. A second important category of previous works is on how to solve issues to partition-consumer assignments. The authors of [19] use bin packing problem variation to abstract the challenge of finding the necessary number of consumers and the partition-consumer assignments. They suggest indicative metrics that take the rebalancing expenses into consideration and introduce and evaluate a variety of methods in comparison to known strategies for the bin packing problem in this context. In [20], the authors look into an adaptive tuning method to calibrate the parameters related to an Apache Kafka consensus algorithm based on feedback control theory for blockchain-specific use case applications. According to the authors of [21], data starvation may occur if Apache Kafka's data production rate outpaces its consumption rate. A load-shedding method is introduced to restrict the incoming data and keep the efficiency high when the system is under stress to address the starvation issue. In [22], a simulation platform that allows assessments of potential future mobility use cases is described by the authors. To support all of these needs and the coupling of various simulation tools into a co-simulation, Apache Kafka is used as a communication building block.

Our proposed contribution makes a novel addition to the research on resource allocation and optimisation in fog environments with Apache Kafka. Specifically, our approach addresses the optimisation of the number of Apache Kafka topic partitions and brokers, which are key configuration parameters that impact the performance of the system. To the best of our knowledge, this is the first work that proposes a modelling and simulation perspective to evaluate the effectiveness of related resource allocation and scheduling in Apache Kafka. Moreover, our approach proposes two efficient heuristics to solve the optimisation problem, which is formulated to determine the optimal number of partitions to satisfy the application requirements and constraints. This trade-off between resource utilisation and application performance is carefully considered in our approach, and we demonstrate its practicality through numerical simulations and a real implementation on an Apache Kafka cluster. Our proposed approach has the potential to significantly improve the performance of Apache Kafka-based systems, especially in high-volume, real-time fog environments.

3. Effective design of Apache Kafka topic partitioning

In this section, we tackle the effective design of topic partitioning in Apache Kafka when used for real-time data streaming applications, which requires careful design and evaluation.

3.1. System model

We now introduce a modelling approach that provides a more practical and scalable alternative to the traditional methods for optimising Apache Kafka topic partitioning. Our approach is based on the use of combinatorial modelling, which allows us to test different configurations of the system and evaluate their performance under various conditions. This approach enables us to effectively abstract the dynamic and complex nature of the Apache Kafka topic partitioning process and consider the inherent uncertainties in the system. Our approach offers an efficient way to design efficient heuristic algorithms which are application-driven (as they take as input the values of the application requirements) and optimise Apache Kafka topic partitioning while providing more accurate performance predictions. In the following, we describe the main components of our approach and explain how they work together to optimise Apache Kafka topic partitioning for a specific application.

[4] Assuming a single consumer group for all instances simplifies the analysis but has limitations. It may not accurately model load balancing, isolation, and different commit strategies found in real-world scenarios with multiple consumer groups. These simplifications can impact the precision of the model in reflecting complex Apache Kafka consumer behaviours.

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 maximise 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 beforehand, on 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 do not specify any partition, Apache Kafka automatically chooses more than one for them, based on the number of available partitions. Anyway, a topic cannot support more consumers than partitions. Consequently, the extent of parallelism on 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 optimising the number of partitions. In some cases, having too many partitions may also have a negative impact [23], which we highlight in the following sections.

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

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

Replication latency

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

We can improve replication latency by 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 replication latency threshold, which we denote as L. Therefore, this threshold bounds the related system configurations as follows:

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

Table 1
Notation used in the paper.

Symbol	Description
P	Number of partitions
c	Number of consumers
T	Target throughput
T_p	Production throughput
T_c	Consumption throughput
$H_{\max}$	Maximum number of open file handles per broker
b	Number of brokers
B	Number of brokers available
r	Replication factor
l_r	Replication latency
L	Replication latency threshold
u	Observed unavailability time
U	Unavailability threshold

Unavailability

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

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

3.2. Problem formulation

We now formally define the partition/broker assignment problem, using the notation defined in Section 3.1, which is summarised in Table 1.

The number of partitions P to be employed is determined by conflicting forces according to the description of components in Section 3.1, therefore it is application driven, and selecting the "right" number of partitions for each topic necessitates knowledge of a variety of variables. An additional burden for metadata operations and request/response between the partition leader and its followers is added when the partition density, i.e., the number of partitions per broker, increases. Increasing the number of partitions in a cluster will result in more concurrent data consumption, which in turn enhances the throughput of an Apache Kafka cluster, but it will also increase the amount of time needed to replicate data between replica sets [24]. Even though Apache Kafka comes with certain built-in optimisations, the well-defined fine-tuning required to boost cluster performance is still an unsolved research issue. We construct the issue as the following programme (the programme variables are denoted by bold font) based on the equations and inequalities (1)–(6).

Objective:

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

Constraints:

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

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

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

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

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

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

The objective function (7) maximises 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 the potential broker unavailability does 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 formulation is considered computationally intractable since it is an integer programme. The precise number of partitions required to fulfil all of the system requirements for any particular instance of the issue cannot, therefore, be ideally maximised in polynomial time.

The choice of the "maximising the number of partitions" as an objective function in our study may seem unconventional at first glance. However, while objectives such as minimising unavailability or maximising throughput are indeed critical considerations in system design, the number of partitions plays a pivotal role in achieving these broader objectives in Apache Kafka [25]. The number of partitions directly influences the parallelism and distribution of data processing within a Kafka cluster. A higher number of partitions may allow for finer granularity in workload distribution among brokers, leading to optimised resource utilisation and reduced contention. By maximising the number of partitions, we aim to strike a balance between efficient resource utilisation and achieving effective throughput, which indirectly contributes to minimising unavailability by preventing bottlenecks and reducing processing latency. Thus, the "number of partitions" objective is intricately linked to the broader goals of enhancing system efficiency, serves as a key lever in achieving these objectives and is, therefore, a valid consideration in our study.

3.3. Two heuristics for solving the problem

The easiest manner to address the problem is: taking away the constraint regarding P's and b's integer status, i.e., modifying (13) so that $b, P \in \mathbb{R}$; providing a solution to the resulting linear relaxation of the integer programme; and, finally rounding the values of the solution. However, this way, there is a high chance that in many problem instances, the solution will not be optimal, or even feasible, due to the fact that it might violate any of the given constraints.

Therefore, we provide Algorithms 1 and 2 that target to define the partitioning of a given topic while maintaining the necessary throughput, low OS load, efficient replication latency and high availability. The goals of the two algorithms are to maximise (Algorithm 2 - BroMax) or minimise (Algorithm 1 - BroMin) the number of brokers employed while fully using the capabilities of each individual broker. Although the two strategies appear to pursue different paths, they both aim to maximise the number of partitions required in the Apache Kafka cluster to meet the constraints, though with different side objectives: use the fewest amount of HW resources possible in the first strategy and utilise all hardware resources available in the second strategy. Eventually, both algorithms provide a (potentially different) solution to the problem.

Both algorithms are given a set of measured inputs (T_p, T_c, l_r, u) and a set of parameter values ($T, c, r, H_{\max}, L, U, B$). With the lowest number equal to the replication factor r and the highest number equal to the number of available brokers B, each algorithm begins by progressively

