Evaluation of Apache Kafka in Real-Time Big Data Pipeline Architecture

Thandar Aung, Hla Yin Min, Aung Htein Maw
University of Information Technology
Yangon, Myanmar
thandaraung@uit.edu.mm, hlayinmin@uit.edu.mm, ahmaw@uit.edu.mm

Abstract

Today, many applications based on real-time analytics need to enable time-critical decision with real-time requirements and process with reliability requirements. Modern distributed systems are growing exponentially as far as performance and scale. The main purpose of Big Data real-time processing is to realize an entire system that can process such mesh data in a short time. And the performance of processing time can be guaranteed in a satisfactory range. To develop a distributed data pipeline, the system proposes real-time big data pipeline by using Apache Kafka and Apache Storm. Apache Kafka is currently the most popular framework used to ingest the data streams into the processing platforms. However, there are many challenges how to send reliable messages on many servers. This paper focuses on the comparison of the processing time between successful processes and failed processes on many servers. The experimental results show the performance impact on both producer and consumer of the Apache Kafka framework.

Keywords- Apache Kafka, Apache storm, Asynchronous replication, Real time processing,

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 analyze it. This analysis typically includes User behavior data, Application performance tracking, Activity data in the form of logs and Event messages. Processing or analyzing the huge amount of data is a challenging task. It requires a new infrastructure and a new way of thinking about the business and IT industry works. Today, organizations have a huge amount of data and at the same time, they have the need to derive value from it.

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 organization or system to act immediately. Fast response time is critical in these examples: bank ATM operations, money transfer systems, aircraft control or security systems. Real-time processing is fast and prompt data processing technology that combines data capture, 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. 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. This leads to the 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.

This paper has proposed 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 analyze the large-scale data efficiently. This proposed system evaluates the performance impact on Apache Kafka to develop big data pipeline architecture by using Apache Kafka and Apache Storm. We performed several experiments to prowle the performance testing of the system under different servers.

The remainder of this paper is organized as follows: Section 2 reviews the related work of this paper. Section 3presents the system architecture to develop a real-time messaging system. Section 4 describes the experimental setup of the system and experimental results for the large messaging system in big data pipeline architecture. Section 5 describes the conclusion and future work.

2. Related Work

Seema Balhara, Kavita Khanna [1] has shown that Kafka has superior performance when compared to two popular messaging systems. The author uses ActiveMQ and RabbitMQ. The author describes that Kafka achieves much higher throughput than the conventional messaging system.

Hassan Nazeer, Waheed Iqbal, Fawaz Bokhari[2] Faisal Bukhari, Shuja Ur Rehman Baig has proposed to evaluate

their proposed real-time text processing pipeline using open-source big data tools. This paper intends to minimize 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.

Martin Kleppmann [3] explains the reasoning behind the design of Kafka and Samza, which allow complex applications to be built by composing a small number of simple primitives – replicated logs and stream operators. These draw parallels between the design of Kafka and Samza, batch processing pipelines, database architecture and design philosophy of UNIX.

Muhammad Syafrudin [8]explains capable of processing a massive sensor data efficiently when the number of sensor data and devices increases. This paper is expected to support the management in their decision-making for product quality inspection and support manufacturing sustainability. The OSRDP used several open source-based big data processing such as Apache Kafka, Apache Storm and MongoDB.The results showed that the proposed system is capable of processing a massive sensor data effeciently when the number of sensors data and devices increases.

Paul Le Noac'h, Alexandru Costan, Luc Boug´e[4]has proposed the evaluation of several configurations and performance metrics of Kafka and studied how the ingestion performance can impact the overall stream processing.

Wenjie Yang, Xingang Liu and Lan Zhang [5] have also proposed to ensure the practical applicability and high efficiency, to show acceptable performance in the simulation. The paper showed an entire system RabbitMQ, NoSQL and JSP are proposed based on Storm, which is a novel distributed real-time computing system. The paper organized a big data real-time processing system based on Strom and other tools.

Mohit Maske, Dr. Prakash Prasad, International Journal of Advanced [6] intends to ensure the practical and high efficiency in the simulation system that is established and shown acceptable performance in various expressions using data sheet. It proved that data analysis system for stream and real-time processing based on storm can be used in the various computing environments.

Overall, Kafka has weak guarantees as a distributed messaging system. There's no ordering guarantee when the messages are coming from different partitions. The weak point of related papers is to develop the performance of processing in the pipeline. This paper has thoroughly evaluated the performance of producer and consumer in Kafka. 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 on the pipeline architecture. The performance of Kafka processing modifies to be more reliable on the pipeline architecture in Figure 1.

