

Evaluation of Highly Available Cloud Streaming Systems for Performance and Price

Dung Nguyen
School of Computing
Clemson University
dungn@clemson.edu

Andre Luckow
School of Computing
Clemson University
aluckow@clemson.edu

Edward B. Duffy
Elect. & Comp. Eng.
Clemson University
duffy2@clemson.edu

Ken Kennedy
School of Computing
Clemson University
kkenned@clemson.edu

Amy Apon
School of Computing
Clemson University
aapon@clemson.edu

Abstract—This paper presents a systematic evaluation of Amazon Kinesis and Apache Kafka for meeting highly demanding application requirements. Results show that Kinesis and Kafka can provide high reliability, performance and scalability. Cost and performance trade-offs of Kinesis and Kafka are presented for a variety of application data rates, resource utilization, and resource configurations.

I. INTRODUCTION

This paper evaluates cloud streaming solutions for a use case that requires both high performance and high availability of the message infrastructure. An example application of such a stream system is one that captures log messages and detects potential denial-of-service attacks in real time. Because the goal is to identify potential security threats, the system must be extremely reliable, without loss of any messages at all. We also require manageable costs.

Data streaming frameworks vary greatly with respect to their level of reliability, message production and consumption performance, and their ability to adaptively scale to avoid losing data or to manage costs. Two main architectural framework options exist: open source tools such as Kafka and cloud-native solutions such as Kinesis, which their specifications and architectures are described in [1] and [2]. While open source solutions provide great flexibility, the management and scaling efforts are typically higher than for managed cloud solutions.

The majority of prior related research focuses on qualitative comparisons such as architectures that use traditional, on-premise deployments. Formal evaluation of streaming processing architectures using the commercial cloud has emerged only recently. Luckow et.al. [3] investigate stream processing architectures in the context of high performance and cloud computing. The performance of Apache Kafka has been studied previously [1], [4], [5], and some investigations have focused on cloud architectures for Kafka [6] with high-level guidance. There are a few previous studies of Kafka and Kinesis [7] [8] [9], focus on comparing their delivery mechanism and supported features.

This paper provides the first comprehensive analysis of the performance and costs trade-offs of Kafka and Kinesis using comparable cloud deployments. Our main contributions are:

- We present a systematic evaluation of reliability and performance of the Amazon Web Service (AWS) Kinesis

cloud-native solution, and the Kafka open source solution deployed using AWS.

- We compare costs of the two solutions for a range of configurations that meet performance targets (Section III).

Our results show that Kinesis throttles producing clients, ensuring that the infrastructure will have sufficient buffers to receive all sent messages, while Kafka does not throttle producing clients and must maintain sufficient buffers in the infrastructure to avoid dropping messages. We analyze the cost and performance trade-offs of Kinesis and Kafka for a variety of data rates, resource utilization, and resource configurations.

II. PERFORMANCE AND RELIABILITY EVALUATION

In this section we describe the experimental evaluation of the performance and reliability of Kinesis and Kafka. Our message stream is synthetically generated with some variation to provide ranges of evaluation for different workloads. In each experiment, messages have a fixed length in bytes, and we assume that there are no malformed messages. We perform the evaluation with different counts of shards, or partitions, for values 1, 2, 4, 8, 16, and 32. For each count of shards/partitions, the producers generate six levels of data velocities: 1, 2, 4, 8, 16, and 32 MB/s. Throughput, latency, buffer time, and errors metrics for both consumers and producers are measured. Experiments also record the delivery status of all records to confirm if all produced records are received.

We primarily report the results for “m4.large”, which is the smallest instance type that provides non-burstable network performance in AWS, which is critical for real-time performance. The only configuration parameter of Kinesis is the count of shards. All experiments are repeated to provide statistically significant results. To test reliable delivery of messages, we added an ID to each message and then recorded its delivery to the consumers. This allowed us to test the effect of producing a burst messages at a rate that is higher than that guaranteed by the configuration options and to observe the behavior of the systems under extremely high load.

In all of the experiments, only one message was dropped in the Kinesis experiments. This occurred for an experiment in which Kinesis was configured with one shard and the data rate was 32 MB/s, which is 32 times the guaranteed throughput of a single Kinesis shard. We were not able to reproduce this error. No data loss was observed with the Kafka experiments.

Kinesis Producer Throttling: When the message production rates by Kinesis Producers are larger than the guaranteed rate of the Kinesis Stream’s settings, we expect and observe throttling by Kinesis. It is the responsibility of the producer to buffer the messages and resend them if redelivery is required.