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 ≤ B; b++ do
2     for P = ⌊b·H_max / r⌋; P ≥ max(T/T_p, T/T_c, c); P-- do
3         if P·r·l_r ≤ b·L and P·u ≤ b·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 ≥ r; b-- do
2     for P = ⌊b·H_max / r⌋; P ≥ max(T/T_p, T/T_c, c); P-- do
3         if P·r·l_r ≤ b·L and P·u ≤ b·U then
4             return P, b;
5 return "No feasible solution found."
```

raising or lowering the number of brokers to be assigned. Following that, both algorithms begin reducing the number of partitions P for the chosen number of brokers, starting at the highest allowed number per broker in terms of open file handles and ending at the lowest allowed number in terms of the desired cluster throughput and consumer support (line 2).

The first instance of the P, b combination that fulfils the replication latency and unavailability time restrictions is then returned by the algorithms as a solution (line 3–4). The underlying drawback of heuristic techniques is that, as is typically the case, it is impossible to tell if an algorithm fails to discover a solution (line 5) because there is no workable solution or because it was unable to do so. Furthermore, the degree to which a solution given by these approaches is optimum is sometimes hard to measure. We do a numerical simulation analysis in Section 3.4 to address the last point. The computational complexity of both algorithms is $\mathcal{O}(\frac{H_{\max}}{r} B^2)$.

The objective of maximising the number of partitions stems from the need to efficiently distribute processing load, ensuring that data is evenly processed across the Kafka cluster. While it may appear counter-intuitive to aim for higher resource consumption, the primary goal is to optimise workload distribution. BroMax (which maximises the number of brokers assigned to partitions) does so with the intention of avoiding overloading a single broker, thereby preventing resource contention and improving overall system performance. On the other hand, BroMin (which minimises the number of brokers assigned to partitions) seeks to strike a balance between distribution and resource conservation, potentially benefiting from brokers with greater available resources. Both BroMax and BroMin aim to address the complex trade-off between efficient resource utilisation and workload distribution. Therefore, by providing a range of heuristic strategies, our approach accommodates varying system requirements and preferences. Also, since the algorithm inputs are application-dependent (and given that the algorithms return a solution that is always feasible for the given inputs), their philosophy is application-driven.

3.4. Performance evaluation via numerical simulations

We conclude the section by presenting our numerical simulation results and the experimental evaluation setup, which enables us to

Table 2
Numerical simulation parameter settings.

Parameter	Value
c	200–600
T_p	10 MB/s
T_c	20 MB/s
$H_{\max}$	10.000
B	20–60
r	4–20
l_r	1 ms
L	200 ms
u	5 ms
U	2000 ms

analyse and evaluate the performance of our approach. We utilise a combination of simulations and prototype experiments to validate the efficacy of our approach in a high-volume, real-time fog environment. As mentioned in previous sections, our modelling approach involves constructing a detailed model that captures the behaviour of the Apache Kafka brokers, producers, and consumers under different workload conditions. The model is built based on a set of input parameters that represent the various system parameters, such as network latency, throughput, message size, and processing time. We assign values to these parameters based on state-of-the-art measurements, as discussed in the previous section.

In this section, we present the results of our extensive numerical simulation evaluation, performed in Octave 6.2.0. According to validated state-of-the-art measurements, we assign values to the algorithm's measured inputs. For instance, parameters like batching size, compression codec, type of acknowledgement, replication factor, etc. affect the per-partition throughput T_p that may be achieved on the producer. On a single partition, however, one may typically ingest data at $T_p = 10$ MB/s, as demonstrated in [26]. Since the consumer throughput T_c reflects how quickly the consumer logic can process data, it frequently depends on the application. In our situation, we set it to $T_c = 20$ MB/s as an example. Additionally, it is mentioned in [17] that it might take up to 5 ms to elect a new leader for a single partition, therefore the comparable unavailability time was set to $u = 5$ ms. The same source asserts that Apache Kafka production clusters can support more than 30,000 open file handles per broker. In our situation, we set $H_{\max} = 10.000$ in order to be able to record even weaker values. Following that, it can be inferred from the report's information that the replication latency for a single partition (without taking into account other partitions) may be as low as $l_r = 1$ ms. We set the unavailability time as $u = 5$ ms since, finally, the report claims that leader election for a single partition might take up to 5 ms. The parametrical inputs T, c, r, L, U and B are determined by the needs of the application. These parameters may have entirely different values in other applications. The parameter settings are displayed in Table 2.

In our numerical simulations, we measure several performance metrics that are relevant to evaluate the effectiveness of our proposed approach. The first important metric is throughput, which we measure indirectly with the number of partitions as an indicator, giving us an idea of the efficiency of the system in handling large volumes of data. The desired throughput T also feeds the algorithms as a system constraint. Another key metric that we use to evaluate the performance of our system is latency, which is the time delay between the production of a message by the producer and its consumption by the consumer. Lower latency values indicate that the system is able to process and deliver messages more quickly, which is especially important in real-time applications where timely delivery of messages is critical. The dominant factor in this end-to-end procedure (which included processing, storing, etc.) is the replication time of the data. We capture this metric by appropriately measuring l_r, but also through the system input constraint L. We also consider the number of brokers in the infrastructure as an important metric, since it can significantly impact the overall performance of the system. A larger number of brokers can improve the scalability of the system and increase its capacity to handle more data, but at the same time, it can also increase the communication overhead and infrastructure costs. The OS load, measured in terms of open file handles, is another important metric that we consider in our evaluation. Finally, we also measure the unavailability of the system, which refers to the amount of time that the system is not available to process messages. This can happen due to various reasons, such as network failures, hardware failures, or software errors. By measuring the unavailability of the system, we can evaluate its resilience and identify potential areas for improvement.

