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

Performance Prediction for the Apache Kafka Messaging System

Han Wu, Zhihao Shang, Katinka Wolter
Institut für Informatik
Freie Universität Berlin
Berlin, Germany
Email: {han.wu, zhihao.shang, katinka.wolter}@fu-berlin.de

***Abstract*—Apache Kafka is a highly scalable distributed messaging system that provides high throughput with low latency. Various kinds of cloud vendors provide Kafka as a service for users who need a messaging system. Given a certain hardware environment, how to set the configurations of Kafka properly will be the first concern of users. In this paper, we analyze the structure and workflow of Kafka and propose a queueing based packet flow model to predict performance metrics of Kafka cloud services. The input configuration parameters of this model contain the number of brokers in Kafka cluster, the number of partitions in a topic and the batch size of messages. Through this model users can obtain the impact of certain configuration parameters on the performance metrics including the producer throughput, the relative payload and overhead and the change of disk storage usage over time. We use queueing theory to evaluate the end-to-end latency of packets. In the experimental validation we see a strong correlation between packet sizes and packet send interval, and the service time of packets fits a phase-type distribution. The correlation and fitting results are substituted to the essential constants in the model. Experiments are performed with various configurations for observing their effects on performance metrics. The results show that our model achieves high accuracy in predicting throughput and latency.**

***Index Terms*—Messaging system, Apache Kafka, Performance evaluation, Queueing theory**

I. Introduction

Apache Kafka has been developed originally at LinkedIn to process real-time log data with delays of no more than a few seconds, and it is designed to be a distributed, scalable, durable and fault-tolerant messaging system with high throughput [1], [2]. LinkedIn relies heavily on the scalability and reliability of Kafka for use cases like monitoring system metrics, traditional messaging or website activity tracking [3]. Kafka can handle more than 1.4 trillion messages per day across over 1400 brokers in LinkedIn [4]. Currently, due to its strong durability and high throughput with low latency, Apache Kafka has been widely used in today's Internet companies [5]. Twitter uses Kafka as a part of their Storm stream processing infrastructure, and Netflix applies Kafka in the Keystone pipeline for real-time monitoring and event processing.

Furthermore, many cloud vendors provide Kafka as a service for users who want to build and run applications that use Apache Kafka to process data. Kafka as a service is a cloud computing service model similar to Infrastructures as a Service (IaaS), where basic computing resources like CPUs and storage are provisioned over the Internet. Users generally pay according to the length of use, while the cloud vendor manages and maintains the Kafka cluster. The advantage is that users can easily scale up or down resources to meet their requirements, without purchasing, installing and networking Kafka cluster servers by themselves. Currently there are services like Amazon Managed Streaming for Kafka (MSK) provided by Amazon Web Services (AWS) [6], Event Streams provided by IBM [7], Apache Kafka for HDInsight provided by Microsoft Azure [8], and Confluent Cloud provided by Confluent Cloud Enterprise, which is also built by the creators of Kafka [9]. Those vendors provide limited resources, such as CPUs, memory and disks, and in the given hardware environment, users can set multiple Kafka configuration parameters on their own, such as batch size, partition number, and log retention time. How to configure those parameters to gain the best performance for a specific application is unclear without numerous experiments, which are time-consuming and costly. Besides, it is very inefficient to tune those parameters after a Kafka cluster is running.

In this paper, we propose a queueing based model to predict the performance of Kafka. Users can tune the configuration parameters in the model to observe the parameters' impact on Kafka's performance. This can help users to utilize the limited resources provided by cloud vendors more economically and give explicit advice on the configuration settings of Kafka.

II. Related work

Plentiful experiments have been done to compare the performance of Apache Kafka and traditional message brokers like RabbitMQ, which is primarily known and used as an efficient and scalable implementation of the Advanced Message Queuing Protocol (AMQP). The results indicate that both systems are capable of processing messages with low-latency, and increasing the Kafka partition number can significantly improve its throughput, while increasing the producer/channel count in RabbitMQ can only improve its performance moderately [10]. The Kafka protocol is more suitable for an application that needs to process massive messages, but if messages are important and security is a primary concern, AMQP is more reliable [11]. Despite those horizontal comparisons, how to set the configuration parameters properly inside Apache Kafka, which is also the major concern of the users who purchase Kafka cloud services, still remains challenging.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00036

Researchers have developed various tools to help the users of cloud services make appropriate decisions in pursuance of better performance with limited resources. CloudSim [12] is a well-known simulation framework which supports simulation tests across three major cloud service models (e.g. SaaS, PaaS, and IaaS) and it is capable of cloud simulations including VM allocation and provisioning, energy consumption, network management and federated clouds. CloudCmp [13] systematically compares the performance and cost of four different cloud vendors to guide users in selecting the best-performing provider for their applications. PICS (public IaaS cloud simulator) [14] provides the capabilities for accurately evaluating the cloud cost, resource usage and job deadline satisfaction rate. All those simulators aim at evaluating the public clouds without running numerous tests on real clouds, and focus more on power consumption and resource management.