We find that Kinesis begins throttling at a rate that is lower than the theoretical (i.e., configured) throughput. Fig. 1 shows Kinesis producer throughput compared to the theoretical throughput for each count of shards from 1 to 32. For one shard, the producer can produce at 1MB/s, which is the same as the theoretical rate. However, Fig. 1 shows that the producer is not able to produce to the level of theoretical throughput. That is, messages are returned with a failure code. The impact is higher for higher shard counts. The ratio of achieved throughput to theoretical throughput for 32 shards is close to 85%. The architectural design of Kinesis that makes it highly reliable by avoiding overload impacts the throughput performance at higher data rates.

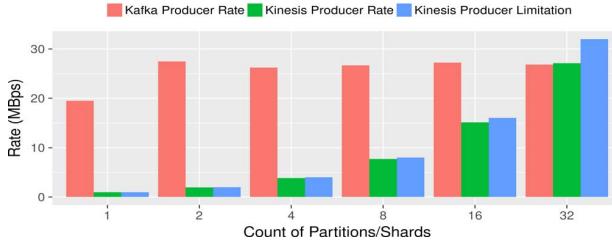


Fig. 1: Kinesis producer vs theoretical throughput. The maximum throughput that Kinesis actually receives from producers is less than the theoretical limit. The Kinesis throughput limit (in MBps) equals the shard count.

Kinesis Error Reporting: Kinesis provides a guaranteed level of message delivery that depends on the count of shards. When the message rate exceeds the guarantee, messages may still be delivered successfully in many cases. However, when a message cannot be delivered from the Producer to Kinesis, Kinesis returns an error code. It is up to the Producer to take action to mitigate data loss. Fig. 2 graphs the relation between Producer throughput and this error rate. Actual data loss can only be verified on the Consumer side. In our experiments, the Producer client was set to retry an indefinite number of times, and an error rate of up to 10% still provided data transmission without loss, even with throttling, due to the retries.

Fig. 2 shows in the upper left subchart that errors are reported for a message rate of 2 MB/s with a single shard (though still with successful delivery after retries), but that errors are not reported until the message rate reaches 32 MB/s in the case of 32 shards, as shown in the lower right subchart.

Kinesis Latency with High Producer Data Rates: When the production rate is high, producers buffer records until the records’ TTL expires. This behavior causes the time to the delivery of the messages (i.e, the latencies) to increase.

Fig. 3 shows that when the Producer throughput does not exceed guaranteed throughput, the latency is small and stable. Latency increases as the throughput increases. The maximum latency is approximately 3 seconds (as shown on the log scale). Fig. 3 also shows that the latency for a higher number of shards

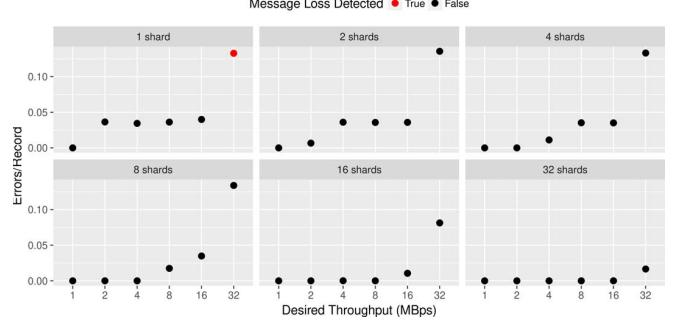


Fig. 2: Kinesis error rate (which indicates throttling) and producer throughput for different shard counts. The rate is close to 0 when the generation rate is smaller than the stream capacity. Only for the case when 32 MB/s of data are put into a stream configured for 1 MB/s is message loss detected.

is the same as the latency for a lower number of shards as long as the message rate is below the guaranteed throughput.

Kinesis streams behave as expected in most cases. The maximum throughput that a stream can handle in practice is close to its theoretical throughput. When the data rate exceeds the stream’s designed throughput then the producer stores non-delivered data in its buffer until they are successfully sent. By configuring the expiration time of a record to high values, the producer will try to resend the data until all data in the stream is received by consumers.

Kafka Latency Compared to Kinesis Latency: Kafka does not throttle producers so we do not need to test for that effect. We evaluate Kafka latency for increasing message rates for different counts of partitions, and compare the result to that of Kinesis with an equivalent count of shards. We execute Kafka on “m4.large” instance.