As discussed in Section 2.2, there is no established topic partitioning benchmark or related approach in the literature to compare our algorithms with. For this reason, to define a baseline we took into account a combination of industry best practices in the fog computing realm, from Microsoft and Confluent. Microsoft reports in [28] that it is a general rule of thumb not to exceed 1.000 partitions and replicas per broker. Additionally, Confluent, reports in [23,27] that the number of partitions P per broker b ideally can be concentrated around to $100 \cdot B \cdot r$. Consequently, after considering these industry best practices, we define the MS-CNFL (Microsoft/Confluent) benchmark method and we express it as follows:

$$P = \min\left(P \in_R \left[1 \ldots \frac{1.000 \cdot B}{r}\right], P \in_R [1 \ldots 100 \cdot B]\right), b \in_R [1 \ldots B]$$

with $\in_R$ defined as selection uniformly at random. Table 3 displays a descriptive comparison of the three methods discussed so far.

In Fig. 2 we illustrate the numerical simulation results on the performance evaluation for increasing numbers of consumers in the system. We can see that BroMax and MS-CNFL maintain a similar number of partitions regardless of the number of consumers, whereas BroMin linearly increases the number of partitions when the number of consumers increases. The same behaviour can be observed with the number of brokers used. It is apparent that MS-CNFL maintains a larger number of partitions compared to BroMin and BroMax throughout the simulations. However, this overuse of partitions comes at a cost, with MS-CNFL violating the latency constraint. The unavailability and OS load thresholds are respected by all the algorithms; MS-CNFL, however, underperforms.

In Fig. 3 we illustrate the numerical simulation results on the performance evaluation for increasing numbers of available brokers in the system. We can see that BroMax and MS-CNFL linearly increase the number of partitions when the number of available brokers increases, whereas BroMin maintains a similar number of partitions regardless of the number of available brokers in the system. The same behaviour can be observed with the number of brokers used. Like in the previous batch of results, MS-CNFL uses more brokers than BroMin and BroMax, but this leads to violating the latency constraint. MS-CNFL also exhibits the highest unavailability time and OS load, even though they remain below the respective thresholds.

In Fig. 4 we illustrate the numerical simulation results on the performance evaluation for increasing replication factor value in the system. We can see that BroMax and MS-CNFL decrease the number of partitions when the replication factor increases, whereas BroMin maintains a similar number of partitions regardless of the replication factor. Instead, BroMax and MS-CNFL assign a constant number of brokers, irrespective of the replication factor, while the number of brokers assigned by BroMin increases linearly with the replication factor up to the point when the curves BroMax and BroMin meet at $r = 20$. Due to the higher number of partitions, MS-CNFL exceeds the replication latency threshold even in this last batch of simulation results and is outperformed by BroMax/BroMin in terms of unavailability time and OS load once more.

The key messages of the numerical simulation analysis are as follows:

Table 3
Methods and benchmark used in the numerical simulation performance evaluation.

Name	Source	Description	Insight
BroMin	Current paper	Heuristic	Meet the objective/constraints via minimising broker usage
BroMax	Current paper	Heuristic	Meet the objective/constraints via maximising broker usage
MS-CNFL	Microsoft, Confluent [23,27]	Industry best practices	Rules of thumb regarding partition and replica numbers

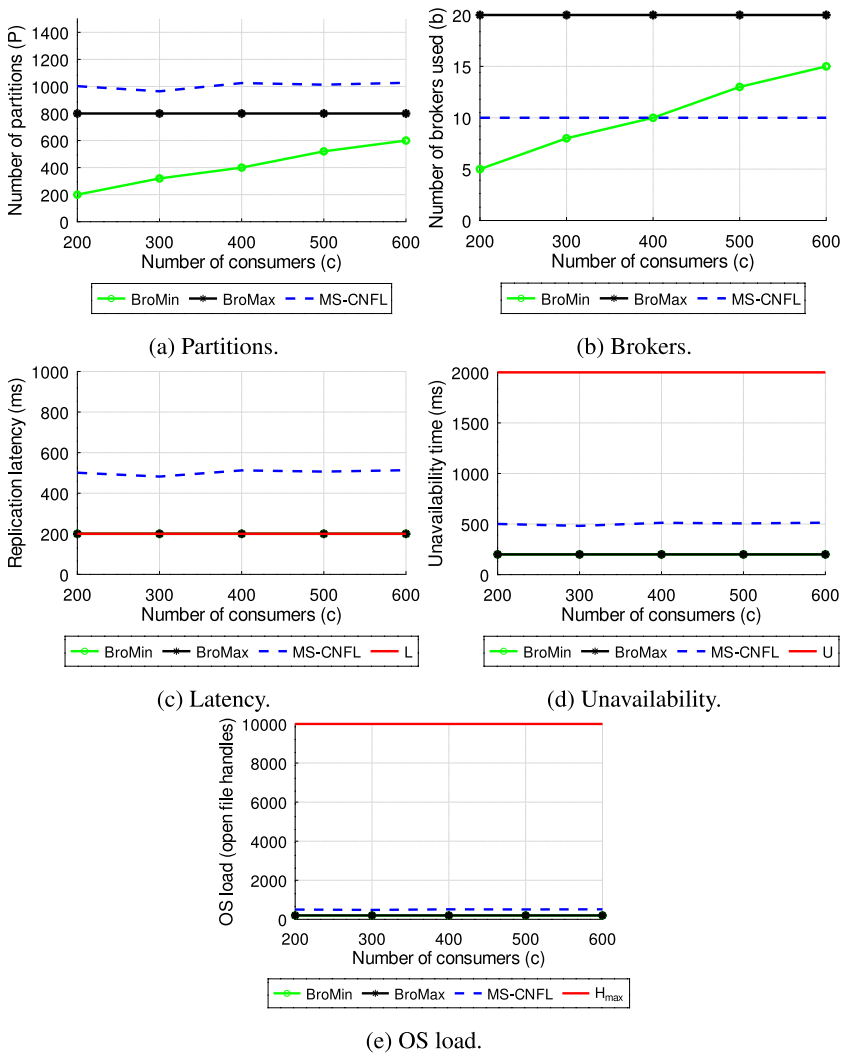

Fig. 2. Numerical simulations: Performance for variable numbers of consumers.

- BroMin and BroMax satisfy the application constraints, while the industrial best practices can violate the latency constraint.
- BroMin and BroMax are able to achieve efficient performance with a smaller number of partitions.
- This holds true even for the unavailability and OS load metrics.
- BroMin and BroMax distribute partitions across the brokers in a smarter manner, especially when r is increasing.

4. Prototype experiments

In this section, we complement the numerical simulation analysis in Section 3.4 with experiments obtained in a prototype system that includes Apache Kafka brokers, producers, and consumers, where we have conducted a series of experiments to measure its performance under various conditions. The experimental evaluation allows us to verify the applicability of our simulation results and provides us with a reliable method for evaluating the performance of our approach. The prototype experiments have been carried out in a realistic fog environment, where two servers are connected to a local network and communicate with each other. We have performed three experiment types (consumption, production, end-to-end) and changed various parameters, such as message sizes to investigate the behaviour of our two heuristics under wildly different target scenarios. We begin by illustrating the framework developed and the methodology used for the experiments.

4.1. Framework and methodology

Experiments have been carried out in controlled and repeatable conditions with a reusable framework that we made publicly available to the research community on GitHub[5]: the repository includes the detailed steps to prepare the prototype, the full set of well-documented

