Comparative analysis of real-time messages in big data pipeline architecture

Thandar Aung*, Hla Yin Min and Aung Htein Maw

University of Information Technology,
Parami Road, Universities' Hlaing Campus,
Ward (12), Hlaing Township, Yangon, Myanmar
Email: Thandaraung@uit.edu.mm
Email: hlayinmin@uit.edu.mm
Email: ahmaw@uit.edu.mm
*Corresponding author

Abstract: Nowadays, real-time messaging system is the essential thing in enabling time-critical decision making in many applications where it is important to deal with real-time requirements and reliability requirements simultaneously. For dependability reasons, we intend to maximise the reliability requirement of the real-time messaging system. To develop a real-time messaging system, we create real-time big data pipeline by using Apache Kafka and Apache Storm. This paper focuses on analysing the performance of producer and consumer in Apache Kafka processing. Apache Kafka is the most popular framework used to ingest the data streams into the processing platforms. The comparative analysis of Kafka processing is more efficient to get reliable data on the pipeline architecture. Then, the experiment will be conducted the processing time in the performance of the producer and consumer on various partitions and many servers. The performance analysis of Kafka can impact on messaging systems in real-time big data pipeline architecture.

Keywords: messaging; real-time processing; Apache Kafka; Apache Storm; messaging system; performance analysis; big data pipeline.

Reference to this paper should be made as follows: Aung, T., Min, H.Y. and Maw, A.H. (2019) 'Comparative analysis of real-time messages in big data pipeline architecture', *Int. J. High Performance Computing and Networking*, Vol. 15, Nos. 3/4, pp.191–201.

Biographical notes: Thandar Aung is currently a PhD candidate at University of Information Technology. She received her MCSc in Computer Science from University of Computer Studies (2010). She is one of Faculty of Computer Sciences,University of Computer Studies (MAUBIN). She taught programming languages in undergraduate degree programs and distributed computing in Advanced level. Her research interests lie in the areas of big data analytics, distributed system and real-time processing. She has experience in the computer science, with emphasis on real-time messaging system, checkpointing, Apache Kafka framework and Apache Storm framework. Currently, her research focuses on distributed computing environment on big data analytics.

Hla Yin Min is a Lecturer at the Faculty of Computer Systems and Technologies, University of Information Technology, Yangon, Myanmar. She received her MCTech degree in Computer Technology from University of Computer Studies, Mandalay in 2009 and PhD in Information Technology from University of Technology (Yatanarpon Cyber City), Myanmar in 2014. The work of her Master thesis is 'Financial data security system using symmetric key encryption' and PhD thesis is 'Fault management using cluster-based routing protocol for WSNs'. Her current research interests include cyber security, wireless sensor networks and routing protocols. Her current research interests include cyber security, wireless sensor networks and advanced networking technology and security. She has been cooperating as an instructional designer in ASEAN cyber university (ACU) project.

Aung Htein Maw received his Master of Information Science (MISc) degree from University of Computer Studies, Yangon (UCSY), in 2001, master degree in Engineering Physics (Electronics) from Yangon Technological University (YTU), Myanmar, in 2002, and PhD degree in Information Technology from UCSY, in 2009. He is one of the Professors of the Faculty of Computer Systems and Technologies, University of Information Technology. His research interests include wireless sensor network, virtualisation technology and distributed computing environment. He has published technical papers in these areas, in the conference proceedings and journals like *ACM Computing Survey.* He is a Program Chair of International Conference on Advanced Information Technologies (ICAIT) organised by the University of Information Technology, Yangon, Myanmar. He has been cooperating at Research Collaborator in Asia Connect Project and Subject Matter Expert in ASEAN Cyber University (ACU) project.

This paper is a revised and expanded version of a paper entitled 'Performance evaluation for real-time messaging system in big data pipeline architecture' presented at The 10th International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC 2018), Zhengzhou, China, 18–20 October 2018.

1 Introduction

In the present big data era, the very first challenge is to collect the data as it is a huge amount of data and the second challenge is to analyse it. This analysis typically includes User behaviour data, Application performance tracing, Activity data in the form of logs and Event messages. Processing or analysing the huge amount of data is a challenging task. It requires a new infrastructure and a new way of thinking about the way business and IT industry works. Today, organisations have a huge amount of data and at the same time, they have the need to derive value from it. Due to the volume and velocity characteristics of big data, streaming data processing and storage might require different compression techniques to ensure efficiency and scalability (Atat et al., 2018).

Real-time processing involves continual input, process and output and minimal response time. Real-time processing is used when acting within a very short period of time is significant, so this type of processing allows the organisation/system to act immediately. It guarantees that the computation is done in near-real-time (seconds at most). Fast response time is critical in these examples: bank ATM operations, money transfer systems, aircraft control or security systems. Apache Storm is one of those automated processing techniques for real-time data processing (Simko, 2015).

Real-time processing is a fast and prompt data processing technology that combines data capturing, data processing and data exportation together. Real-time analytics is an iterative process involving multiple tools and systems. It consists of dynamic analysis and reporting, based on data entered into a system less than one minute before the actual time of use (Thein, 2017).In contrast to traditional data analytical systems that collect and periodically process huge-static-volumes of data, streaming analytic systems avoid putting data at rest and process it as it becomes available, thus minimising the time a single data item spends in the processing pipeline (Wingerath et al., 2016). The main purpose of big data real-time processing is to realise an entire system that can process such mesh data in a short time (Yang et al., 2013). Real-time information is continuously getting generated by applications (business, social, or any other type), and this information needs easy ways to be reliable and quickly routed to multiple types of receivers. Most of the time, applications that are producing information and applications that are consuming this information are well apart and inaccessible to each other. This leads to redevelopment of information producers or consumers to provide an integration point between them. Therefore, a mechanism is required for seamless integration of information of producers and consumers to avoid any kind of rewriting of an application at each end.