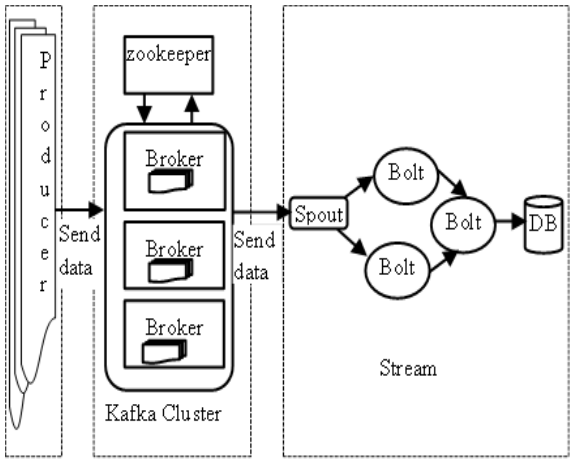

Figure 1. Real-time big data pipeline Architecture

A producer publishes messages to a Kafka topic by using asynchronous type. Brokers can divide messages into many partitions. Zookeeper serves as the coordination interface between the Kafka broker in topic and consumers. 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 can act as a buffer or feeder for messages that need to be processed by Storm. Kafka and Storm naturally complement each other, and their powerful cooperation enables real-time streaming analytics for fast-moving big data. The paper intends to evaluate the performance of Apache Kafka in the pipeline architecture.

3.1 Apache Kafka Architecture

Kafka [9] is an open source, distributed publish-subscribe messaging system. A producer publishes messages to a Kafka topic that is created on a Kafka broker acting as a Kafka server. Kafka maintains feeds of messages in categories called topics. Kafka is run as a cluster comprised of one or more servers, each of which is called a broker. Brokers can divide messages into many partitions. Each partition is optionally replicated across a configurable number of servers for fault tolerance. Producers send messages over the network to the Kafka cluster which in turn serves them up to consumers. Brokers

and consumers use Zookeeper to get the state information and to track messages offsets, respectively.

Kafka supports two replication modes in processing: Synchronous replication: A message to be published is acknowledged as soon as it reaches 1 replica. Asynchronous replication: The only difference in this mode is that, as soon as a lead replica writes the message to its local log, a message is only acknowledged after it reaches multiple replicas. But, as a downside, this mode does not ensure message delivery in case of broker failure. 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. The leader is responsible for handling all read and write requests for the partition while the followers asynchronously replicate data from the leader.

If any of the follower in-sync replicas fail, the leader drops the failed follower from its ISR (in-sync replicas) list. After the configured timeout period will continue on the remaining replicas in ISR (in-sync replicas) list. 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 them with Zookeeper. The very first registered replica becomes the new lead replica and its log end offset (LEO) becomes the offset of the last commit. The rest of the registered replicas become the followers of the newly elected leader.

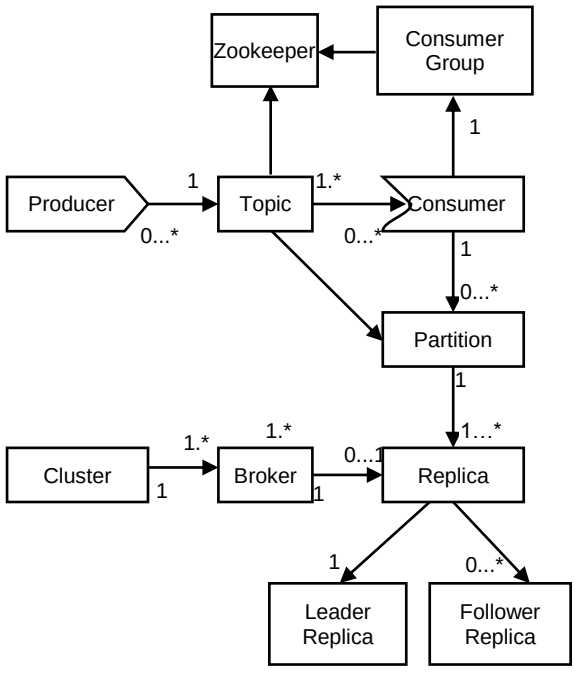

Figure 2. Processes in Kafka

Figure2 describes the processing of an Apache Kafka flow. 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 zero or more partitions per topic. A topic is replicated over one or more partitions. A partition has one or more replica. A replica is on one broker. A broker has zero or one replica per partitions. A partition has one leader and zero or more followers.