Fig. 3 shows that for the smallest message rate Kafka has lower latency than Kinesis. However, as the message rate increases, Kafka latency increases to a higher value than that of Kinesis. This is true for all message rates except the highest rate with 32 shards (for Kinesis) and 32 partitions (for Kafka). In that case the message rate does not exceed the capability of the systems as configured and the latency stays low.

Throughput Comparison: Fig. 1 compares Kinesis throughput and Kafka throughput for a fixed high message rate and a varying number of shards or partitions. Fig. 1 illustrates again the effect of the throttling by Kinesis of the producer throughput. For a fixed high message rate, Kinesis throughput increases as the count of shards increases. That is, it passes the messages to the consumer at a higher rate as more shards are added. Kafka throughput is high for even a single partition, though higher for more partitions than a single partition.

Consumer Performance: Fig. 4 shows an important characteristic of the scalability in the performance of consumers in both Kafka and Kinesis systems. They are strongly affected by the number of shards/partitions. In both systems, the higher the number of shards/partitions, the more consumer units can concurrently process data.

III. COST ANALYSIS

Cost analysis is an important step in the engineering of a complete end-to-end streaming system. We compare Kinesis

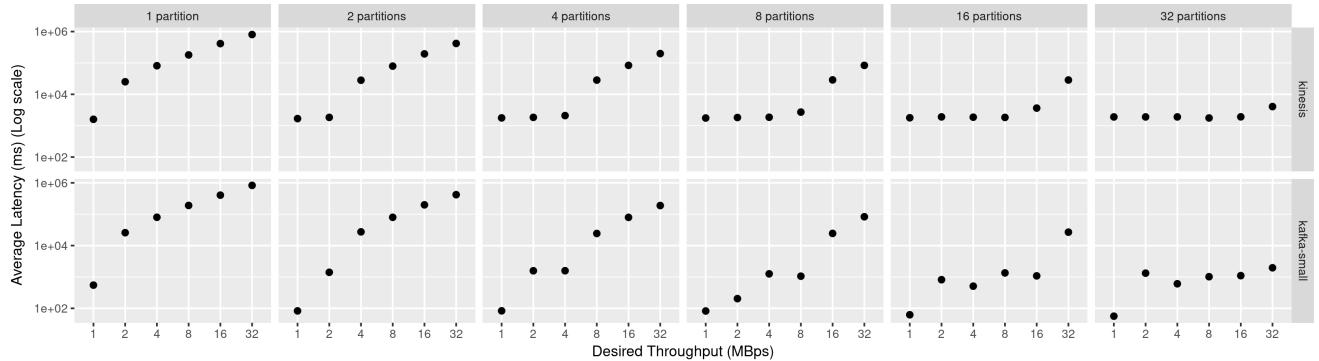


Fig. 3: Relation between producer throughput and latency for different numbers of shards/partitions. For each count of shards, the Kinesis producer throughput (upper row) is limited by the Stream capacity and the corresponding latency grows. Increasing the number of shards in the Kinesis Stream only helps when producers requires a higher throughput. For Kafka, using more partitions may decreases the latency, even the producer throughput does not exceed its capacity and a smaller number of partitions can handle the throughput well. With 32 shards/partitions, Kinesis and Kafka have the similar latencies.

and Kafka under various configuration scenarios using the of m4.large, our weakest instance type, in the Northern California region of AWS. Different pricing models are possible for computer instances in AWS, including Spot, On-Demand, Reserved, and Scheduled Instances. For our comparison we consider the cheapest instance purchase model for Kafka, Reserved Instances requiring a contract of three years with payment all upfront.

Parameters for Cost Calculation: Kinesis pricing is mainly based on shard hours, currently US \$0.0182 per shard hour in AWS Northern California Region. A surcharge of US \$0.025 applies for extended data retention, which keeps the data alive inside a Kinesis Stream up to seven days. By default, data are temporarily stored inside the Kinesis Stream for twenty-four hours from the time of reception.

Amazon charges for the amount of incoming data. A PUT Payload Unit (PPU) is counted in each 25KB chunk of data or a whole record, depends on which is smaller. For example, a record up to 25 KB is one PPU, while a 30 KB record is counted as two PPUs. In California Region, Amazon charges PPU by chunks of millions, with the price of US \$0.0185 per million units. Other data transmissions are free, including the data transferred from Amazon Kinesis to consumers.