Real-time usage of these multiple sets of data collected from production systems has become a challenge because of the volume of data collected and processes. Kafka has high throughput, built-in partitioning, replication, and fault tolerance, which makes it a good solution for large scale message processing applications (Garg, 2013).

This paper proposes and evaluates a real-time processing pipeline using the open-source frameworks that can capture a large amount of data from various data sources, process, store, and analyse the large-scale data efficiently. This proposed system evaluates the performance of Apache Kafka processing. The performance impacts on Apache Kafka to develop real-time pipeline architecture.

The remainder of this paper is organised as follows: Section 2 reviews the related work of this paper. Section 3 gives the system architecture to develop real-time messaging system. Section 4 describes the architecture of Kafka, the zookeeper which needs to run Kafka. Section 5 describes experimental set up of the system and experimental results for large messaging system. Section 6 describes conclusion and future work.

2 Related work

Thein (2017) has proposed to enforce security policies to protect sensitive data in big data security. The author has proposed to provide the secure big data pipeline architecture for the scalability and security. The author used sticky policies and AES algorithm for secure big data pipeline for real-time streaming applications. But their pipeline architecture has the weakness of fault tolerance in real-time data processing in Kafka.

Atat et al. (2018) provided the improvement of big data challenges and the effectiveness of real-time analytics in any application area. The authors point out that real-time analysis is an approach to produce useful information from massive raw data. The authors indicate many application domains such as healthcare, transportation systems, environmental monitoring, and smart cities will require real-time decision making and control.

Kreps et al. (2015) show that Kafka is a scalable message system lie between the message producers and consumers It describes a consumer is much easier to support in the pull model than the push model. If the consumer crashes, the unflushed data is lost. In this case, the consumer take checkpoint the smallest offset of the unflushed messages and re-consume from that offset when it's restarted. However, their contribution can't solve lost messages by using build-in replication for fault tolerance.

Nazeer et al. (2017) have proposed to evaluate real-time text processing pipeline using open-source big data tools. The authors minimise the latency to process data streams and conduct several experiments to determine the performance and scalability of the system against different cluster size and workload generator. They need to improve data stream in real-time processing.

Panda and Naik (2018) have described how data replication plays an important role in distributed systems. The authors proposed various data replication Algorithms to manage the redundancy of data. They tested on the existing Algorithm in four different datasets.

The authors describe that Kafka achieves much higher throughput than conventional messaging system.

Dobbelaere and Esmaili (2017) developed a common framework depend on common features of Apache Kafka and RabbitMQ. The authors compare the throughput and latency based on delivery guarantees of two systems. The authors prove that increasing the Kafka partition count on same node can significantly improve its performance.

Wei et al. (2019) introduced the popular reliability problem of the distributed stream processing system. The authors focused on a novel upstream backup mechanism to solve these reliability problems in DSPSs. The authors point out the checkpoint mechanism can solve high recovery instead of existing replication processes.

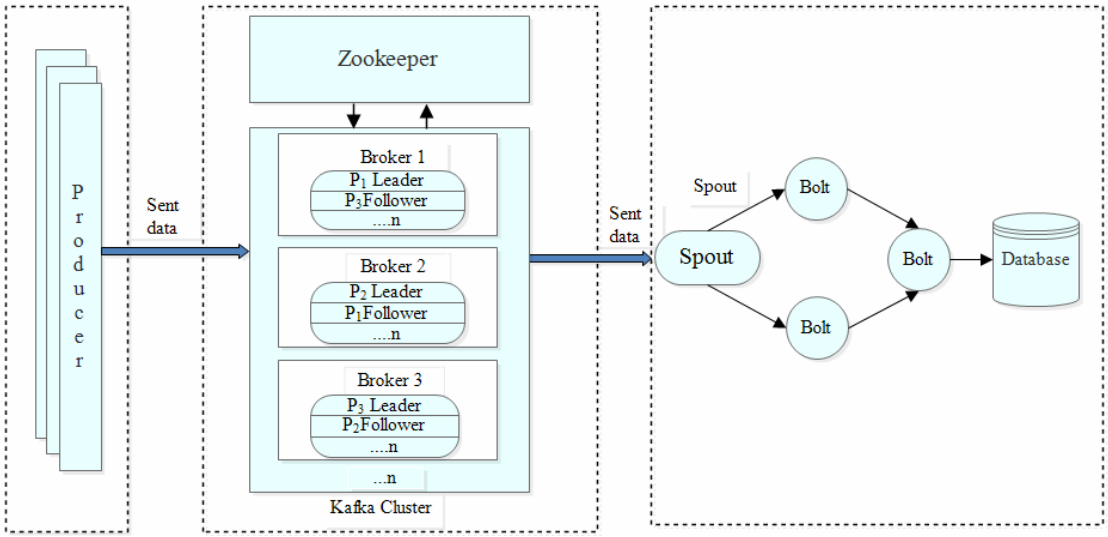
Overall, Kafka has weak guarantees as a distributed messaging system. The weak point of related papers is to develop the performance of processing in the pipeline. In the system, we emphasised the comparative analysis of producer and consumer performance of Apache Kafka. And then, the system reveals the weak point of performance evaluation of replication process. The results show that higher performance can lead to significant performance improvements of the real-time messaging system.

3 System architecture for real-time messaging system

This section focuses the performance of producer and consumer in Apache Kafka processing. The performance of Kafka processing modify to be more reliable on the pipeline architecture in Figure 1. Table 1 shows the processing steps for developing real-time big data pipeline architecture. The system describes the analysis of Kafka processing on various partitions and many servers in Algorithm 1. Although it can be high complexity of Kafka process, the processing time is faster. More servers and partitions can improve the performance of Kafka processing. The increasing Kafka partition count on the same node can significantly improve its performance. The comparative performance of the successful and failed process point out the weak point of fault tolerance in Kafka processing.