[5] https://github.com/ccicconetti/kafka-hdd tag `1.0.0`, experiments labelled from 001 to 003.

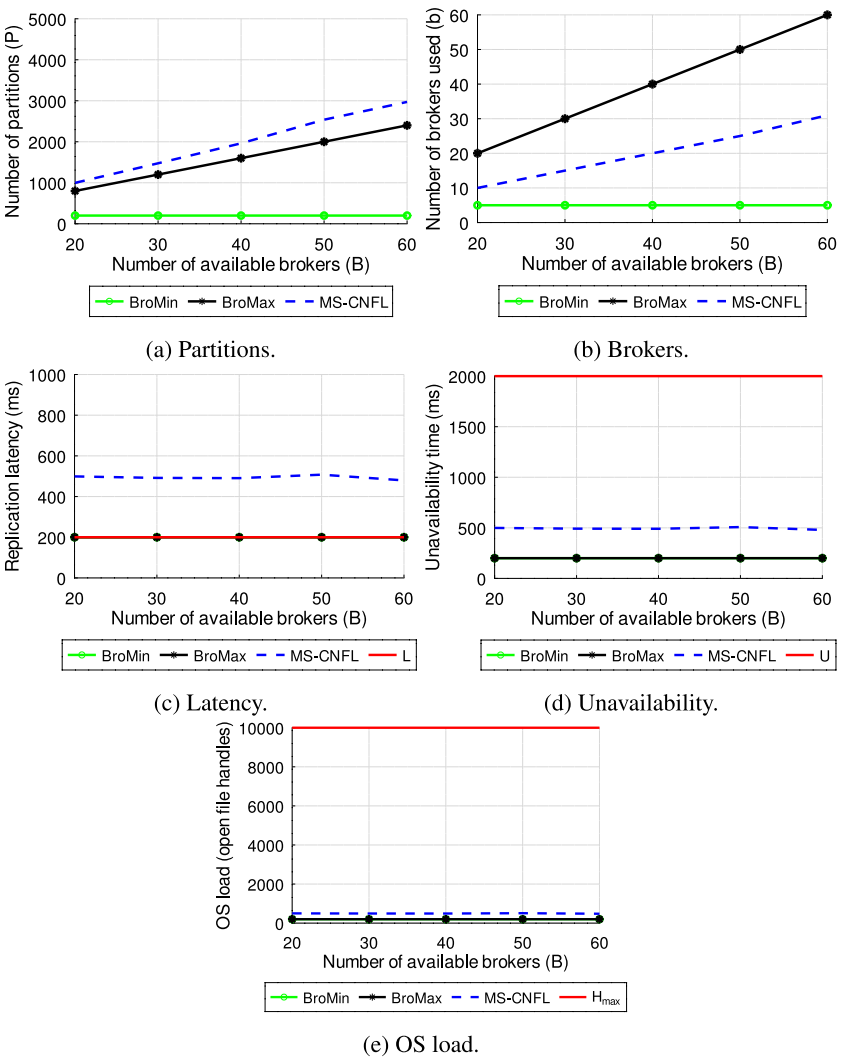

Fig. 3. Numerical simulations: Performance for variable numbers of available brokers.

scripts to execute the experiments, and the dataset of results obtained on our servers. Our framework enables three types of experiments, each stimulating different aspects of the system performance, which can be more relevant for a specific target production scenario:

1. *Consumption experiment*, which focuses on the consumption throughput T_c that can be obtained by a set of clients from an Apache Kafka cluster for messages of a given size M, where the data injection is not a choke point. For consumption, the rebalance time τ_R, i.e., the time needed for the assignment of the partitions to the consumers to converge to its final configuration, can be also relevant and, thus, it is produced as output by this type of experiment.
2. *Production experiment*, which is complementary to the previous one and focuses on the production throughput T_p that can be obtained, together with the production latency l_p, i.e., the time between when a given message is generated by a client and when it is committed by the cluster
3. *End-to-end experiment*, where there are both producers and consumers concurrently generating/reading data in a real-time streaming fashion, with the end-to-end latency l_{e2e} being the main performance metric.

Our prototype consists of two high-end servers: one hosting the *client-side* scripts and tools and another handling the *cluster-side*, i.e., the Apache Kafka brokers and a Zookeeper instance for leader election among them. The software on the cluster-side server runs within Docker containers configured with Docker Compose,[6] which is a tool to start/stop/manage applications consisting of multiple containers defined in a single YAML file. This approach is suitable for running the entire cluster within a single physical server, but the scripts we developed can be adapted to match the specific characteristics of the target deployment under test, e.g., a distributed environment where Apache Kafka is run within a Kubernetes (K8s) cluster. The methodology would remain the same and it is illustrated by means of the sequence diagram in Fig. 5, which is entirely managed through the execution of a single Bash script on the client-side server as detailed in the following.

At the beginning of the experiment, there is no Apache Kafka cluster running. In principle, there are situations where we could reuse a running cluster from the previous experiment, i.e., when the cluster parameters P, b remain the same but we decided to start with clean conditions to ensure independence and repeatability. The input of the experiment is: the Apache Kafka cluster provisioning algorithm A, the replication factor r, the number of consumers c, and the size of the Apache Kafka messages exchanged M, in bytes. The algorithms also require additional system parameters, as described in Section 3.4, which are considered static in the experiments described in this work.

[6] https://docs.docker.com/compose/

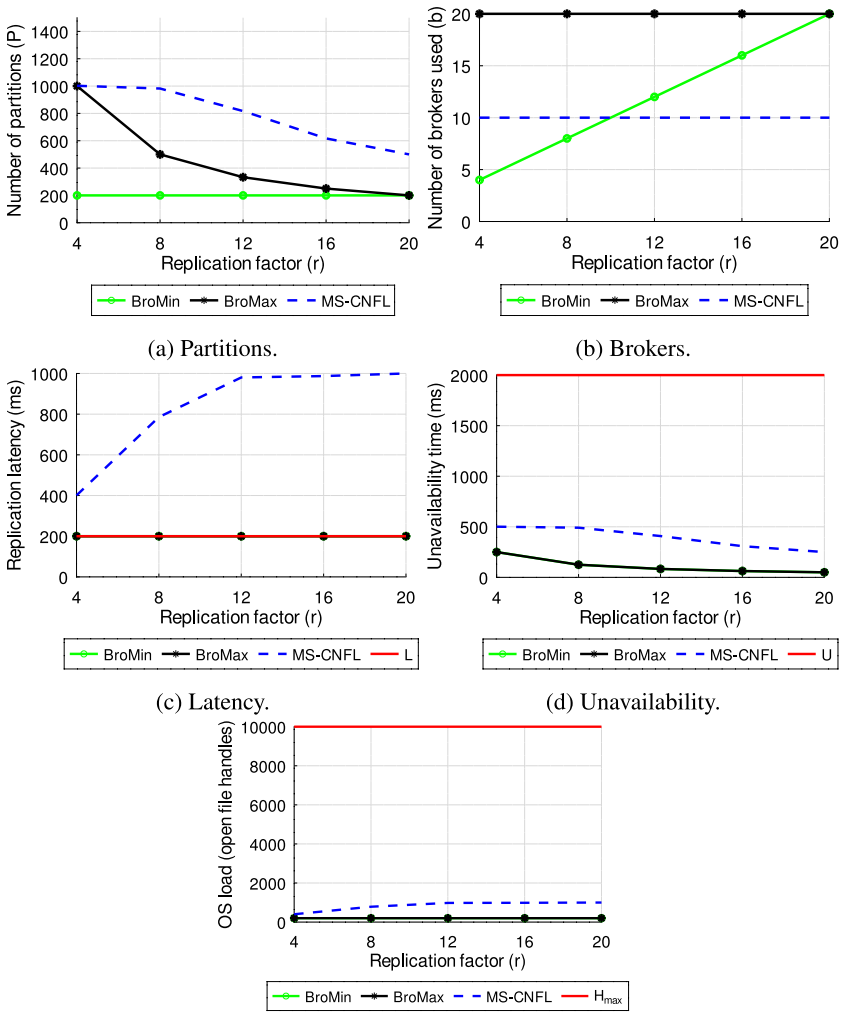

Fig. 4. Numerical simulations: Performance for variable replication factor values.

On the client side, the algorithm A, i.e., BroMin or BroMax, is run to determine the number of topic partitions P and brokers b. Then, the client issues the commands required to remotely start the Apache Kafka cluster of b brokers via Docker Compose, creates the topic that will be used in the current experiment, and configures it with the given replication factor r and the number of partitions P. Once done, the next steps depend on the experiment type:

1. *Consumption experiment*. The client starts a producer to fill the topic with a dataset of messages of size M. When the dataset is complete, the core part of the experiment starts. It is carried out by means of a pool of c consumers, all belonging to the same consumer group, started as independent processes with a small random initial delay. All the consumers greedily read messages from the dataset as fast as they can, without performing any processing on the payload or saving it, until the dataset is exhausted. This is because the experiment intends to measure an upper bound of the consumer throughput T_c, as well as the rebalance time. Both these metrics are produced by the script `consumer-perf-test.sh`, bundled with Apache Kafka, which is used under the hood by our framework.
2. *Production experiment*. The client starts a multi-threaded producer that fills the topic with a dataset of messages of size M as fast as possible and the experiment terminates after a preconfigured number of messages have been generated. If the client-side server is more powerful than the cluster-side server and the interconnecting network is not a performance bottleneck (these conditions are both verified in our testbed), then the message rate produced in output can be considered as an upper of the production throughput T_p. On the other hand, the production latency l_p provides an indication of the time required by the cluster to confirm that a given message has been replicated across all the required brokers as needed based on the replication factor requested. The tool used internally for this experiment type is `kafka-producer-perf-test.sh`.
3. *End-to-end experiment*. In this case, two pools of processes are spawned on the client-side server: producers and consumers. The producers inject messages of a random size drawn from a uniform distribution on the cluster with a configurable constant message rate: this emulates data streaming from a continuous but variable-rate application, like an IP video camera. Concurrently, the consumers read messages as fast as possible. Timestamps are recorded by both the producers and the consumers so that the time elapsed between when a message is created and when it is consumed, called end-to-end latency l_{e2e}, can be

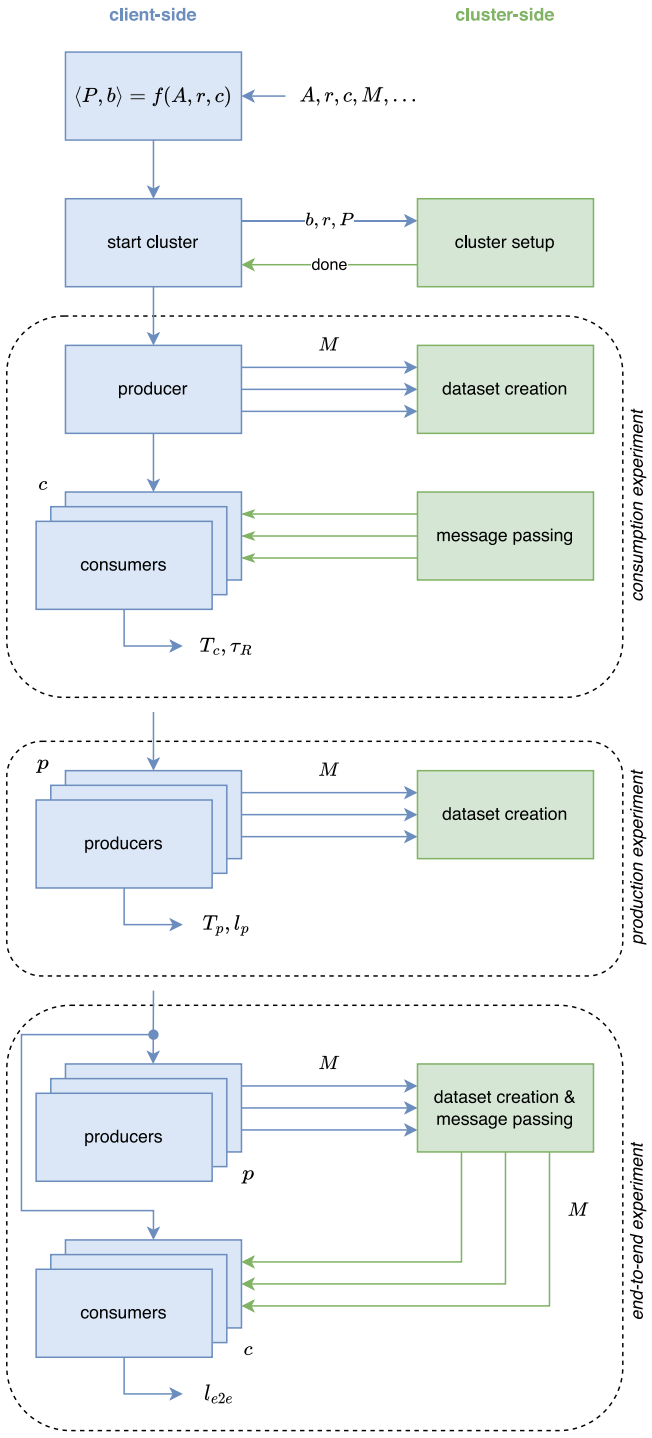

Fig. 5. Methodology adopted for the execution of automated experiments with an Apache Kafka cluster.

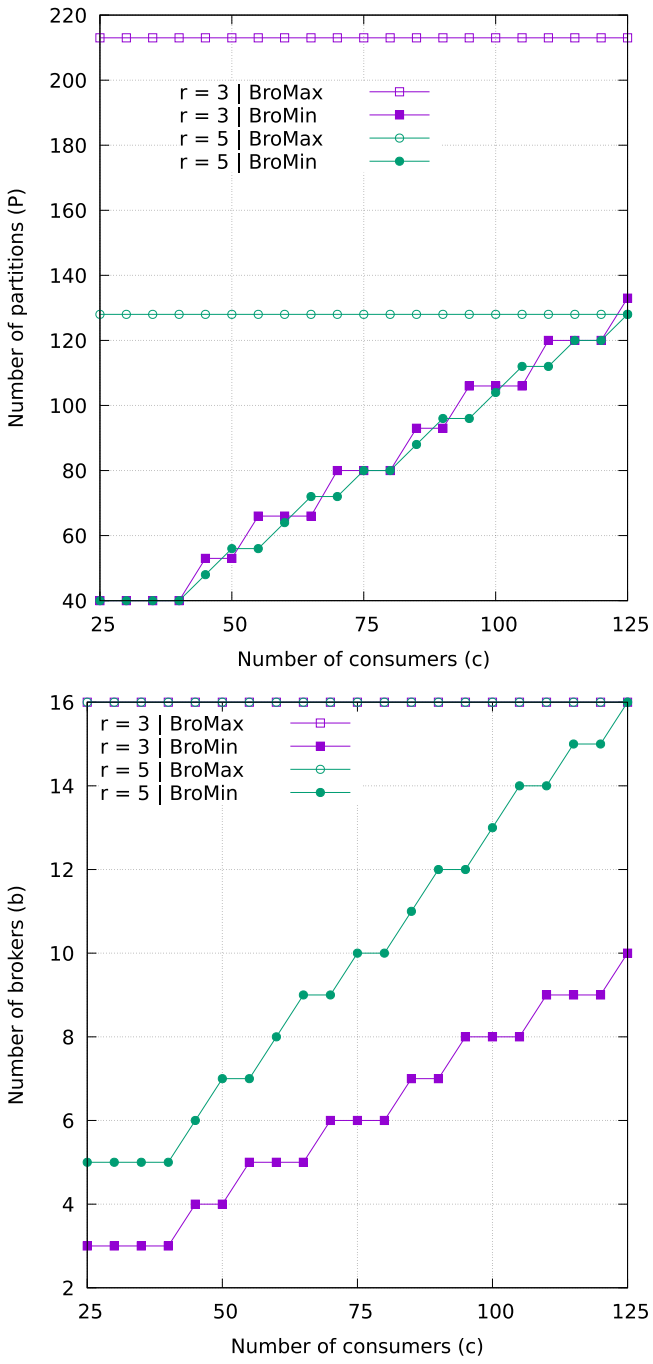