Zookeeper [12] is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Zookeeper is also a high-performance coordination service for distributed applications. 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 hostname and port, and the set of topics and the partitions stored on it. The service itself is distributed and highly reliable.

3.2 Apache Storm

Storm [10] is also an open source, distributed, reliable, and fault tolerant system for processing streams of large volumes of data in real-time. A Storm cluster follows a master-slave model where the master and slave processes are coordinated through Zookeeper. The Nimbus node is the master in a Storm cluster. Supervisor nodes are the worker nodes in a Storm cluster .In Storm terminology, [10] 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 topology of the storm is described as a part of the pipeline architecture as in Figure 1.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. The spout is the input stream source which can read from the external data source.

Bolt: The spout passes the data to a component called a bolt. Bolts are processor units which can process any number of streams and produce output streams. A bolt is responsible for transforming a stream.

4. 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 3.5.0uses as in Kafka-Storm integration. The Real-time data for experiments are mobile phone spam messages from the Grumble text Web site.

Table 1.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 @3.40GHz

The evaluation of overall Kafka processing in pipeline architecture is as follows:

1. Start Zookeeper server for processing.
2. Start Kafka local server to define broker id, port, and log dir.
3. Create a topic to show a successful creation message.
4. Producer publishes them by asynchronous type as messages to the Kafka cluster.
5. The consumer consumes messages from Zookeeper.
6. Check the leader and followers in ISR list and replica in Zookeeper.
7. Get process id and kill leader in one server.
8. Check total messages in processing by using GetOffsetShell tools
9. Evaluate the performance of producer running alone by using producer performance tools.
10. Evaluate the performance of the consumer by using consumer performance tools.
11. Compare the performance by testing the successful and failed processes
12. Get some messages and run Kafka-Storm integration pipeline.

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

Table 2.Summery of Experiments

Experiment	Description
1.Producer Performance	Deployed Apache Kafka on two servers, 40 partitions, two replication factor, different types of dataset with five batch size
2.Consumer Performance	Deployed Apache Kafka on two servers, 40 partitions, two replication factors and different types of dataset with five batch size
3.Producer Performance	Deployed Apache Kafka on three servers, 40 partitions ,three replication factors, different types of dataset with five batch size
4.Consumer Performance	Deployed Apache Kafka on three servers, 40 partitions ,three replication factor and different types of dataset with five batch size

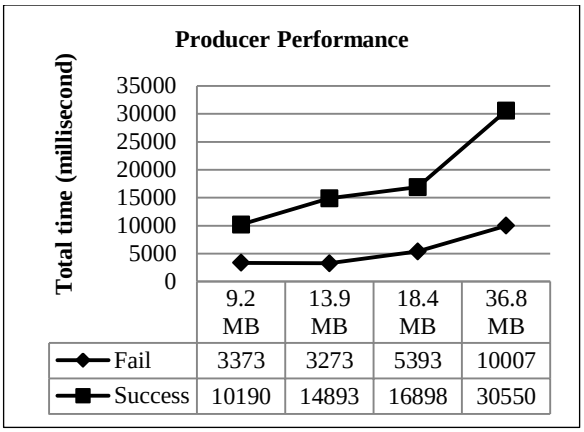

Figure 3. Producer Performance on two servers

A. Experiment 1:Producer Performance on two servers

Figure 3 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 3 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 and failed processes with 40 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. The successful processing time increases about three times than the failed process. There are a few reasons why successful processes performed much better. First, the Kafka producer currently doesn't wait for acknowledgments from the broker and sends messages as fast as the broker can handle. Data size is directly proportional to the performance time in producer performance.

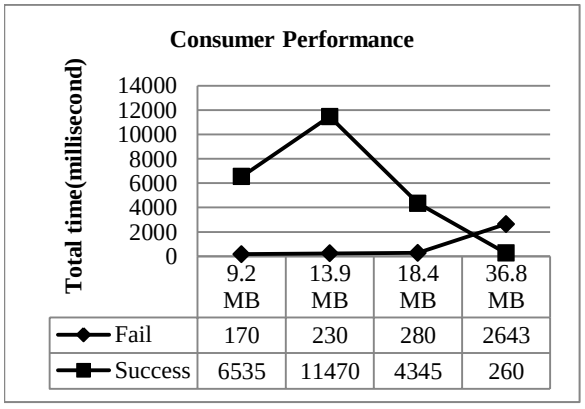

Figure 4. Consumer Performance on two servers

B. Experiment 2:Consumer Performance on two servers

Figure 4 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 consume messages asynchronously by five batches size. Figure 4 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 and failed processes with 40 partitions in various data sizes on two servers. The experiment emphasizes the total time by calculating based on consumer performance tools in Apache Kafka. The total time of failed processes is directly proportional to the data set. The comparison of successful process and failed processes are different .In 36.8 MB, the failed processing time raises than successful processes because of increasing messages lost. The system must reschedule processes for losing messages .Overall total time in this experiment is different in Experiment 1.