In Apache Kafka, producer sends messages by using asynchronous type of producer. Brokers can divide messages into many partitions. Each partition is optionally replicated across a configurable number of servers for fault tolerance. In Kafka, producers and consumers work on the traditional push-and-pull model, where producers push the message to a Kafka broker and consumers pull the message from the broker. Kafka is a high-performance publisher-subscriber-based messaging system. Kafka can act as a buffer or feeder for messages that need to be processed by Storm. The Kafka spout uses the same Zookeeper instance that is used by Apache Storm, to store the states of the message offset and segment consumption tracking if it is consumed. Kafka and Storm naturally complement each other, and their powerful cooperation enables real-time streaming analytics for fast-moving big data.

Figure 1 Real-time big data pipeline architecture (see online version for colours)



3.1 Zookeeper

Zookeeper is a centralised service for maintaining configuration information, naming, providing distributed synchronisation, and providing group services. Zookeeper is also a high-performance coordination service for distributed applications (Molnar, 2019). Each time they are implemented, there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. When it works correctly, different implementations of these services lead to management complexity when the applications are deployed. The service itself is distributed and highly reliable.

Kafka uses Zookeeper for the following tasks: detecting the addition and the removal of brokers and consumers. Triggering a rebalance process in each consumer when the above events happen, and maintaining the consumption relationship and keeping track of the consumed offset of each partition. Specifically, when each broker or consumer starts up, it stores its information in a broker or consumer registry in Zookeeper. The broker registry contains the broker's host name and port, and the set of topics and the partitions stored on it.

3.2 Apache Storm

Storm (Jain and Nalya, 2014) is also an open source, distributed, reliable, and fault tolerant system for processing streams of large volumes of data in real-time. It supports many use cases, such as real-time analytics, online machine learning, continuous computation, and the extract transformation load (ETL) paradigm. Storm can be used stream processing, continuous computation, distributed RPC, real-time analytics.

A Storm cluster follows a master-slave model where the master and slave processes are coordinated through Zookeeper. The Storm cluster is made up of a main node and several working nodes (Yang et al., 2013).

- Nimbus

  The Nimbus node is the master in a Storm cluster. A daemon process called 'Nimbus' is running on main node, in order to allocate codes, arrange tasks and detect errors.

- Supervisor

  Supervisor nodes are the worker nodes in a Storm cluster. Each working node has a daemon process called 'Supervisor' to monitor, start and stop working process. The coordination work between Nimbus and Supervisor is handled by 'Zookeeper' as shown in Figure 2. Zookeeper is the subproject of Hadoop and it aims at coordinates works in large-scale distribution system. The Storm cluster is similar with Hadoop, where Nimbus corresponds to job tracker, and the Supervisors correspond to task trackers.

Figure 2 Storm cluster's architecture

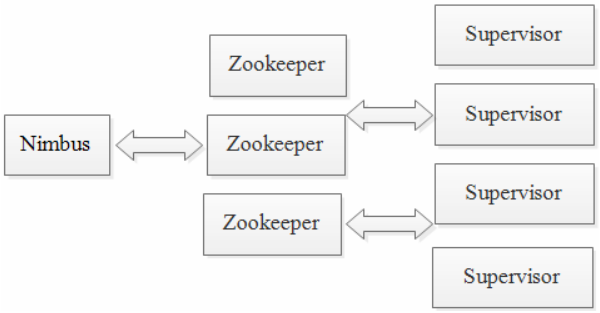
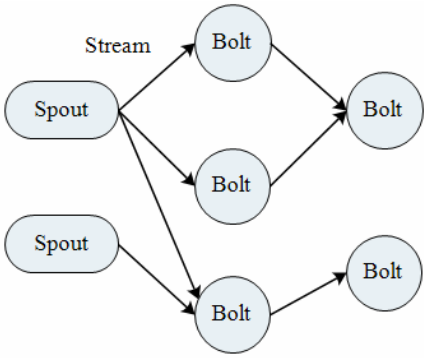


Figure 3 Storm topology (see online version for colours)



In Storm terminology (Jain and Nalya, 2014), a topology is an abstraction that defines the graph of the computation. There is an example of one topology in Figure 3. A topology can be represented by a direct acyclic graph, where each node does some kind of processing and forwards it to the next node(s) in the flow. The followings are the components of a Storm topology:

- *Stream:* a stream is an unbounded sequence of tuples that can be processed in parallel by Storm. Each stream can be processed by a single or multiple types of bolts.
- *Spout:* a spout is the source of tuples in a Storm topology. Spout is the input stream source which can read from external data source (Cai and Jin, 2015). For example, by reading from a log file or listening for new messages in a queue and publishing them-emitting, in Storm terminology-into streams.
- *Bolt:* the spout passes the data to a component called bolt. Bolts (Cai and Jin, 2015) are processor units which can process any number of streams and produce output streams. A bolt is responsible for transforming a stream. Each bolt in the topology should be doing a simple transformation of the tuples, and such bolts can coordinate with each other to exhibit a complex transformation.

4 Apache Kafka architecture

Kafka (Garg, 2013) is an open source, distributed publish subscribe messaging system. Kafka, which provides a real-time publish-subscribe solution for overcoming the

challenges of consuming the real-time and batch data volumes that may grow in order of magnitude to be larger than the real data.

Apache Kafka is a real-time, fault tolerant, scalable messaging system for moving data in real-time. Kafka maintains feeds of messages in categories called topics. We'll call processes that publish messages to a Kafka topic are producers. And we'll call processes that subscribe to topics and process the feed of published messages are consumers. Kafka is run as a cluster comprised of one or more servers, each of which is called a broker. Producers send messages over the network to the Kafka cluster, which in turn serves them up to consumers. A producer publishes messages to a Kafka topic. Kafka topic is also considered as a message category or feed name to which messages are published. Kafka topics are created on a Kafka broker acting as a Kafka server. Processes that subscribe to topics and process the feed of published messages are called consumers. Brokers and consumers use Zookeeper to get the state information and to track messages offsets, respectively.