Kinesis users typically also pay for Amazon CloudWatch for the metrics used to monitor the health and workload of Amazon Kinesis. Though the use of this service is optional, monitoring the system is extremely critical, more so than normal applications that also utilize the CloudWatch service. The price of this service depends on how many metrics are set up for the Kinesis, and how often these metrics are requested. In our experiments, the first 1,000,000 requests are free of charge, while the first 10,000 metrics cost US \$0.30 per month for each metric-month. Also, if the consumers use the Amazon Kinesis Consumer Library (for example, Kinesis integration for Apache Spark), AWS charges US \$0.81 per month for each DynamoDB instance used by one application, which is provisioned by the library by default. The capacity of the default DynamoDB table includes 10 reads/writes per second. With a higher frequency of checkpoints in the consumers or a larger number of shards in the stream, it requires a higher

read/write units of the DynamoDB.

An industrial Kafka system must consist of at least two machines with independent storage devices to eliminate the single point of failure in the broker. Each EC2 instance requires two EBS volumes to work as two physical disks in a typical computer. The first 20GB volume is used to store the OS, necessary software applications, and libraries. With the cheapest type of EBS this volume costs US \$4 per month. The second volume is designed to store Kafka's temporary data. The required size of the second volume depends on the incoming data throughput and the length of data retention before deletion. With an incoming throughput of 1 MB/s the amount of data received every 24 hours is 86.4 GB. To accommodate the overhead in storing data, we provision a volume of 100 GB to store each unit of incoming throughput up to 1 MB/s in 24 hours.

In general, AWS charges for the amount of data transferred in and out of EC2 instances, as well as data transferred between Availability Zones (AZ) inside each region and between regions. Data transmissions between instances in the same AZ has no charge. One TB of data incoming from the Internet to EC2 instances costs \$1, while each TB of data transferred from EC2 instances to the Internet is charged \$10. For data transmitted between multiple AZs in the same Region and between Regions, the prices for each TB are \$1 and \$2, respectively.

Cost Comparison: Figure 5 shows costs of Kafka and Kinesis settings for 24 hours of data storage. All costs are calculated based on Amazon Simple Monthly Calculator. The *y*-axis of the graphs shows the monthly total cost in US Dollars of using those services in Northern California region while the *x*-axis indicates different level of throughput in MB/s.

With the Kinesis Stream service, the price of services does not depend only on the throughput of incoming data but the number and the size of records (PUT Payload as described above). Since the average size of records in our experiments is 3 KB, we consider the costs in several scenarios, in which the sizes of records are 1 KB, 3 KB or 10 KB in our analysis. To deal with the scale of incoming throughput, Kinesis Stream has to use more streams to handle the stream

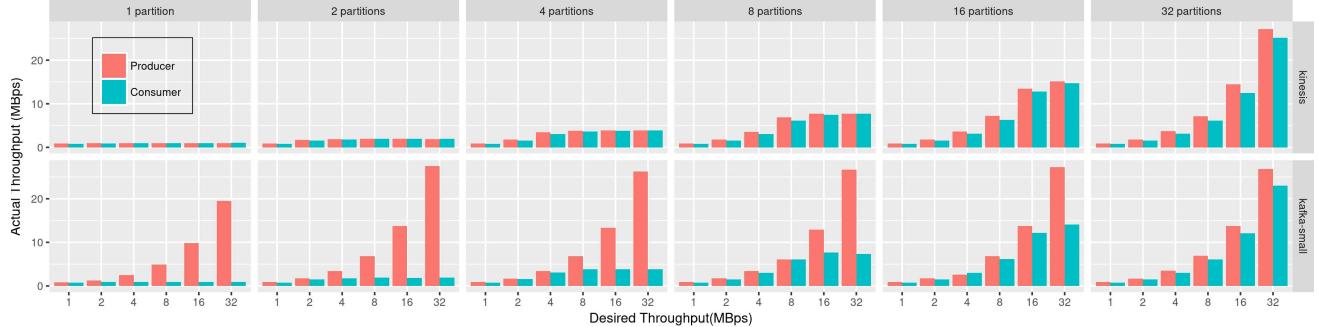


Fig. 4: **Producer/Consumer Throughput Across Different Number of Shards/Partitions for Kinesis and Kafka.** Though there are differences in Kafka and Kinesis producer's behavior with respect to the number of shards/partitions, their consumer's performances and scalability are similar. Both Kafka and Kinesis consumer performances are related closely with the number of shards/partitions.

of data efficiently: each shard in a Kinesis Stream will handle at most 1 MB/s of data. If not, throttling will happen and lead to an unsustainable situation in the producers' buffers. The cost of Kinesis Streams, therefore, largely depends on the number of shards, which scales by the rate of incoming data.