Fig. 6. Prototype experiments: Number of partitions (P, top) and brokers (B, bottom) selected by the BroMin/BroMax algorithms for replication factor $r = 3, 5$ and used in the prototype experiments.

retrieved. Customer Python scripts are used as producers and consumers and shipped as part of our framework.

In the experiments illustrated hereafter, we have set the maximum number of brokers available B to 16, based on the hardware characteristics of the cluster-side server[7] and we varied the number of consumers c in the range $[25, 125]$. In Fig. 6 we show the output of the BroMax and BroMin algorithms, i.e., P on the left-side plot and b on the right-side one, with two replication factors: $r = 3$, which is a typical value used in production systems, and $r = 5$, which is intended for the dispatch of critical messages in a system where brokers are prone to failures. These plots are akin to Figs. 2–4, previously discussed as part of the numerical simulation analysis, only for different ranges adjusted to fit the scope of the prototype experiments in this section. They are reported here to show the values of P and b with all the combinations of A, r, and c. The data collected during the warm-up and cool-down phases of each experiment have been removed to focus on steady-state measures. Finally, for each combination of factors we have run multiple iterations:

[7] A Supermicro server with four CPUs AMD Opteron(tm) Processor 6282 SE and 128 GB RAM.

in the plots, we show only the average value across them for better readability; the interested reader can find the plots with confidence intervals in the GitHub repository.

4.2. Consumption experiments

We begin with consumption experiments and we start focusing on messages of small size, that is $M = 1$ kB. In Fig. 7(a), we show the consumer throughput T_c. As can be seen, all the curves decrease with increasing number of consumers c. This is an expected behaviour with small message sizes and lightweight consumers because they can ingest messages very fast, thus a more aggressive parallelisation results in a negligible speed-up compared to a noticeable overhead. Furthermore, BroMax always achieves the lowest T_c, even though it always saturates the number of brokers, i.e., $b = B = 16$ in the experiments, and uses more partitions for the topic, as shown in Fig. 6 above.[8] Finally, the impact of the replication factor $r = 3$ vs. $r = 5$ is small with BroMax and negligible with BroMin. Again, this is because of the small size of messages, which keeps the additional overhead due to an increased replication low, especially with fewer brokers/partitions, i.e., in the BroMin case.

In Fig. 7(b) we show the rebalance time τ_R. Based on the results, the replication factor does not affect significantly this metric. On the other hand, BroMax requires a significantly longer time for the consumer group to converge to a stable distribution of partitions than BroMin with a small number of consumers, i.e., as $c < 85$. This can be explained by the higher number of brokers. However, as the number of consumers becomes very large, i.e., in the right part of the plot, the τ_R curves become almost overlapping, due to the counterbalancing effect of the number of consumers. Remember that in our experiments the consumers join the group with small random offsets and every new consumer entering the group will trigger a rebalancing procedure: more consumers result in a higher rate of rebalancing events triggered.

In Fig. 8(a) we report a representative snapshot of the CPU load measured on the physical servers, which were unloaded except for the Apache Kafka prototype experiments. The valleys correspond to when the experiment iterations terminate, after which the load increases up to peaks that mark the end of the respective experiments. In addition to following the same time pattern, incidentally, the CPU load of the client and server also have similar values. We included this plot to show that neither of the two servers is overloaded (the server has 64 real CPU cores), which otherwise would have affected the results.

For the same reason, we report in Fig. 8(b) the disk I/O on the cluster server. This metric is irrelevant on the client side because the clients do not use the disk at all as they drop immediately the payload of messages after ingesting them. Also for this metric, we can see that the server cannot be considered to be overloaded, and in fact, Apache Kafka is known to be quite efficient in this respect despite it relying heavily on data persistence on disk.

We now move to the case of large message size, i.e., $M = 100$ kB. First we show T_c in Fig. 9(a). Unlike with small message size, here the performance is dominated by the replication factor: $r = 3$ achieves a much higher throughput (about 10×) than $r = 5$. This suggests that a large value of r should be configured only when strictly necessary, for instance, if the Apache Kafka brokers are highly unreliable; otherwise, the performance can be unnecessarily degraded. The difference between BroMax and BroMin is less noticeable than with small messages, with BroMin achieving a higher throughput in the order of 10% for

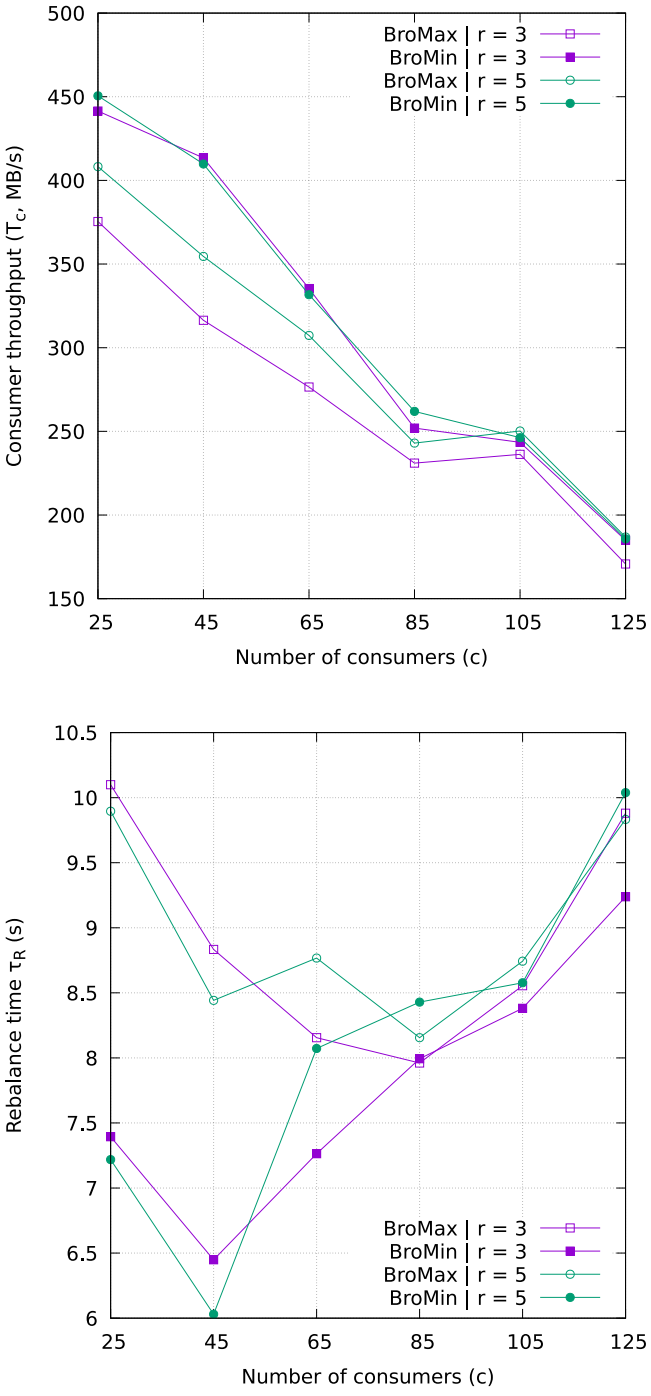

Fig. 7. Prototype consumption experiments: (a) Consumer throughput, and (b) rebalance time, with message size $M = 1$ kB.

$c = 25$ and $c = 45$ for both $r = 3$ and $r = 5$ (the latter cannot be seen from the graph for scale reasons); for both replications factors, the difference becomes negligible with more concurrent consumers.

We conclude the analysis with τ_R, shown in Fig. 9(b). As expected, the rebalance time is always smaller with $r = 3$, even though not drastically so. With $r = 3$, τ_R is comparable to that with a small message size previously shown in Fig. 7(b), with the BroMin curve laying below the BroMax curve, again, only for $c = 25$ and $c = 45$. With $r = 5$, τ_R shows a more erratic behaviour, which however remains in a 10 s range around 20 s.

[8] There are some low-level parameters in Apache Kafka that could be adjusted to fine-tune the communication performance, such as the size of the buffers used by TCP sockets. However, to the best of our knowledge, there are no widely-used guidelines that can be adopted to optimise performance metrics according to a specific scenario, thus we have left the default values and did not consider them as part of our analysis.

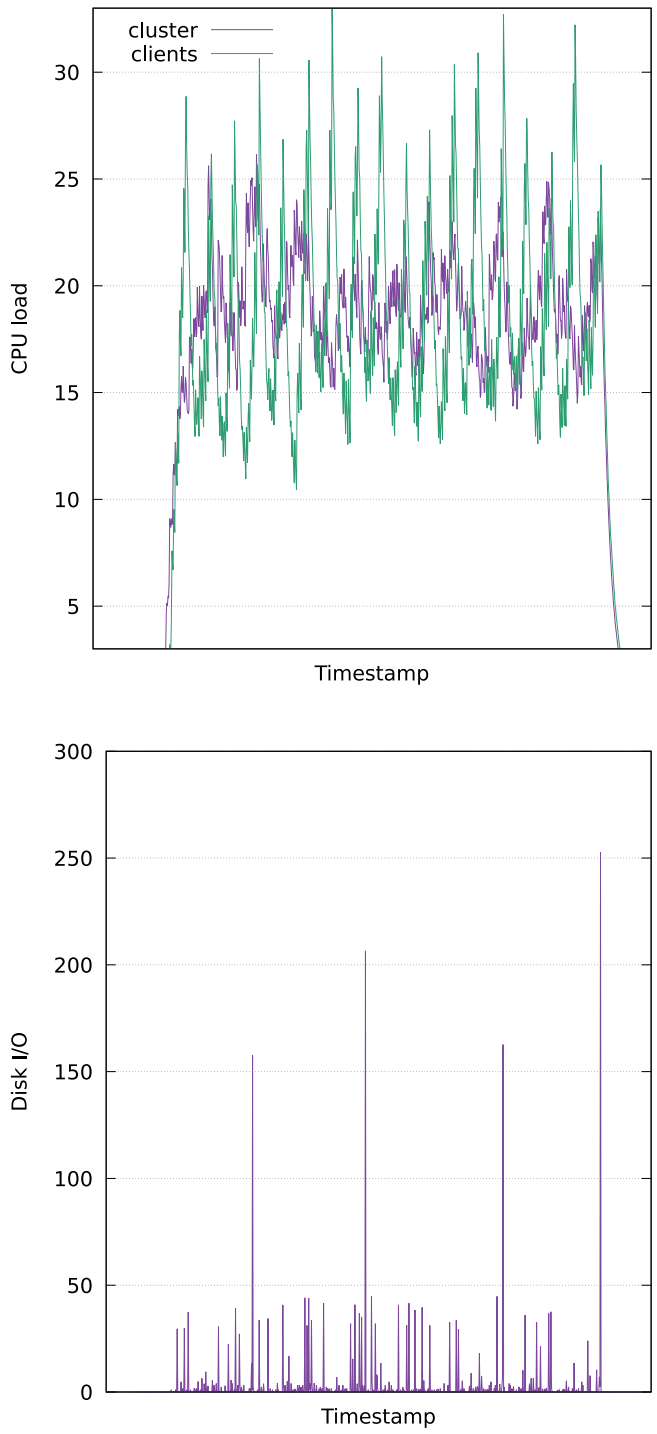