However, from the perspective of users, none of the simulators can address their concerns because the particular mechanisms of Kafka are not considered. The users need a tool to help them choose proper configuration parameters in Kafka and this motivated us to conduct this research. A queueing model is used in our approach to evaluate the performance of Kafka, referring to other related works. The application processed at the cloud data centers has been modeled as an infinite $M/G/m/m+r$ queueing system in [15]. A queueing model has been proposed in [16] for elastic cloud apps where the service stage involves a load balancer and multiple workers running in parallel. The analytic model in [17] can evaluate simple cloud applications using message queueing as a service (MaaS), but still neglects the essential features in Kafka.

III. Overview

In this section we present a brief introduction of the Kafka messaging system to explore its mathematical characteristics.

Apache Kafka provides a publish-subscribe messaging service, where a producer (publisher) sends messages to a Kafka topic in the Kafka cluster (message brokers), and a consumer (subscriber) reads messages from the subscribed topic. A topic is a logical category of messages, for instance the producer will send the logs of a web server access records to the $serverLogs$ topic, while the records of the querying records on a website will be sent to the $searches$ topic. As depicted in Fig. 1, we observe 2 producers sending messages to 2 topics in this Kafka cluster, topic A and topic B respectively.

A topic may be stored in one or more partitions, which are the physical storage of messages in the Kafka cluster. The Kafka cluster consists of several brokers (Kafka servers), and all the partitions of the same topic are distributed among the brokers. In the example in Fig. 1 there are 3 brokers, and we see topic A consisting of 6 partitions while topic B with 3 partitions, which are denoted by tA-p{0-5} and tB-p{0-2} respectively.

Each partition is physically stored on disks as a series of segment files that are written in an append-only manner, and it is replicated across the Kafka broker nodes for fault tolerance, and we denote the replica of a partition with the suffix -r in

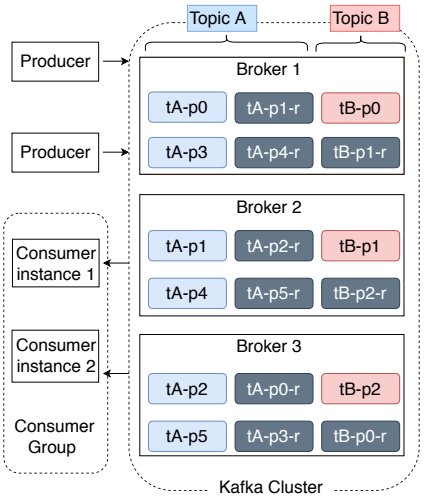

Fig. 1. Overview of Apache Kafka

Fig. 1. The messages of the leader partition tA-p0 on broker1 are replicated to the partition tA-p0-r on broker3, so once the broker1 server fails, tA-p0-r is chosen as the leader partition, which is similar for other partitions on broker1. Only the leader partition handles all reads and writes of messages with producer and consumer, which is performed in FIFO manner. Kafka uses partitions to scale a topic across many servers for producers to write messages in parallel, and also to facilitate parallel reading of consumers.

We see 2 consumer instances in our example, and they belong to the same consumer group. When the consumer group is subscribed to a topic, each consumer instance in the group will in parallel fetch messages from a different subset of the partitions in the topic. A consumer instance can fetch messages from multiple partitions, while one partition has to be consumed by only one consumer instance within the same consumer group. However, different consumer groups can independently consume the same messages and no coordination among consumer groups is necessary. As a result the number of partitions controls the maximum parallelism of the consumer group, and it is obvious that the number of consumer instances in the consumer group should not exceed the number of partitions in the subscribed topic, otherwise there will be idle consumer instances.

Cloud service vendors provide Apache Kafka as a service mainly in two different modes, the full-service mode and cluster-service mode. The VPC provided by AWS offers full-service mode where a user can create producer clients, consumer clients and a Kafka cluster on the cloud servers, and the user only needs a laptop connecting to the console for sending commands, as depicted in Fig. 2a. This mode is applicable when the upstream and downstream applications that cooperate with Kafka are also deployed on cloud services.

Under many circumstances data are generated or consumed locally, and users will run Kafka producer or consumer clients

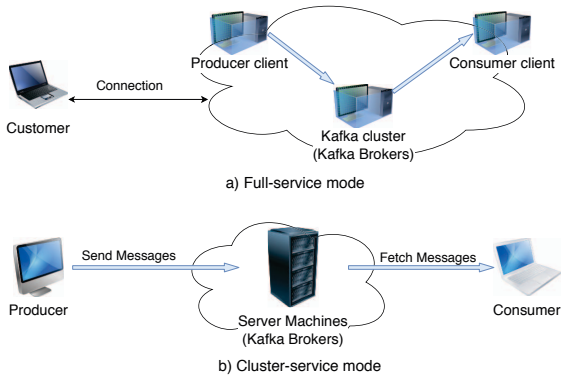

Fig. 2. Kafka as a Service

on their own machines. Then the choice should be cluster-service mode where cloud vendors like Confluent and Aiven provide a Kafka cluster as a service, as shown in Fig. 2b. In this mode producer clients send messages with local machines, then cloud servers distribute and store those messages for consumer clients to fetch. Normally users have to run test cases on a Kafka cluster and compare the performance results under different configuration parameters. However, the Kafka cluster needs a restart every time the configuration parameter is changed, and running test cases on cloud servers is time-consuming and expensive.