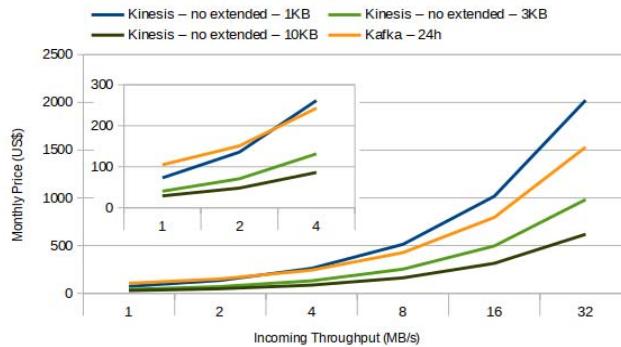


Fig. 5: Cost Comparison. Kafka costs less than Kinesis with small (1KB) messages, but generally costs more with larger (3KB or 10KB) messages. In Kinesis, with the same throughput, larger messages require fewer PUT calls, reducing total cost.

The cost of Kafka, however, grows using a different mechanism. With a throughput of 1 MB/s of incoming data, a Kafka cluster needs a 100 GB of storage for each Kafka instance for 24 hour storage. With a higher level of throughput, larger storage is needed to keep temporary data. Also, deploying Kafka in AWS EC2 instances requires paying the cost for data bandwidth, which also grows with the rate of incoming data.

Figure 5 shows that Kinesis Stream has different prices for different sizes of records, which is more expensive when record size is small, and cheaper if the size of records are large. The price of the Kafka system is lying between Kinesis prices for record sizes of 1KB and 3KB.

IV. DISCUSSION AND CONCLUSIONS

We have evaluated the cloud-native Kinesis and the open-source Kafka along several dimensions of reliability, performance, and cost. We find that both systems are highly reliable for the configurations tested, even under very high load. Kinesis dropped one message in our experiments when the tested load was 32 times higher than its guaranteed level of service. With respect to performance, we find that Kinesis

throttles the Producer client and only accepts messages it can receive into its configured buffers. Kafka does not throttle the Producer client, which places the responsibility on Kafka to provide sufficient buffers to support the stream.

The cost of Kinesis is about four times lower than that of Kafka for the largest message sizes, and provides similar service guarantees. Kafka requires more expertise to configure and to set up in a reliable, fault tolerant way. It provides however a higher degree of customizeability and flexibility. In comparison, Kinesis is much easier to use "out of box". Also, Kinesis is a managed service with very high availability.

In the tested configurations neither Kinesis nor Kafka can dynamically scale to add or subtract shards/partitions as the stream rate changes. Both of these can be set up to scale dynamically with some programming effort. In the future, we will investigate further aspects of cloud-based stream processing, e.g., the ability to process streaming data using scalable machine learning algorithms for abnormal event detection and other types of analyses.

REFERENCES

- [1] J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB)*, Athens, Greece, 2011.
- [2] Amazon kinesis. <https://aws.amazon.com/kinesis/>, 2017.
- [3] Andre Luckow, Peter Kasson, and Shantenu Jha. Pilot-Streaming: Design Considerations for a Stream Processing Framework for High-Performance Computing, March 2016.
- [4] Jay Kreps. Benchmarking apache kafka: 2 million writes per second (on three cheap machines). <https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines>, 2014.
- [5] Yuheng Du, Mashrur Chowdhury, Mizanur Rahman, Kakan Dey, Amy Apon, Andre Luckow, and Linh Bao Ngo. A distributed message delivery infrastructure for connected vehicle technology applications. *IEEE Transactions on Intelligent Transportation Systems*, 2017.
- [6] R. Ranjan. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, 1(1):78–83, May 2014.
- [7] Kartik Paramasivam. How were improving and advancing kafka at linkedin. <https://engineering.linkedin.com/apache-kafka/how-we-re-improving-and-advancing-kafka-linkedin>, 2015.
- [8] Tianning Zhang. Reliable event messaging in big data enterprises: Looking for the balance between producers and consumers. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15*, pages 226–233, New York, NY, USA, 2015. ACM.
- [9] M. Rostanski, K. Grochla, and A. Seman. Evaluation of highly available and fault-tolerant middleware clustered architectures using rabbitmq. In *2014 Federated Conference on Computer Science and Information Systems*, pages 879–884, Sept 2014.