Fig. 8. Prototype consumption experiments: (a) CPU load (client vs. cluster), and (b) disk I/O (cluster only), with message size $M = 1$ kB.

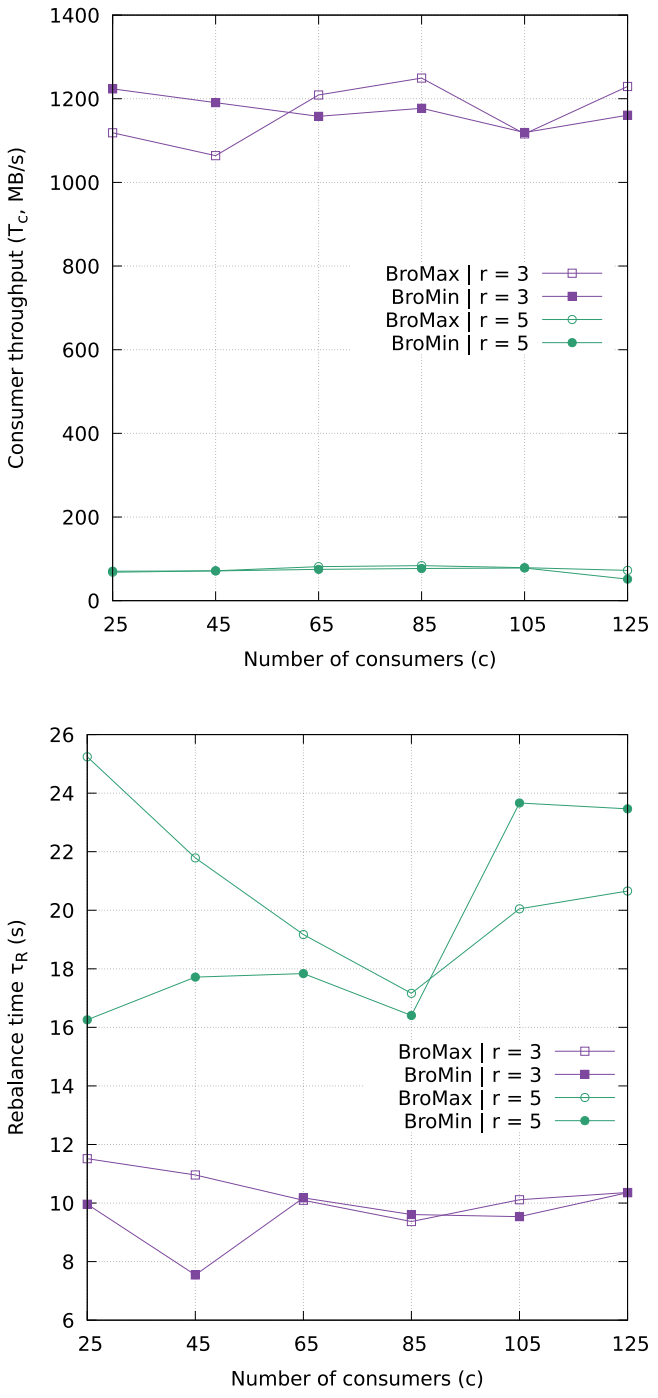

Fig. 9. Prototype consumption experiments: (a) Consumer throughput, and (b) rebalance time, with message size $M = 100$ kB.

4.3. Production experiments

We now move to a batch of production experiments, obtained by increasing the message size from 1 kB to 100 kB. Recall from Section 4.1 that, in this type of experiment, the client-side server uses all possible resources to produce data as fast as possible in an attempt to estimate the maximum production throughput of the cluster, therefore the number of producers is not relevant. By looking at the two plots in Fig. 10, which show the production throughput T_p measured in MB/s vs. record rate, we can see that the qualitative trends are opposite: this is because larger messages consume more resources than small ones to be transferred/serialised/replicated, thus the record rate decreases as M increases, but the throughput in MB/s is more than compensated by decreasing per-message overhead and it increases significantly overall.

To better understand the results for what concerns the other parameters, i.e., the replication factor and the resource allocation policy, we report in Table 4 the number of partitions P and brokers b selected by the two heuristic algorithms BroMax and BroMin. As can be seen from Fig. 10, the production throughput with BroMax is much higher than that with BroMin for all the values of M: this is because the former allocates way more resources than the latter, in terms of brokers, i.e., 16

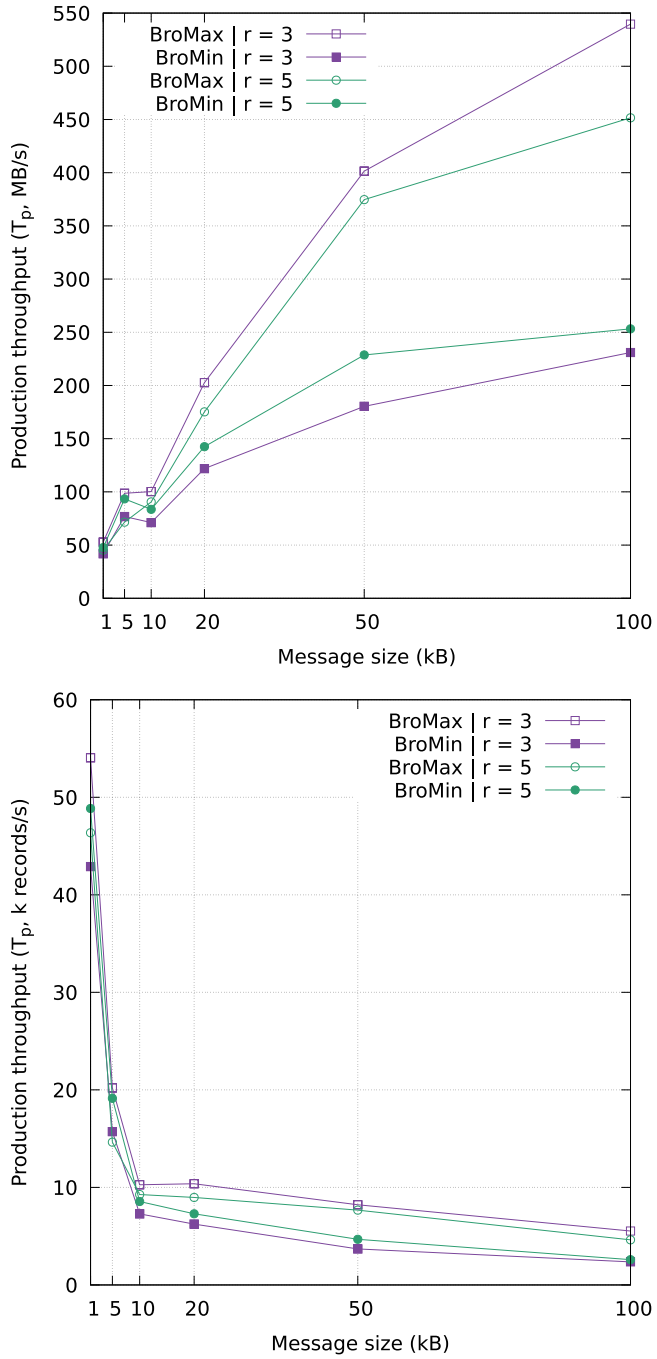

Fig. 10. Prototype production experiments: Production throughput in MB/s (top) and record rate (bottom).

instead of 3 or 5. As a result, the Apache Kafka cluster can increase the level of parallelism when ingesting data, which leads to a higher rate of produced messages. T_p is slightly smaller with a higher replication factor because the amount of resources remains the same but each message must be replicated 5 times instead of only 3 in the cluster. On the other hand, with BroMin the opposite happens: the throughput with $r = 5$ is higher than that with $r = 3$ because, while BroMin is conservative in the use of resources, it cannot allocate a number of brokers that is less than the replication factor, which leads to improved performance thanks to the extra resources available with $r = 5$.

We conclude this scenario with the production latency, whose average value is shown in Fig. 11. Counter-intuitively, the average latency

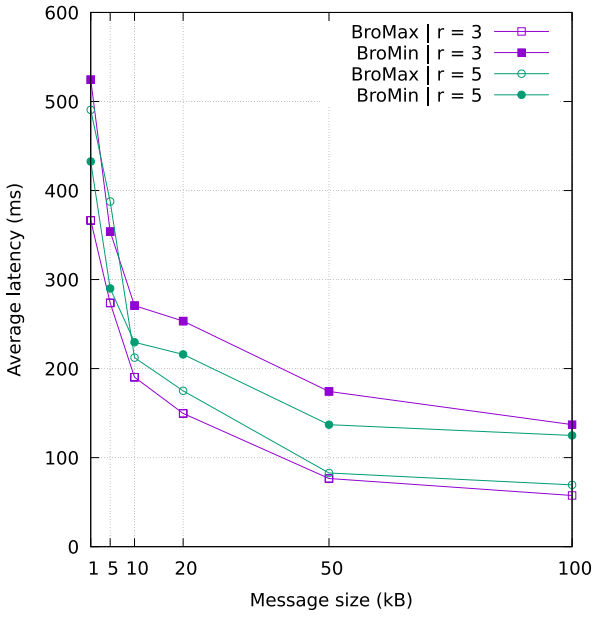

Fig. 11. Prototype production experiments: Average production latency.

Table 4
P, b selected by BroMax vs. BroMin in the production experiments.

	BroMax	BroMin
$r = 3$	$P = 213, b = 16$	$P = 40, b = 3$
$r = 5$	$P = 128, b = 16$	$P = 40, b = 5$

decreases as the message size increases, even if the mere transmission time from client to server obviously increases with M. This is because the production latency takes into account also the time for the cluster to replicate the data and such an operation is done in batches: with larger messages, the brokers need to pack together a smaller amount of messages to perform some of their housekeeping operations, which eventually leads to lower latency. On the other hand, the BroMax curves are lower than the BroMin ones, since the latency is inversely proportional to the throughput. Finally, with BroMax, the difference in the average latency for the two replication factors is negligible, while it is noticeable with BroMin, for which $r = 5$ enjoys a relatively smaller average latency, because of the availability of more brokers, as discussed above.

4.4. End-to-end experiments

We conclude the prototype performance evaluation with an end-to-end experiment carried out with p producers generating 30 messages/s with variable message size in $U[1\text{ kB}, 100\text{ kB}]$. Initially, we keep the number of consumers equal to 5 to best emulate a real production scenario where there is a fixed number of services that consume data, e.g., for analytics or monitoring purposes, while the number of producers may change over time due to, e.g., organic growth of the infrastructure. We consider only a replication factor of 3, which is adequate for most production systems deployed with nodes having industry-standard mean time before failure. We report the Cumulative Distribution Function (CDF) of the latency, with 10, 15, and 20 producers, in Fig. 12. First, we observe that the latency increases significantly with the number of producers: this is expected because each producer generates data independently from the others, so increasing p corresponds to increasing the overall cluster load proportionally, which yields higher latencies. This suggests that resource allocation should also take the number of producers as an input variable of the problem;

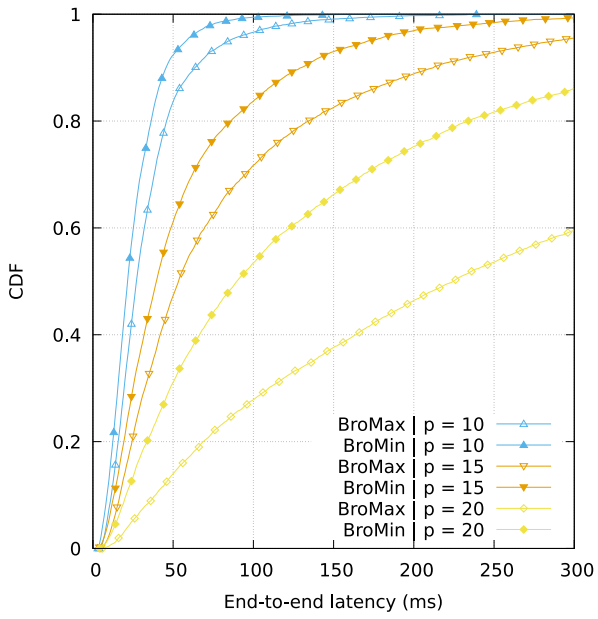

Fig. 12. Prototype end-to-end experiments: Cumulative distribution of the end-to-end latency with different resource allocation policies (BroMax vs. BroMin) and number of producers ($p = 10, 15, 20$), with 5 consumers ($c = 5$).

when a value for this parameter cannot be estimated, the latter can be monitored in the field and used to periodically re-run an appropriate resource allocation algorithm; such a study is complementary to this work and will be considered as part of our future research activities. Second, BroMax exhibits a higher latency than BroMin; in other words, unlike in the production experiment above, allocating *more* resources to the cluster, that is brokers and partitions, leads to *worse* performance, especially in terms of the high tail latency. This effect is due to the increased overhead for cluster management and it implies that, contrary to general intuition, overprovisioning does not necessarily produce optimal results, when the key performance is maintaining a tight pipeline from producers to consumers.