IV. Performance Model

Predicting the performance of a Kafka messaging system accurately plays a critical role in improving the user's efficiency of deploying Kafka cloud services. This section presents the performance model of Kafka to estimate the performance metrics under given configuration parameters.

A. Producer Throughput

The producer performance metric for which the user cares most is usually the throughput of messages, the number of messages sent from the producer to the cluster in a certain time interval.

A Kafka producer generally performs a batch-based transmission scheme, where messages are accumulated in memory to larger batches before they are sent over the network. Through the Kafka producer API, the user of the Kafka cloud service can configure the batching to accumulate no more than a fixed number of messages, and the topic those messages go to must be specified. Therefore the user should create a topic before sending messages from producer to the Kafka cluster, and the number of partitions under this topic is another indispensable configuration parameter. We assume that the upstream applications always generate sufficient messages to deliver to the Kafka messaging system. Thus the batches are always filled with maximum number of messages and send before the timeout is reached. The producer client will group batches into a single packet based on the number of leader partitions on a broker, as depicted in Fig. 3.

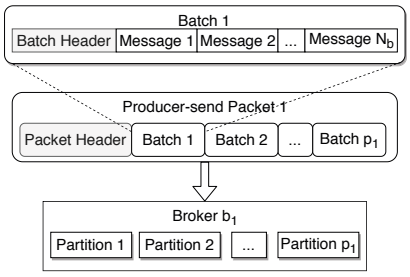

Fig. 3. Packet sent from producer

We use N_b to denote the number of messages per batch, and B to denote the number of brokers in the Kafka cluster. Then for any topic, the number of its leader partitions on broker $b_i (1 \leq i \leq B)$ can be denoted by p_i, and we define the assignment matrix of partitions as $\boldsymbol{P} = \begin{bmatrix} p_1 & p_2 & \dots & p_B \end{bmatrix}$. As illustrated in the example of Fig. 3, every packet sent by a producer to broker b_1 contains p_1 batches, because Kafka provides each leader partition with 1 batch in a single packet to achieve load balancing among partitions. Therefore, each element in the matrix $N_b\boldsymbol{P}$ denotes the number of messages per packet to its corresponding broker. Both batch and packet contain headers, as shown in Fig. 3. Those headers consist of many metadata attributes that are required to enable successful delivery. In this paper we define the intended messages in a packet as the payload, and the left headers are the overhead of the packet. We use S_m to denote the message size in bytes, then the batch size can be obtained as $S_b = (S_m \cdot N_b + batchHeaderSize)/compressionRatio$, where the compression ratio is depending on the type of compression chosen by users. The size of the packet received by each broker can be expressed in the form of a matrix:

$$\boldsymbol{S}_r = S_b \cdot \boldsymbol{P} + packetHeaderSize \cdot \boldsymbol{e}_B^T \tag{1}$$

where $\boldsymbol{e}_B$ is a $B \times 1$ column vector with all B elements equal to 1.

Considering all the packets sent from producer to any individual broker b_i, we define the period of time between two subsequent packets as the packet send interval of b_i. In our experiments we observe that a Kafka producer sends packets to different brokers in random order, and the mean packet send interval largely depends on the size of a packet. Therefore we assume the packet send interval is a function of the packet size $\boldsymbol{G}_s = f_p(\boldsymbol{S}_r)$, where each element in the $1 \times B$ matrix $\boldsymbol{G}_s$ represents the mean send interval of packets to the corresponding broker. In a correlation analysis of packet sizes and packet send intervals we have obtained the parameters of the function $f_p(x)$. The experimental results are explained in Section V. Then we define the number of packets sent from a producer to a broker in an unit of time as the packet rate of this broker, and use $\boldsymbol{R}_s$ to denote the matrix of packet rates to each broker which satisfies the equation $\boldsymbol{R}_s \circ \boldsymbol{G}_s = \boldsymbol{e}_B^T$, where $\circ$ stands for the Hadamard product. Given two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of the same dimension, the

Hadamard product $\boldsymbol{A} \circ \boldsymbol{B}$ is a matrix of the same dimension with elements $(A \circ B)_{ij} = (A)_{ij}(B)_{ij}$. Thus we can obtain the mean number of packets sent from producer per unit of time, namely the throughput of producer-send packets in quantity $X_{sp} = \boldsymbol{R}_s \cdot \boldsymbol{e}_B$ and the throughput of producer-send packets in bytes per second $X'_{sp} = (\boldsymbol{R}_s \circ \boldsymbol{S}_r) \cdot \boldsymbol{e}_B$, which is also the network bandwidth required to send packets.

The throughput in number of messages can be denoted by $X_{sm} = (\boldsymbol{R}_s \circ N_b \boldsymbol{P}) \cdot \boldsymbol{e}_B$ as well as the message throughput in bytes per second $X'_{sm} = (S_m(\boldsymbol{R}_s \circ N_b \boldsymbol{P}))\cdot$.

The required network bandwidth is determined by X'_{sp}, and we define the relative message throughput in the occupied bandwidth as the relative payload of producer-send packets $\varphi_{ps} = X'_{sm}/X'_{sp}$, therefore the relative overhead $\varphi_{os} = 1 - \varphi_{ps}$.