All the message partitions are assigned a unique sequential number called the offset, which is used to identify each message within the partition. Each partition is optionally replicated across a configurable number of servers for fault tolerance. Each partition available on either of the servers acts as the leader and has zero or more servers acting as followers. Here the leader is responsible for handling all read and write requests for the partition while the followers asynchronously replicate data from the leader. Kafka dynamically maintains a set of in-sync replicas (ISR) that is caught-up to the leader and always persist the latest ISR set to Zookeeper. In a Kafka cluster, each server plays a dual role; it acts as a leader for some of its partitions and also a follower for other partitions. If any of the follower in-sync replicas fail, the leader drops the failed follower from its ISR list. After the configured timeout period will continue on the remaining replicas in ISRs. Whenever the failed follower comes back, it truncates its log to the last checkpoint and then starts to catch up with all messages from the leader, starting from the checkpoint. As soon as the follower becomes fully synced with the leader, the leader adds it back to the current ISR list.

If the leader fails, the process of choosing the new lead replica involves all the followers' ISRs registering themselves with Zookeeper. The very first registered replica becomes the new lead replica and its log end offset (LEO) becomes the replicas offset of the last committed. The rest of the registered replicas become the followers of the newly elected leader. Kafka provides between producer and consumer. There are multiple possible ways to deliver messages, such as: messages are never redelivered but may be lost, messages may be redelivered but never lost, and messages are delivered once and only once.

Table 1 The algorithm on Kafka processing in pipeline architecture

Input: *Real-time messages*

Output: *Performance of Kafka process, number of lost messages*

Begin

1 Initialise Zookeeper server for processing

2 Initialise Kafka local server to define broker id, port and log dir

3 Create a topic to define replication factors and partitions

4 Publish messages by asynchronous type to Kafka cluster

Start

a Divide partitions by round-robin policy

b Select a leader randomly from portion of the partitions

c Identify offset and execute the replication process.

d Check leader and replicate followers in ISR (in-synchronous replica) list

e Store replica in zookeeper

End

5 Repeat Step 4 on this process until the termination criterion

6 Zookeeper handles between topic and consumer

7 Get process id and kill leader in one server

8 **if** Server = fail **then**

a Process failure recovery depend on the leader or follower

9 Check total messages in processing by using GetOffsetShell tools

10 Evaluate the performance by using performance tools of Kafka

a Total consumer messages per bytes (MB)

b Average consumed messages bytes per second

c Total and average messages count per second

11 Compare the performance of successful processes depend on various partitions and many servers

12 Compare the performance of failed processes depend on various partitions and many servers

14 Compare the lost messages based on various partitions and many servers.

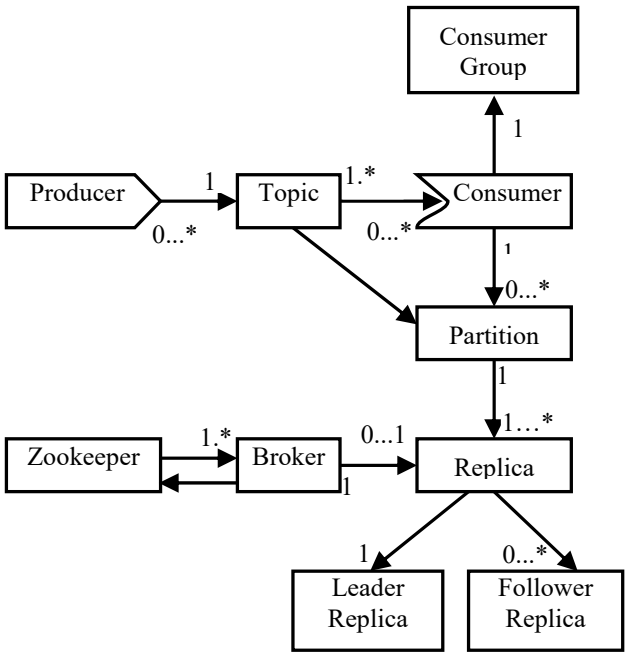
13 Consume the messages and run Kafka-Storm Integration pipeline.

End

Figure 4 describes the processing of apache Kafka. Firstly, a producer sends a message to one topic at a time. A topic has 0 or more producers. A consumer subscribes to one or more topics. A topic has zero or more consumers. A consumer is a member of one consumer group. Zookeeper serves as the coordination interface between the Kafka broker in topic and consumers. A partition has one consumer per group. A consumer pulls messages from 0 or more partitions per

topic. A topic is replicated over one or more partitions. A partition has one or more replicas. A replica is on one broker. A broker has also zero or one replicas per partitions. A partition has one leader and zero or more followers.

Figure 4 Kafka processing



5 Experimental setup and results

The experimental setup is performed by using two open source frameworks Apache Kafka 2.11-0.9.0.0 and Apache Storm 0.9.7 as the main pipeline architecture. Java 1.8 is running on underlying pipeline architecture. The Apache Maven 3.5.0 is used Kafka-Storm integration. The real-time data for experiments are mobile phone spam messages from the Grumble text website. In Table 1, the performance of the algorithm is evaluated for the real-time process in Apache Kafka. The implementation of the system is based on the hardware specification shown in Table 2.

Table 2 Hardware specification

Operating system	Windows 32-bit operating system
RAM	4.00 GB
Hard-disk	1 TB
Processor	Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz

There are eight different experiments that are conducted to evaluate the performance over Kafka processing.