We conclude with an extreme scenario, where we increase further the number of producers, to 25 and 50, and the number of consumers, to 10 and 15. As shown in Fig. 13, the latency increases significantly, due to the high total throughput (310 Mb/s for $p = 25$ and 620 Mb/s for $p = 50$) compared to the resources available in the cluster. Unlike in previous consumer-/producer-only experiments, here the number of consumers affects significantly the latency, which increases steeply from $c = 10$ to $c = 15$. Furthermore, we can see that BroMax generally exhibits better performance than BroMin, especially in terms of the high quantiles of latency. This is because the performance bottleneck in this scenario is created by how fast the messages can be dispatched by the brokers, hence having more brokers helps in reducing the congestion.

4.5. Take-away messages

The key messages of the analysis of prototype experiments are as follows:

- The performance in a real Apache Kafka cluster depends significantly on the size of the messages exchanged, as well as on the replication factor, which can significantly degrade the consumption throughput with large messages.
- With small messages, the advantages of parallelising data ingestion with multiple consumers can be offset by the increased communication overhead, thus leading to a reduced cluster throughput; on the other hand, larger size messages incur smaller cluster management overhead and more efficient batching, which lead to improved production throughput and latency.
- In a real Apache Kafka cluster, a more aggressive allocation of resources (BroMax), in terms of the number of brokers and topic partitions, does not always lead to improved performance: it does so for production throughput but, on the contrary, the consumption throughput can be lower and it can take longer for the rebalancing procedure to converge compared a more conservative use of resources (BroMin).
- In a real-time streaming scenario, BroMin achieves significantly lower end-to-end latency due to the reduced complexity of the cluster management procedures, but BroMax reduces the congestion in extremely loaded conditions.

5. Conclusions and future work

In this paper, we have formulated the Apache Kafka topic partitioning process as an optimisation problem to determine the optimal number of partitions to satisfy the requirements and constraints of high-reliability real-time data streaming applications. We have proposed two efficient heuristics that strive to achieve a balanced allocation of resources while avoiding under- or over-utilisation of the system resources, which have been evaluated in a broad set of configurations through numerical simulations. Furthermore, we have devised a methodology for the fully-automated performance evaluation of an Apache Kafka cluster in three types of experiments: consumption, production, and end-to-end. We have showcased the framework with a testbed prototype and compared BroMax and BroMin in representative scenarios with different message sizes and replication factors. The results have highlighted some qualitative differences with respect to the conclusions obtained through simulations: this suggests the need for cross-checking analysis with real-life data before production deployment. Our evaluation framework, which was made available to the community, can be used precisely for this purpose. Future directions on the topic include but are not limited to (i) modelling and addressing the more generalised problem of multi-topic partitioning, (ii) employing multi-objective optimisation techniques to simultaneously address potential concurrent application requirements, and, (iii) designing data-driven ML methodologies, with data from actual Apache Kafka deployments.

CRediT authorship contribution statement

Theofanis P. Raptis: Conceptualization, Methodology, Software, Visualization, Writing – original draft. **Claudio Cicconetti:** Data curation, Visualization, Writing – original draft, Software, Validation. **Andrea Passarella:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data reported is available from the GitHub repository cited in the paper.

Acknowledgements

The work of C. Cicconetti and A. Passarella was partially funded by the European Union's Horizon 2020 research and innovation programme MARVEL under grant agreement No 957337. The work of T. P. Raptis was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on *Telecommunications of the Future* (PE00000001, program RESTART). 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.

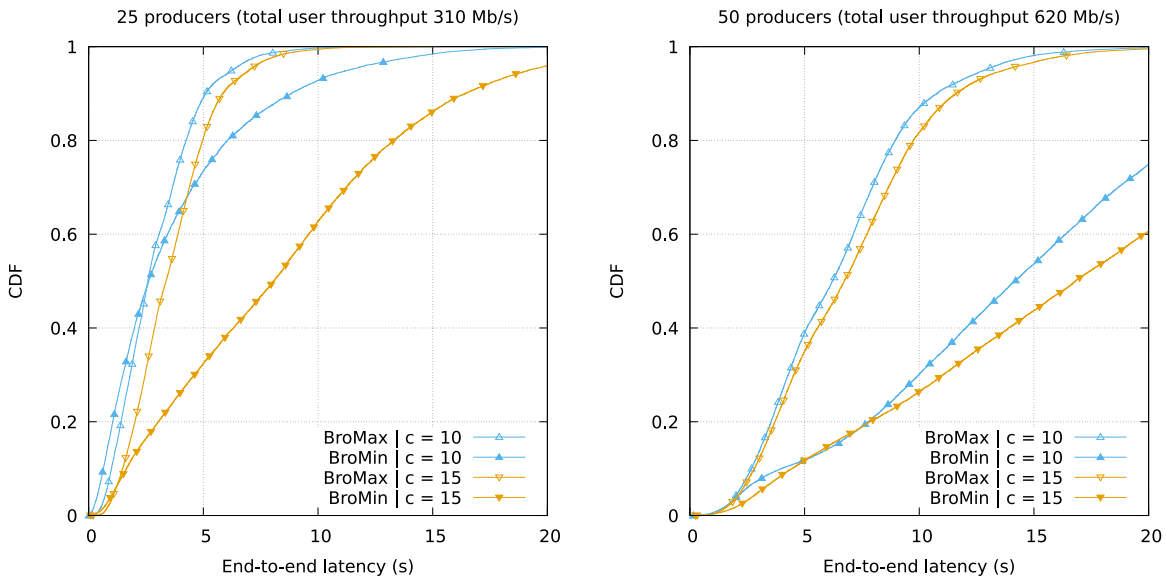

Fig. 13. Prototype end-to-end experiments: Cumulative distribution of the end-to-end latency with different resource allocation policies (BroMax vs. BroMin), number of consumers ($c = 10, 15$), and number of producers ($p = 25, 50$).

References

[1] K. Fizza, N. Auluck, A. Azim, Improving the schedulability of real-time tasks using fog computing, IEEE Trans. Serv. Comput. 15 (1) (2022) 372–385, http://dx.doi.org/10.1109/TSC.2019.2944360.

[2] R. Beraldi, C. Canali, R. Lancellotti, G.P. Mattia, Distributed load balancing for heterogeneous fog computing infrastructures in smart cities, Pervasive Mob. Comput. 67 (2020) 101221, http://dx.doi.org/10.1016/j.pmcj.2020.101221.

[3] J.-H. Moon, Y.-T. Shine, A study of distributed SDN controller based on Apache Kafka, in: 2020 IEEE International Conference on Big Data and Smart Computing, BigComp, 2020, pp. 44–47, http://dx.doi.org/10.1109/BigComp48618.2020.0-101.

[4] B. Costa, J. Bachiega, L.R. de Carvalho, A.P.F. Araujo, Orchestration in fog computing: A comprehensive survey, ACM Comput. Surv. 55 (2) (2022) http://dx.doi.org/10.1145/3486221.

[5] Y. Zhou, L. Tian, L. Liu, Y. Qi, Fog computing enabled future mobile communication networks: A convergence of communication and computing, IEEE Commun. Mag. 57 (5) (2019) 20–27, http://dx.doi.org/10.1109/MCOM.2019.1800235.

[6] T.P. Raptis, A. Passarella, A survey on networked data streaming with Apache Kafka, IEEE Access 11 (2023) 85333–85350, http://dx.doi.org/10.1109/ACCESS.2023.3303810.

[7] S. Mirampalli, R. Wankar, S.N. Srirama, Evaluating NiFi and MQTT based serverless data pipelines in fog computing environments, Future Gener. Comput. Syst. (2023) http://dx.doi.org/10.1016/j.future.2023.09.014.

[8] S. Park, J.-H. Huh, A study on big data collecting and utilizing smart factory based grid networking big data using Apache Kafka, IEEE Access (2023) 1, http://dx.doi.org/10.1109/ACCESS.2023.3305586.

[9] B. Leang, S. Ean, G.-A. Ryu, K.-H. Yoo, Improvement of Kafka streaming using partition and multi-threading in big data environment, Sensors 19 (1) (2019) http://dx.doi.org/10.3390/s19010134.