B. Broker Storage

As mentioned in Section III, a partition is an ordered, immutable sequence of messages which can not be split across brokers. This means whenever a new message is published to a topic, it is appended to the end of a partition and assigned with an offset, which is a per-partition monotonically increasing sequence number. There are multiple segment files with the same size (by default it is 500MB) in a partition. At first segment file 1 is generated, including one log file, which is the actual file where messages are stored, and one index file, which stores the metadata information. When the size of the log file reaches 500MB, the next segment file (file 2) will be generated and its log file and index file will be named based on the offset value of the last message from the segment file 1. The metadata in the index file points to the physical offset addresses of the messages in the corresponding log file. The index file is made up of 8 byte entries with 4 bytes to store the offset relative to the base offset and 4 bytes to store the position. In Kafka the configuration parameter $index.interval.bytes$ sets an index interval which describes how frequently (after how many bytes) an index entry will be added. In cloud services the disk space is quite limited, so in order to save disk space, the user of Kafka as a service can specify how long data should be retained, after which time the data will be deleted. The configuration parameter that controls the log retention time is $log.retention.hours$ and the default value is 168 hours.

If a producer has been producing messages for a period of time T_p, we can denote the used storage of the Kafka cluster by matrix:

$$\boldsymbol{H} = (1 + \frac{8}{index.interval.bytes}) T_p (\boldsymbol{R}_s \circ \boldsymbol{S}_r) \qquad (2)$$

where each element in $\boldsymbol{H}$ denotes the size of stored segment files on the corresponding broker.

C. Consumer Throughput

Similar to the process in the producer, the messages fetched by a consumer are also effectively batched. More specifically, the message format on brokers is exactly the same as what the producer sends through the network, which means messages are still batched on brokers. The consumer sends fetch requests to the brokers in the Kafka cluster, and each broker responds with a packet that contains several batches of messages. The number of batches in a consumer-fetch packet is determined by the number of partitions on the broker, because each partition will provide at least k batches in the response packet, where k is a non-zero positive integer. Therefore we can denote the number of messages fetched per request as $kN_b\boldsymbol{P}$, and define the size of a packet that a consumer fetches from each broker through a single fetch request as the fetch size of the consumer $\boldsymbol{S}_f = S_b \cdot \boldsymbol{P} + packetHeaderSize \cdot \boldsymbol{I}_B^T$.

In the situation that there are multiple consumer instances in a consumer group, the elements in matrix $\boldsymbol{P}$ should be adjusted to the number of partitions that are assigned to the specific consumer instance. By default the consumer uses the $poll$ method to fetch messages, thus when there are plenty of messages to be consumed, the rate of the consumer-fetch packets depends on how fast the broker accumulates enough batches in a packet. We study the correlation between the size of the consumer-fetch packet and the time interval of sending these packets from a broker to the consumer. We use the matrix $\boldsymbol{G}_f = f_c(\boldsymbol{S}_f)$ to denote the interval on each broker, and $\boldsymbol{R}_f$ to denote the matrix of consumer-fetch packet rates which fits $\boldsymbol{R}_f \circ \boldsymbol{G}_f = \boldsymbol{e}_B^T$. Then the throughput of consumer-fetch packets is $X_{fp} = \boldsymbol{R}_f \cdot \boldsymbol{e}_B$ in number of messages and $X'_{fp} = (\boldsymbol{R}_f \circ \boldsymbol{S}_f) \cdot \boldsymbol{e}_B$ in bytes.

D. Packet Latency

Sometimes the users of Kafka as a service want to make a real-time analysis, and their metric of concern is the end-to-end latency of a packet, which is defined as the time between the sending of a packet from the producer to the moment the packet is received by the consumer. In queueing theory the end-to-end latency is the response time of packets, and we denote the mean response time of packets through each broker by a $1 \times B$ matrix $\boldsymbol{R}$. According to the description in Section IV-A, over a certain period of time, the number of packets that are sent to each partition is almost equal. Thus we use the number of partitions on a broker b_i as the weight of the mean response time on this broker, then we can take the weighted mean response time of all brokers as the mean response time of packets through the Kafka cluster, given by $(\boldsymbol{R} \circ \boldsymbol{P}) \cdot \boldsymbol{e}_B / \boldsymbol{P} \cdot \boldsymbol{e}_B$. We model one of the brokers as server in a queueing model, which deals with 3 types of request streams. The first request stream is exactly the packet sent from producer with arrival rate λ_s, and the service time is from when the broker receives a packet to when it is stored on disk, as illustrated in Fig. 4. The second request stream is the fetch request with arrival rate λ_r from the other brokers to replicate the messages on leader partitions, and the service time is the time spent to send replicated messages. The last request stream received by a broker is the fetch request with arrival rate λ_f from a consumer, and the service time is the time to send packets to the consumer. According to the research in [18], we can use a Poisson process to approximate the arrival process of requests. Since any distribution with a strictly positive support

in $(0, \infty)$ can be approximated arbitrarily close by a phase-type distribution [19], we assume the service time is phase-type (PH) distributed, represented by the three distributions $PH_s(\boldsymbol{\pi_s}, \boldsymbol{T_s})$, $PH_r(\boldsymbol{\pi_r}, \boldsymbol{T_r})$ and $PH_f(\boldsymbol{\pi_f}, \boldsymbol{T_f})$ respectively. Therefore, the process of a broker dealing with each job is an $M/PH/1$ queue.