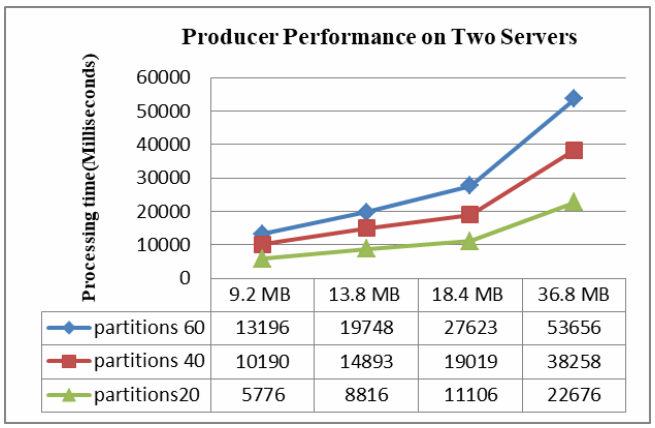
Table 3 Summary of experiments

Experiment		*Description*
1	Producer performance in successful process	Deployed Apache Kafka on two servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in successful process
2	Consumer performance in successful process	Deployed Apache Kafka on two servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in successful process
3	Producer performance in failed process	Deployed Apache Kafka on two servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in failed process
4	Consumer performance in failed process	Deployed Apache Kafka on two servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in failed process
5	Producer performance in successful process	Deployed Apache Kafka on three servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in successful process
6	Consumer performance in successful process	Deployed Apache Kafka on three servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in successful process
7	Producer performance in failed process	Deployed Apache Kafka on three servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in failed process
8	Consumer performance in failed process	Deployed Apache Kafka on three servers, 20 partitions, 40 partitions and 60 partitions on different types of dataset with five batch size in failed process

- Experiment 1: Producer performance of successful process on two servers

  Figure 5 shows a single producer to publish various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. It configured the Kafka producer to send messages asynchronously by five batches size. Figure 5 shows the result. The x-axis represents the amount of data sent to the broker, and the y-axis corresponds to the total time in producer performance. It compares the total times on the difference between successful processes with three types of partitions in various data sizes on two servers. We consider this experiment as total time by calculating based on producer performance tools in Apache Kafka. Data size is directly proportional to the processing time in producer performance on two servers. Processing time is inversely proportional to the number of messages per second. This experiment can handle the producer performance on two servers.

Figure 5 Successful processes of producer performance on two servers (see online version for colours)

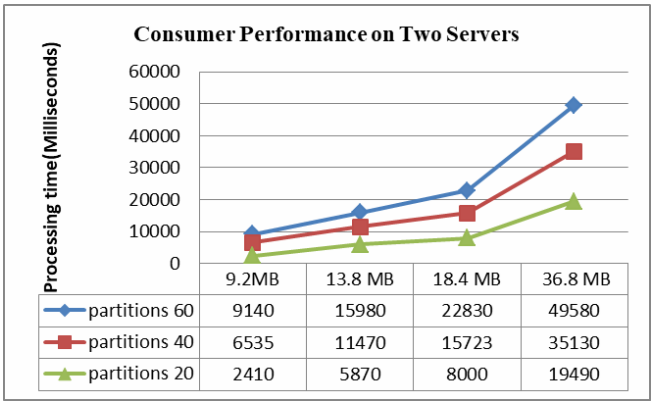


	9.2 MB	13.8 MB	18.4 MB	36.8 MB
partitions 60	13196	19748	27623	53656
partitions 40	10190	14893	19019	38258
partitions20	5776	8816	11106	22676

- Experiment 2: Consumer performance of successful process on two servers

  Figure 6 shows a single consumer to consume various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, and 36.8 MB respectively. It configured the Kafka consumer to consume messages asynchronously by five batches size. Figure 6 shows the result. The x-axis represents the amount of data sent to the broker, and the y-axis corresponds to the total time in consumer performance in the success process. It compares the total times on the difference between failed processes with many partitions in various data sizes on two servers. The experiment emphasises the total time by calculating based on consumer performance tools in Apache Kafka. The comparison of Figures 5 and 6 is different. The consumer can consume more messages in one second. Processing time is inversely proportional to the number of messages per second. Overall total time in this experiment is different in experiment 1. Both experiments 1 and 2 can handle the processing in successful process.

Figure 6 Successful processes of consumer performance on two servers (see online version for colours)

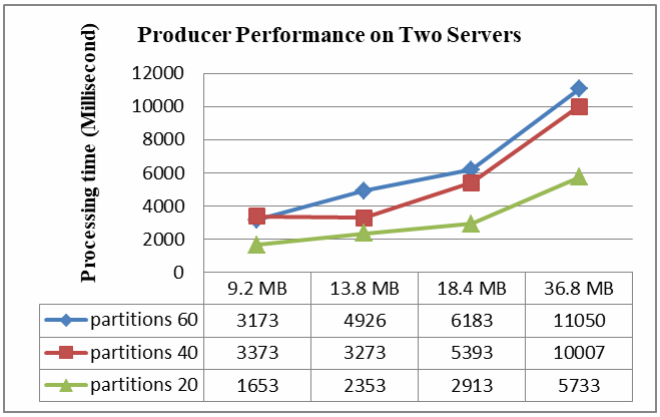


	9.2MB	13.8 MB	18.4 MB	36.8 MB
partitions 60	9140	15980	22830	49580
partitions 40	6535	11470	15723	35130
partitions 20	2410	5870	8000	19490

- Experiment 3: Producer performance of failed process on two servers

  Figure 7 shows a single producer to publish various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. It configured the Kafka producer to send messages asynchronously by five batches size. Figure 7 shows the result. The x-axis represents the amount of data sent to the broker, and the y-axis corresponds to the total time in producer performance. It compares the total times on the difference between failed processes with many partitions in various data sizes on two servers. The experiment emphasises the total time by calculating based on producer performance tools in Apache Kafka. The comparison of Figures 5 and 7 is slightly different. In experiment 3, the total time of failed processes cannot handle between processing time and the number of messages per second.