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

[11] T.P. Raptis, C. Cicconetti, M. Falelakis, G. Kalogiannis, T. Kanellos, T.P. Lobo, Engineering resource-efficient data management for smart cities with Apache Kafka, Future Internet 15 (2) (2023) 43, http://dx.doi.org/10.3390/fi15020043.

[12] G. Wang, L. Chen, A. Dikshit, J. Gustafson, B. Chen, M.J. Sax, J. Roesler, S. Blee-Goldman, B. Cadonna, A. Mehta, V. Madan, J. Rao, Consistency and completeness: Rethinking distributed stream processing in Apache Kafka, in: Proceedings of the 2021 International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, 2021, pp. 2602–2613, http://dx.doi.org/10.1145/3448016.3457556.

[13] K. Daugėla, E. Vaičiukynas, Real-time anomaly detection for distributed systems logs using Apache Kafka and H2O.ai, in: A. Lopata, D. Gudonienė, R. Butkienė (Eds.), Information and Software Technologies, Springer International Publishing, Cham, 2022, pp. 33–42, http://dx.doi.org/10.1007/978-3-031-16302-9_3.

[14] Y. Huang, C. Li, M. Chen, Z. Su, ACT-SAGAN: Automatic configuration tuning for Kafka with self-attention generative adversarial networks, in: Proceedings of the 5th International Conference on Computer Science and Software Engineering, CSSE '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 180–184, http://dx.doi.org/10.1145/3569966.3570024.

[15] T.P. Raptis, A. Passarella, On efficiently partitioning a topic in Apache Kafka, in: 2022 International Conference on Computer, Information and Telecommunication Systems, CITS, 2022, pp. 1–8, http://dx.doi.org/10.1109/CITS55221.2022.9832981.

[16] J. Xu, J. Yin, H. Zhu, L. Xiao, Modeling and verifying producer-consumer communication in Kafka using CSP, in: 7th Conference on the Engineering of Computer Based Systems, in: ECBS 2021, Association for Computing Machinery, New York, NY, USA, 2021, http://dx.doi.org/10.1145/3459960.3459961.

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

[18] H. Wu, Z. Shang, 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, http://dx.doi.org/10.1109/HPCC/SmartCity/DSS.2019.00036.

[19] D. Landau, X. Andrade, J.G. Barbosa, Kafka consumer group autoscaler, 2022, http://dx.doi.org/10.48550/ARXIV.2206.11170.

[20] L. Xu, X. Ma, L. Xu, A novel adaptive tuning mechanism for Kafka-based ordering service, in: W. Ni, X. Wang, W. Song, Y. Li (Eds.), Web Information Systems and Applications, Springer International Publishing, Cham, 2019, pp. 119–125, http://dx.doi.org/10.1007/978-3-030-30952-7_14.

[21] J. Bang, S. Son, H. Kim, Y.-S. Moon, M.-J. Choi, Design and implementation of a load shedding engine for solving starvation problems in Apache Kafka, in: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–4, http://dx.doi.org/10.1109/NOMS.2018.8406306.

[22] M. Gütlein., A. Djanatliev., Modeling and simulation as a service using Apache Kafka, in: Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH, SciTePress, INSTICC, 2020, pp. 171–180, http://dx.doi.org/10.5220/0009780501710180.

[23] A. Povzner, S. Hendricks, 99Th percentile latency at scale with Apache Kafka, 2020, [Posted 25-February-2020, Accessed 06-April-2022], https://www.confluent.io/blog/configure-kafka-to-minimize-latency/.

[24] N. Salinger, Optimizing kafka performance, 2022, (Accessed 06 April 2022), https://granulate.io/blog/optimizing-kafka-performance/.

[25] I. Pelle, B. Sz˝ oke, A. Fayad, T. Cinkler, L. Toka, A comprehensive performance analysis of stream processing with Kafka in cloud native deployments for IoT use-cases, in: NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, 2023, pp. 1–6, http://dx.doi.org/10.1109/NOMS56928.2023.10154377.

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

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

[28] Performance optimization for Apache Kafka HDInsight clusters, 2021, [Posted 13-December-2021, Accessed 06-April-2022], https://learn.microsoft.com/en-us/azure/hdinsight/kafka/apache-kafka-performance-tuning.

Theofanis P. Raptis received the Ph.D. degree from the University of Patras, Greece. He was an Associate Researcher with the Computer Technology Institute and Press Diophantus, Greece. He is currently a Senior Researcher with the National Research Council, Italy. He has published in journals, conference proceedings, and books, more than 80 articles on industrial networks, wirelessly powered networks, Internet of Things testbeds, and platforms. He is also regularly involved in international IEEE and ACM sponsored conference and workshop organisation committees, in the areas of networks, computing, and communications. He has been serving as an Associate Editor for the IEEE Access and IET Networks journals and a Guest Editor for Computer Communications, Computer Networks and Pervasive and Mobile Computing journals.

Claudio Cicconetti received the Ph.D. degree in Information Engineering in 2007 from the University of Pisa, Italy, in 2003. He has been working in Intecs S.p.a. (Italy) from 2009 to 2013 as an R&D Manager and in MBI s.r.l. (Italy) from 2014 to 2018 as a Principal software engineer. He is currently a Senior Researcher with the Ubiquitous Internet group of IIT-CNR, Italy. He coauthored 70+ papers published in international journals and peer-reviewed conference proceedings and two international patents. He has been involved in many funded R&D initiatives and is the technical manager of the Horizon Europe project EDGELESS (Cognitive Edge-Cloud with Serverless Computing).

Andrea Passarella received the Ph.D. degree, in 2005. He is a Research Director at the Institute for Informatics and Telematics (IIT), National Research Council of Italy (CNR). Prior to joining IIT, he was with the Computer Laboratory, University of Cambridge, U.K. He is the coauthor of the book Online Social Networks: Human Cognitive Constraints in Facebook and Twitter Personal Graphs (Elsevier, 2015). He has published more than 200 papers on online and mobile social networks, decentralised AI, next generation internet, opportunistic, and ad-hoc and sensor networks, receiving the Best Paper Award at IFIP Networking 2011 and IEEE WoWMoM 2013. He has served as the General Chair for IEEE PerCom 2022. He is the Founding Associate Editor-in-Chief of Online Social Networks (Elsevier). He was the Guest Co-Editor of several special sections in ACM and Elsevier journals. He is the PI of the EU CHIST-ERA SAI (Social Explainable AI) project.