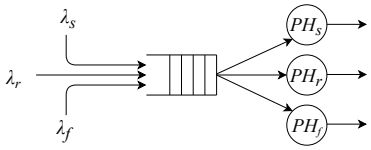

Fig. 4. The queueing model of a broker

In a phase-type distribution with representation $(\boldsymbol{\pi}, \boldsymbol{D})$, $\boldsymbol{\pi}$ is the initial probability vector and $\boldsymbol{D}$ is the generator matrix of an absorbing Markov chain. An $M/PH/1$ queue can be analyzed as a Quasi-birth-death (QBD) process with the state space $M = \{0, (i, j), i \leq 1, 1 \leq j \leq v\}$ and state 0 stands for the empty queue, while state (i, j) represents i jobs in the system and the service process is in phase j. We use λ to denote the arrival rate of the Poisson process, and by symbol $\boldsymbol{I}$ we always denote an identity matrix of the dimension appropriate to the formula in which it appears, then we obtain the generator matrix:

$$\boldsymbol{Q} = \begin{bmatrix} -\lambda & \lambda\boldsymbol{\pi} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\tau} & \boldsymbol{D} - \lambda\boldsymbol{I} & \lambda\boldsymbol{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\tau\pi} & \boldsymbol{D} - \lambda\boldsymbol{I} & \lambda\boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\tau\pi} & \boldsymbol{D} - \lambda\boldsymbol{I} & \lambda\boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \ddots & \ddots & \ddots \end{bmatrix} \tag{3}$$

and the steady state equations:

$$-\lambda x_0 + x_1\boldsymbol{\tau} = 0 \tag{4}$$

$$\lambda x_0\boldsymbol{\pi} + x_1(\boldsymbol{D} - \lambda\boldsymbol{I}) + x_2\boldsymbol{\tau\pi} = 0 \tag{5}$$

$$\lambda x_{i-1} + x_i(\boldsymbol{D} - \lambda\boldsymbol{I}) + x_{i+1}\boldsymbol{\tau\pi} = 0 \tag{6}$$

where $\boldsymbol{x} = [x_0, x_1, x_2, \cdots]$ is the stationary probability vector and $\sum_{i=0}^{\infty} x_i = 1$.

Multiplying Equation (5) and (6) by the column vector $\boldsymbol{e}$ on the right, and combining the results we obtain:

$$x_i = x_0\boldsymbol{\pi}\boldsymbol{M^i}, \quad i \geq 1 \tag{7}$$

where $\boldsymbol{M} = \lambda(\lambda\boldsymbol{I} - \lambda\boldsymbol{e\pi} - \boldsymbol{D})^{-1}$, then the average number of jobs in the queue can be denoted as:

$$E[N] = \sum_{i=1}^{\infty} ix_i\boldsymbol{e} = x_1 \sum_{i=1}^{\infty} \frac{d}{d\boldsymbol{M}}\boldsymbol{M}^i\boldsymbol{e} = x_1(\boldsymbol{I} - \boldsymbol{M})^{-2}\boldsymbol{e} \tag{8}$$

and according to Little's law, the mean time a job spends in the system is:

$$E[R] = \frac{E[N]}{\lambda} = \frac{x_1(\boldsymbol{I} - \boldsymbol{M})^{-2}\boldsymbol{e}}{\lambda} \tag{9}$$

We use $E[R_s]$, $E[R_r]$ and $E[R_f]$, respectively, to denote the mean response time of the three jobs mentioned above. When a user chooses the synchronous replication mode, a message will be available to consumers only after it is replicated to all follower partitions, thus the average end-to-end latency of a packet equals $E[R_s] + E[R_r] + E[R_f]$. However, in the asynchronous replication mode the consumer can fetch a message as long as a it is stored in the leader partition, without waiting for the replication process. In this situation the average end-to-end latency is $E[R_s] + E[R_f]$.

V. Experimental evaluation

In this section we evaluate the proposed model from different perspectives for multiple scenarios.

Our Kafka system is built from the latest version 2.1.1, orchestrated by Zookeeper 3.4.12 and each broker is started on a single PC. We built the Kafka cluster with 3 brokers to simulate the environment of using Kafka as a service, and use two other PCs as the producer client and consumer client, respectively. Each PC is equipped with 8G memory and all have the same type of CPU: Intel(R) core(TM)2 Quad CPU Q9550@2.83GHz, running Debian 4.9.144-3.1, so our servers are homogeneous, which is common for many cloud platforms.

We study the use case where a user buys Kafka cloud service to use the message broker for a server log analysis application. The data source is web server access logs from NASA Kennedy Space Center web servers[18], and each record in these logs is a message in Kafka. Our experiments revolve around the most relevant metrics, such as the throughput of producer and the relative payload under heavy load, the utilization of network bandwidth, the disk space limits and Kafka data retention time.