Figure 7 Failed processes of producer performance on two servers (see online version for colours)



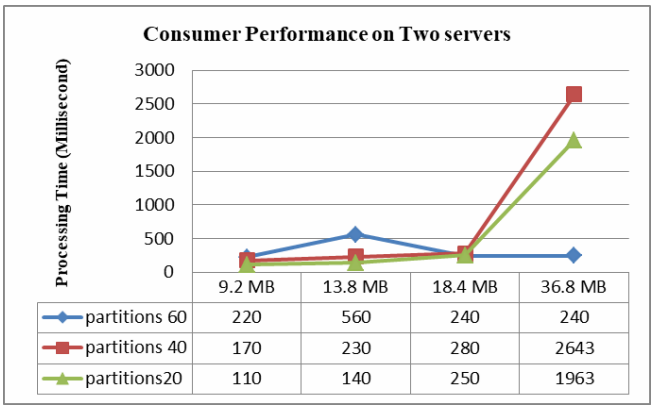
	9.2 MB	13.8 MB	18.4 MB	36.8 MB
partitions 60	3173	4926	6183	11050
partitions 40	3373	3273	5393	10007
partitions 20	1653	2353	2913	5733

- Experiment 4: Consumer performance of failed process on two servers

  Figure 8 shows a single consumer to consume various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. It configured the Kafka consumer to receive messages asynchronously by five batches size. Figure 8 shows the result. The x-axis represents the amount of data sent to the broker, and the y-axis corresponds to the total time in consumer performance. It compares the total times on the difference between successful and failed processes with many partitions in various data sizes on two servers. The experiment emphasises the total time by calculating based on consumer performance tools in Apache Kafka. In Figure 8, the failed processing time of most of the dataset is decreased than the producer performance time. More dataset is directly proportional to lost messages. The consumer consumes more messages in one second

than successful process. When we tested more data size on two servers, we need to handle consumer performance. The performance of processes between Figures 6 and 8 is significantly different.

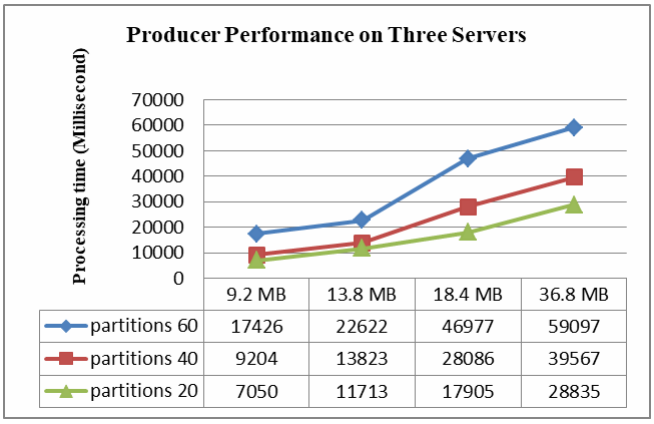
Figure 8 Failed processes of consumer performance on two servers (see online version for colours)



- Experiment 5: Producer performance of successful process on three servers

  Figure 9 shows a single producer to publish various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. It compares the total times on the difference between successful processes with three types of partitions in various data sizes on two servers. Data size is directly proportional to the processing time in producer performance on three servers. Processing time is inversely proportional to the number of messages per second. The producer performance of three servers is more increased than two servers.

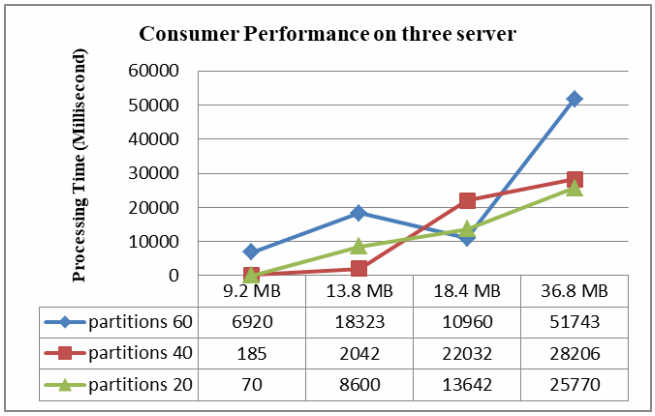
Figure 9 Successful processes of producer performance on three servers (see online version for colours)



- Experiment 6: Consumer performance of successful process on three servers

  Figure 10 shows a single consumer to consume various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. It compares the total times on the difference between failed processes with many partitions in various data sizes on two servers. The comparison of Figures 9 and 10 is different. In 18.4 MB, the consumer performance time cannot handle among many partitions.

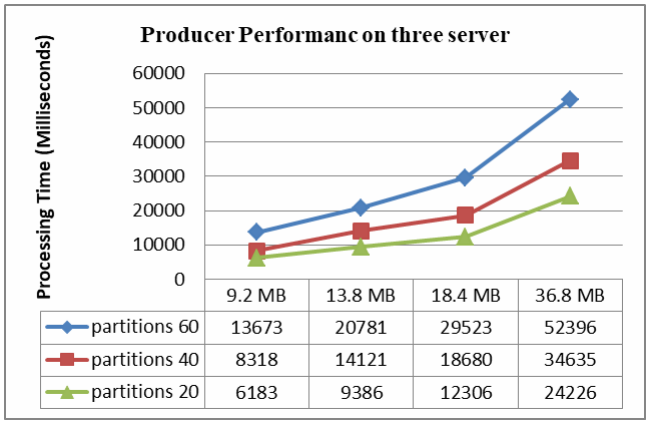
Figure 10 Successful processes of consumer performance on three servers (see online version for colours)



- Experiment 7: Producer Performance of failed process on three servers

  Figure 11 shows a single producer to publish various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. It configured the Kafka producer to send messages asynchronously by five batches size. The experiment emphasises the total time by calculating based on producer performance tools in Apache Kafka. When we tested on three servers, the system is better than testing on two servers. The performance of experiment 7 is higher than the performance of experiment 3.