A. Correlation Analysis

The performance of the Kafka system varies significantly depending on the message type and the compute power of machines (e.g., CPU speed, memory size), making it extremely difficult to build a general model for all cases. Referring to the method in [20], we collected execution traces using a small fraction of the source data, and obtained the essential parameters for our performance model. We analyzed the correlation between producer-send packet size $\boldsymbol{S}_r$ and the send interval $\boldsymbol{G}_s$ to find function $f_p(x)$ through the metrics recorded in the producer throughput tests. All tests run for at least 60 seconds after setup and messages are sent with different batch sizes to disjoint topics. The number of messages in one batch N_b ranges from 1 to 50, and we create 10 topics with 3 to 12 partitions, therefore the packet sizes in our tests vary according to equation (1). Other configurations are all set to the default values. We illustrate part of our correlation analysis results in Fig. 5, including the scatter points for the mean packet sizes

and send intervals for one broker, as well as the lines of best fit.

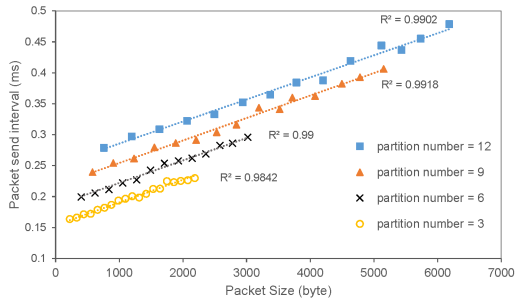

Fig. 5. Correlation analysis of producer-send packet sizes and intervals

In Fig. 5 we can observe a strong, positive and linear correlation between packet sizes and send intervals, grouped by different number of partitions. For the packets of almost the same size, a larger number of partitions causes certain increments on send interval, because the Kafka producer needs longer to arrange metadata information of each partition. The packet send interval in a certain topic increases linearly as the size of the packets grows, and the slopes of different topics are approximately equal. Every coefficient of determination in our correlation analysis results, denoted by R^2, is close to 0.99, as depicted in Fig. 5. Likewise, we study the correlation between the consumer-fetch packet size $\boldsymbol{S}_f$ and the interval $\boldsymbol{G}_f$ and the R^2 is closed to 0.95. This indicates that the correlation between packet sizes and send intervals is positive and we can use the results to complete the function $f_p(x)$ and $f_c(x)$ in the performance model.

B. Network Bandwidth Evaluation

From [13] it is known that the network bandwidth provided by cloud vendors varies significantly, and the available bandwidth for a user is always limited. Moreover, a Kafka cluster has the ability to enforce network bandwidth quotas on packets to control the resources used by the clients. By default, each unique client group receives a fixed quota in bytes/sec as configured by the cluster, which is defined on a per-broker basis. When the upstream application generates many messages to the Kafka system, the user expects to better utilize the available network bandwidth aiming at larger throughput. Since the results of producer-send packets and consumer-fetch packets are similar, we take the example of the former one to analyze and compare the numerical results and experimental results.

In Section IV-A we stated the closed-form expression of packet throughput under heavy load, and the required network bandwidth for each broker can be obtained from the equations in the section. In Fig. 6 we select four sets of experimental results under heavy load in comparison with the model prediction results, separated by the number of partitions in the topic. Through the experiments we intend to explore the impact of batch size and number of partitions on the throughput of packets from producer client in Megabytes per second, and the relative overhead φ_{os}.

The scatter points in Fig. 6 are the experimental results we collected while the curves are generated from our performance model, and the results of the model are close to the experimental results. We can see that accumulating more messages in one batch can indeed increase the packet throughput, but the improvement is less when N_b grows. E.g. when sending messages to the topic with 3 partitions, increasing N_b from 1 to 20 can boost the throughput by a factor of approximately 6.75, while configuring N_b from 20 to 40 only improves the throughput by 37%.

We employ the mean absolute percentage error (MAPE) to measure the prediction accuracy of our model, which is given by:

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{E_i - M_i}{E_i} \right| \tag{10}$$

where E_i is the experimental result and M_i is the prediction result from our model, and n is the number of fitted points. For the results of packet throughput, the MAPE is close to 2% when the throughput is below 25 MB/s, and with higher throughput the MAPE is still less than 10%. Compared to sending messages without batching (i.e. 1 message per batch), collecting 10 messages per batch can sharply decrease the relative overhead from over 50% to less than 15%, as illustrated in Fig. 6. However, the relative overhead remains stable around 10% after the throughput of packets reaches 30MB/s. The MAPE of the relative overhead also achieves 10%. The network bandwidth quota for a user of Kafka as a service is limited, and our performance model can help the user to choose a suitable number of partitions and batch size to achieve good throughput. E.g. if the partition number is set to 3 and the quota is 20 MB/s, our prediction results indicate that setting N_b to 12 can achieve a packet throughput of 19 MB/s, and adding more messages to the batch will risk exhaustion of the bandwidth and exceeding the available memory.