Figure 11 Failed processes of producer performance on three servers (see online version for colours)

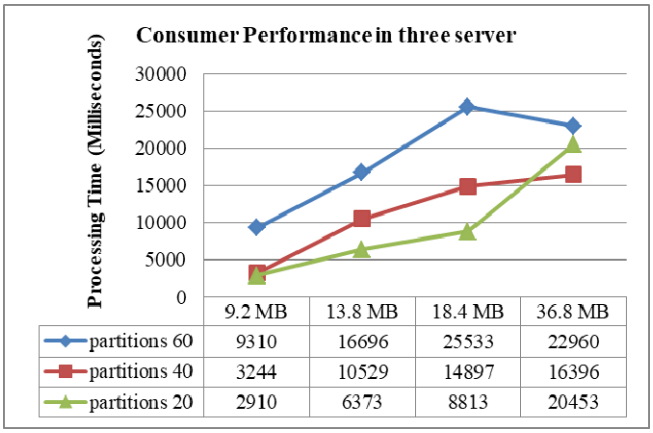


- Experiment 8: Consumer performance of failed process on three servers

  Figure 12 shows a single consumer to consume various data sizes of messages 9.2 MB, 13.9 MB, 18.4 MB, 36.8 MB respectively. The experiment emphasizes the total time by calculating based on consumer performance tools in Apache Kafka. In Figure 12, the failed processing time of most of the dataset is significantly increased than consumer performance time of two servers in Figure 8. More dataset is directly

proportional to losing messages. The system spends more rescheduling time for failed processes. When we tested more data size on three servers, we need to handle consumer performance.

Figure 12 Failed processes of consumer performance on three servers (see online version for colours)



The system was tested on many partitions and servers on the different number of datasets. We used performance tools in Apache Kafka to measure the performance of producer and consumer. The producer and consumer performance of experiments, calculate based on equations (1) to (4). The proposed system uses to input commands for measuring producer performance. There are broker lists, number of messages and topic name. Producer performance commands show the result at the end of the performance testing. There is the start time of the test, the end time of the test, compression, message size, and batch size, total consumed messages bytes (MB), average consumed messages bytes per second, total consumed messages count, average consumed messages count per second. The system uses Zookeeper host, number of messages and topic as an input command for measuring consumer performance. The information included in the command output for consumer performance. There is the start time of the test, the end time of the test, total consumed messages bytes (MB), average consumed messages bytes per second, total consumed messages count, average consumed messages count per second.

In (1), calculate the total consumed message bytes (MB) represent by α, β represents total bytes sent and memory size is 1,024 ∗ 1,024.

$$\alpha = \frac{(\beta * 1.0)}{(1{,}024 * 1{,}024)} \tag{1}$$

In (2), calculate the average consumed MB bytes per second represent by α; β represents total messages sent and elapsed seconds.

$$\alpha = \frac{\beta}{\theta} \tag{2}$$

In (3), calculate the elapsed time (α), the end time (milliseconds) is β and the start time (milliseconds) is £.

$$\alpha = \frac{(\beta - £)}{1{,}000.0} \tag{3}$$

In (4), the calculation of the average consumed message bytes per second (α); β represents total messages sent and £ represents elapsed seconds.

$$\alpha = \frac{\beta}{£} \tag{4}$$

Table 4 Experimental summary for producer performance

Dataset	*Number of servers*	*Success/fail*	*Number of messages per second Partitions 20*	*Number of messages per second Partitions 40*	*Number of messages per second Partitions 60*
9.2 MB	Two servers	Successful	19,299.8	10,940.2	8,447.7
		Failed	67,438.5	33,050.9	35,132.6
	Three servers	Successful	15,812.1	12,112.2	6,397.1
		Failed	18,029.4	13,402.4	8,153
13.8 MB	Two servers	Successful	18,967.3	11,227.8	8,467.4
		Failed	71,065	51,089.5	33,945.5
	Three servers	Successful	14,276.1	12,096.9	7,391.7
		Failed	17,815.4	15,497.3	8,406.5
18.4MB	Two servers	Successful	20,075.7	13,194.2	8,071.5
		Failed	76,539.9	41,341.7	36,060.3
	Three servers	Successful	12,452.4	11,095.5	4,839.2
		Failed	18,118	11,936.1	7,552.1
36.8 MB	Two servers	Successful	19,664.8	14,596.6	8,310.7
		Failed	77,781.4	44,561.4	40,354.8
	Three servers	Successful	15,464.5	12,163.4	7,545.5
		Failed	18,406.7	12,874.9	8,510.5

Table 5 Experimental summary for consumer performance

Dataset	*Number of servers*	*Success/fail*	*Number of messages per second Partitions 20*	*Number of messages per second Partitions 40*	*Number of messages per second Partitions 60*
9.2 MB	Two servers	Successful	92,510.7	34,117.9	24,392.8
		Failed	2,001,845	1,287,541	994,340.9
	Three servers	Successful	1,592,500	1,205,194.6	32,218.3
		Failed	75,976.6	68,171.4	23,772.6
13.8 MB	Two servers	Successful	56,972.9	29,157.1	20,928
		Failed	2,369,057	1,442,182.6	591,503.5
	Three servers	Successful	38,887.3	163,776.2	18,251.9
		Failed	52,169.4	2,374,692.9	19,929.9
18.4MB	Two servers	Successful	55,750.1	102,626.2	19,532.2
		Failed	1,772,524	1,587,103.5	1,846,037.5
	Three servers	Successful	32,687.3	35,514.5	29,085
		Failed	50,387.2	29,775.3	17,355.9
36.8 MB	Two servers	Successful	45,758.9	1,715,096.2	17,987.9
		Failed	17,987.9	244,064.5	1,846,891.6
	Three servers	Successful	34,607.7	1,922,068.9	17,574.9
		Failed	43,510.6	54,285.2	38,762.1

Table 4 summarises experimental results. The table compares the number of messages per one second in many partitions between two servers and three servers. The results indicate that the number of messages per second on three types of partitions is different. The number of messages tested on three servers is slightly different than two servers. The processing time of Figures 5, 7, 9 and 11 correlate to the number of messages per second in Table 3. The experimental results calculated based on equations (1) to (4). According to the result, producer performance in many partitions on three servers improves than two servers.