For the choice of a good number of partitions in a topic, we use our model to study the effect of partition number on the relative payload of producer-send packets, given network bandwidth quotas of 20 MB/s, 30 MB/s and 40 MB/s. The batch size in each test is configured to approach the highest throughput at the quota limit, therefore the network bandwidth is almost fully utilized. As the results in Fig. 7 illustrate, we can observe that for any topic, assigning a higher network bandwidth quota achieves higher relative payload. Since the servers in our experiments are considered homogeneous, we define the topics with partitions evenly distributed among brokers as the load-balanced topic, and in our tests the load-balanced topics are those with the number of partitions equal to $3N$, where N denotes a non-zero positive integer. We can see from Fig. 7 that given the same network bandwidth quota, the relative payload of load-balanced topics decreases as the number of partitions increases. This happens due to the overhead of maintaining the extra partitions' metadata.

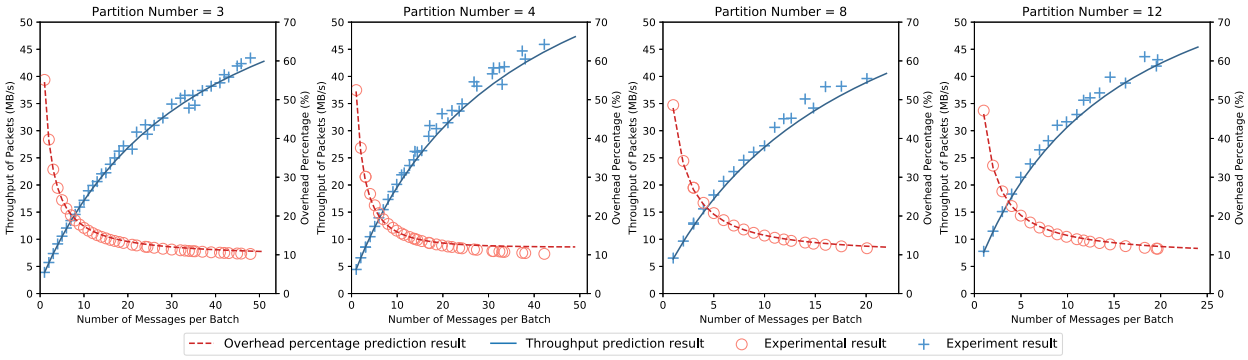

Fig. 6. Experiment and prediction results of producer-send packet throughput

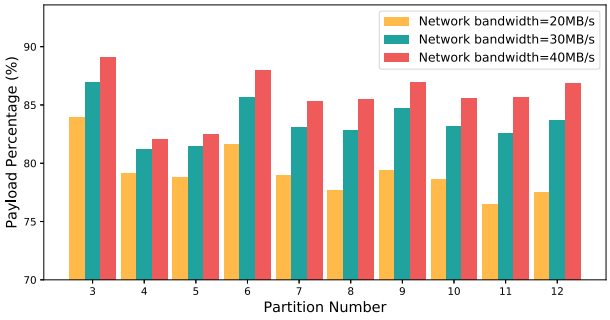

Fig. 7. Partition number and relative payload of producer-send packets given different network bandwidth quotas

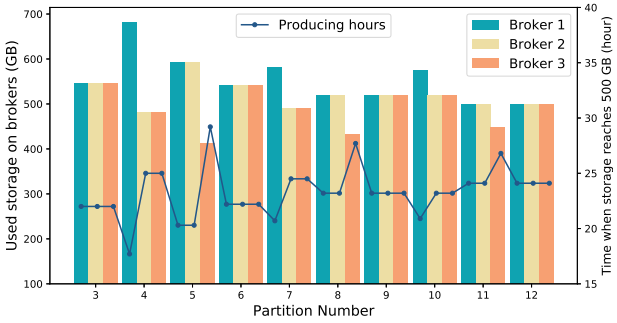

Fig. 8. Used storage on brokers after producing messages for 24 hours

Observing relative payload of the topics that are not load-balanced we find that the relative payload is always below their adjacent load-balanced topics. Thus we suggest the users try to avoid creating topics with the number of partitions that are not load-balanced among brokers, specifically when the servers are homogeneous.

C. Disk Space Evaluation

Limited disk space on cloud servers is an important concern of users, especially when the producer client runs a heavy load for a long time. Our performance model can predict the used storage of disk on each broker over time, and by observing the prediction results the user can set a proper value for the retention configuration parameters. Fig. 8 depicts the used disk space by messages on each broker, after the producer client runs for 24 hours with 20 MB/s network bandwidth quota. The line in this figure specifies the production time on each broker when the size of stored messages on disk reaches 500 GB.

Apparently, the used disk space by messages is almost equal on each broker when the topic is load-balanced, while the other topics see imbalanced sizes of stored messages among brokers. The difference of used disk space among brokers is shrinking as the number of partitions in those imbalanced topics grows. E.g for the topic with 4 partitions the difference of storage between two brokers can be around 200 GB, yet the difference in the topic with 11 partitions is 50 GB. From observing the results of production time, the user can set appropriate configuration parameters to control the retention time of the messages on disks. For the topics that are not load-balanced, there will be some brokers that reach a certain size of stored messages much earlier than the other brokers, and the user has to specify the retention parameter on those brokers.

D. End-to-end Latency

In our latency experiments, we first used a small fraction of the web server log data to collect the service time data for the three types of requests mentioned in Section IV-D to evaluate the end-to-end latency. The producers and consumers are run on two different PCs and we use the JMX (Java Management Extensions) tool to collect the total time to store a producer packet, the time to send replicas to other brokers, and the time to handle a fetch request from the consumer.

We use the fitting tool HyperStar2 [21] to obtain the phase-type distribution of service times including the seen correlation and use the fitted results in our queueing model. The probability density function of the service time and the fitted process are shown in Fig. 9.

In our experiments, inspired by [18], packets are sent from producers with exponentially distributed inter-packet times. In

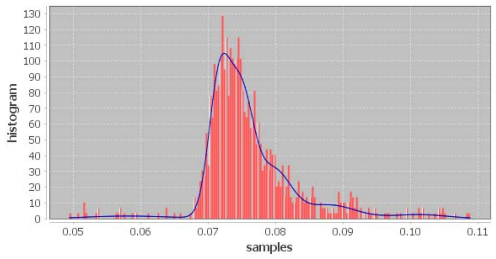

Fig. 9. Probability density function of the service time distribution fitting

Fig. 10 we compare the experimental results and the numerical results from the queueing model under different arrival rates.

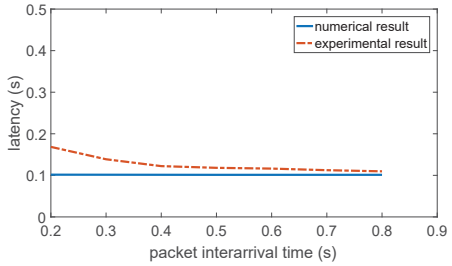

Fig. 10. Latency results under different arrival rate

We can observe that the prediction model underestimates the mean end-to-end latency of packets, especially when the interarrival time is less than $0.3s$. The model works well with lower arrival rate, as shown in Fig.10, when the message interarrival time is larger than $0.4s$, the MAPE of the numerical result could be less than 10%. The difference has a decreasing trend with lower load.

VI. Conclusion

This paper proposes a performance model to predict the performance of Kafka messaging system. Our model provides users an easy and reliable way to study the effects of the configuration parameters on the performance of their Kafka system. By tuning the batch size and the number of partitions in the model, users can observe the changes of producer throughput and relative payload under heavy load. The mean absolute percentage error is close to 2% when the network bandwidth quota does not exceed 25 MB/s, and this can help the user choose appropriate configuration parameters. The model also predicts the disk occupancy over time for changing number of partitions, through which users can adjust the retention parameters in Kafka to avoid using up disk space on cloud servers. We use $M/PH/1$ queueing model to evaluate the mean time a packet travels through a Kafka cluster, which is the main concern when collaborating with real-time applications.

In our future work, we will analyze more configuration parameters to complete our performance model, and study on how to better support the choice of the best Kafka configuration.

References

[1] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[2] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, and J. Stein, "Building a replicated logging system with apache kafka," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1654–1655, 2015.

[3] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao, and V. Y. Ye, "Building linkedin's real-time activity data pipeline." *IEEE Data Eng. Bull.*, vol. 35, no. 2, pp. 33–45, 2012.

[4] J. Kreps. Benchmarking apache kafka: 2 million writes per second (on three cheap machines). [Online]. Available: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

[5] ApacheKafka. Powered by - apache kafka - the apache software foundation. [Online]. Available: https://kafka.apache.org/powered-by

[6] AmazonWebServices. Amazon managed streaming for kafka (msk). [Online]. Available: https://aws.amazon.com/msk/

[7] IBM. Ibm cloud: Event streams. [Online]. Available: https://console.bluemix.net/catalog/services/event-streams

[8] MicrosoftAzure. Hdinsight: Easy, cost-effective, enterprise-grade service for open source analytics. [Online]. Available: https://azure.microsoft.com/en-us/services/hdinsight/

[9] Confluent. Confluent cloud: Apache kafka re-engineered for the cloud. [Online]. Available: https://www.confluent.io/confluent-cloud/

[10] 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*. ACM, 2017, pp. 227–238.

[11] V. John and X. Liu, "A survey of distributed message broker queues," *arXiv preprint arXiv:1704.00411*, 2017.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[13] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 1–14.

[14] I. K. Kim, W. Wang, and M. Humphrey, "Pics: A public iaas cloud simulator," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 211–220.

[15] H. Khazaei, J. Mišić, and V. B. Mišić, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on Parallel & Distributed Systems*, no. 5, pp. 936–943, 2011.

[16] K. Salah and R. Boutaba, "Estimating service response time for elastic cloud applications," in *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*. IEEE, 2012, pp. 12–16.

[17] K. Salah and T. R. Sheltami, "Performance modeling of cloud apps using message queueing as a service (maas)," in *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*. IEEE, 2017, pp. 65–71.

[18] S. Hagihara, Y. Fushihara, M. Shimakawa, M. Tomoishi, and N. Yonezaki, "Web server access trend analysis based on the poisson distribution," in *Proceedings of the 6th International Conference on Software and Computer Applications*. ACM, 2017, pp. 256–261.

[19] P. Buchholz, J. Kriege, and I. Felko, *Input modeling with phase-type distributions and Markov models: theory and applications*. Springer, 2014.

[20] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 2015, pp. 166–173.

[21] Z. Shang, T. Meng, and K. Wolter, "Hyperstar2: Easy distribution fitting of correlated data," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 2017, pp. 139–142.