Table 5 summarises experimental results. The table shows the number of messages per one second on two and three servers respectively. The results indicate that the consumer performance can't handle the messages and processing time. The numbers of messages are significantly different on the two types of servers. Some consumer consumes more messages in one second and some consumes fewer messages. The processing time of Figures 6, 8, 10 and 12 correlate to the number of messages per second in Table 4. The experimental results calculated based on equations (1) to (4).

In Table 6 summaries the amount of lost messages in processing. The system describes by testing various partitions on more servers. Failure occurs in more partitions on many servers, the higher the lost messages. According to the experiments, the system needs to control the performance on many partitions and many servers. We need to reduce completion time and recover lost messages.

Table 6 Number of lost messages in testing many servers

Dataset	*Number of servers*	*Partitions 20*	*Partitions 40*	*Partitions 60*
9.2 MB	Two servers	2,748	2,863	4,196
	Three servers	1,859	1,808	1,628
13.8 MB	Two servers	2,763	2,729	3,189
	Three servers	1,955	1,974	1,680
18.4MB	Two servers	2,790	2,787	2,872
	Three servers	1,858	2,362	2,772
36.8 MB	Two servers	2,574	2,719	2,666
	Three servers	1,917	1,890	1,863

In the future, message logging-based checkpointing will be used to solve the problem of lost messages and to reduce the completion time of the system. It is used to provide fault tolerance in distributed systems in which all inter-process communication is through messages. Message-logging protocols guarantee that upon recovery, no process is an orphan. This requirement can be enforced either by avoiding the creation of orphans during an execution. Checkpointing is utilised to limit log size and recovery latency.

6 Conclusions

Regarding the analysis of the producer and consumer performance in a real-time messaging system, innovation in business and industry works could be used to help. Relying

on the performance analysis, we proposed to develop the reliability of real-time big data pipeline architecture. We measured the performance of big data pipeline architecture by comparing the successful and failed process on different partitions and servers. We published the messages in synchronous mode and then analysed the performance by using the producer and consumer performance tools. According to Figures 5, 7, 9 and 11, increasing servers can achieve a better producer performance in Kafka processing. Moreover, through Figures 6, 8, 10 and 12 we indicate the improvement of the consumer performance. On the other hand, we pointed out the issue of consumer performance because of losing messages in big data pipeline architecture. Conforming to the experimental results, increasing partitions and servers can lead to significant performance improvements. Furthermore, these experimental results pointed out the issue of load balancing between processing time and the number of messages per second in failed process. We proved the improvement of the Kafka processing by comparing the producer and consumer performance, but the system must promote reliability more and more.

As future research, we will perform to promote reliability by handling the fault tolerance in Apache Kafka. A future proposal of this work will be a presentation of innovative and optimised algorithms which improve the reliability of real-time big data pipeline architecture. We would further examine the replication methods and replicate the message logging-based checkpointing. This could lead to recover lost messages and reduce the completion time in failure case in our pipeline architecture. This can be the field of future research.

References

Atat, R., Liu, L., Wu, J., Li, G., Ye, C. and Yi, Y. (2018) 'Big data meet cyber-physical systems: a panoramic survey', *IEEE Access*, November, Vol. 6, pp.73603–73636.

Cai, J. and Jin, Z. (2015) 'Real-time calculating over self-health data using Storm', *4th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering (ICMMCCE 2015)*.

Dobbelaere, P. and Esmaili, K.S. (2017) 'Industry paper', *11th ACM International Conference on Distributed and Event-based Systems*, June.

Garg, N. (2013) *Apache Kafka*, October, PACKT Publishing, UK.

Jain, A. and Nalya, A. (2014) *Learning Storm*, August, PACKT Publishing, UK.

Kreps, J., Narkhede, N. and Rao, J. (2015) Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB , June,Vol. 11, pp. 1-7.

Molnar, A. (20149) http://zookeeper.apache.org (accessed 14 August 2019).

Nazeer, H., Iqbal, W., Bokhari, F., Bukhari, F. and Baig, S.U.R. (2017) *Real-Time Text Analytics Pipeline Using Open-Source Big Data Tools*, arXiv preprint arXiv:1712.04344, December.

Panda, S.K. and Naik, S. (2018) 'An efficient data replication algorithm for distributed systems', *International Journal of Cloud Applications and Computing*, July–September, Vol. 8, No. 3.

Simko, S. (2015) *Performance Testing of Apache Storm Framework*, Masaryk University, Faculty of Informatics, Brno, Autumn.

Thein, K.M.M. (2017) 'Security of real-time big data analytics pipeline', *International Journal of Advances in Electronics and Computer Sciences*, February, Vol. 2, No. 2, pp.1–5.

Wei, X., Zhuang, Y., Li, H. and Liu, Z. (2019) 'Reliable stream data processing for elastic distributed stream processing systems', *Cluster Comput.* [online] https://doi.org/10.1007/s10586-019-02939-9.

Wingerath, W., Gessert, F., Friedrich, S. and Ritter, N. (2016) 'Real-time stream processing for big data', *it-Information Technology*, Vol. 58, No. 4, pp.186–194.

Yang, W., Liu, X. and Zhang, L. (2013) 'Big data real-time processing based on Storm', *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*.