C. Experiment 3: Producer Performance on three 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 and failed processes with 40 partitions in various data sizes on three servers. The experiment emphasizes the total time by calculating based on producer performance tools in Apache Kafka. When we tested on three servers, the total time of failed processes raises than Experiment 1.The producer performance on three servers is more handle and faster than Experiment 1.

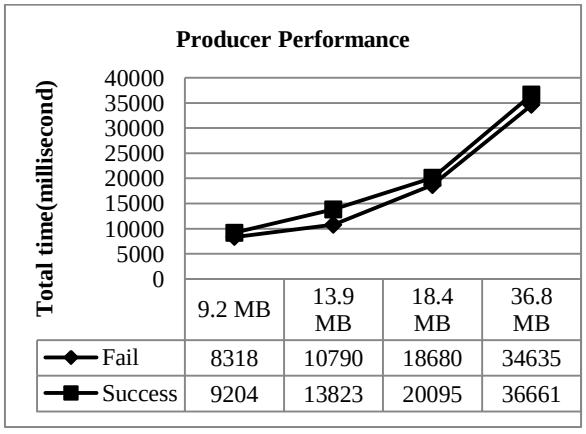

Figure 5. Producer Performance on three servers

D. Experiment 4: Consumer Performance on three servers

Figure 6 shows a single consumer to consume various data size 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 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. It compares the total times on difference between successful and failed processes with 40 partitions in various data size on three servers. The experiment emphasizes the total time by calculating based on consumer performance tools in Apache Kafka. In Figure 6, the failed processing time of the most dataset is significantly rising than successful processes. 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. When we tested more servers in Experiment 4, the processing time of the failed process is more increase than Experiment 2.

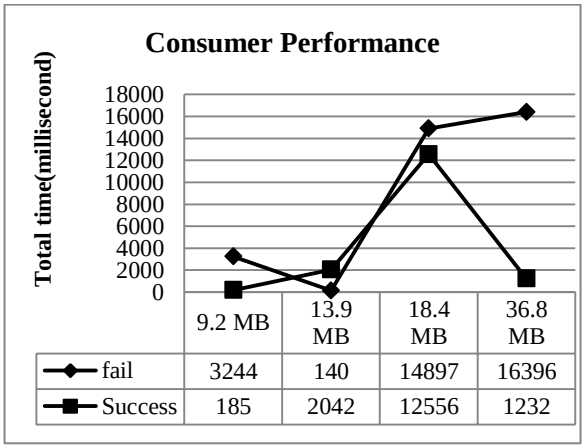

Figure 6. Consumer Performance on three servers

The system tested on two types of 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 Equation 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 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) to calculate Total consumed message bytes (MB) representα, β represents total bytes sent and memory size is 1024*1024.

$$\propto = \frac{(\beta * 1.0)}{(1024 * 1024)} \qquad (1)$$

In (2): Calculation the Average consumed MB bytes per second representα, βrepresents total messages sent and elapsed seconds.

$$\alpha = \frac{\beta}{\theta} \qquad (2)$$

In (3): Calculation the elapsed timeα,the end time (milliseconds) is β and the start time (milliseconds) is £.

$$\propto = \frac{(\beta - £)}{1000.0} \qquad (3)$$

In (4): Calculation the Average consumed message bytes per secondα, β represents total messages sent and £ represents elapsed seconds.

$$\alpha = \frac{\beta}{£} \qquad (4)$$

Table 3. Experimental Summary for Producer

Data size (MB)	success /fail	No of messages per second on two Servers	No of messages per second on three Servers
9.2 MB	S	10940.2	12112.2
9.2 MB	F	33050.9	13402.4
13.9 MB	S	11227.8	12096.9
13.9 MB	F	51089.5	15497.3
18.4 MB	S	13194.2	11095.5
18.4 MB	F	41341.7	11936.1
36.8 MB	S	14596.6	12163.4
36.8 MB	F	44561.4	12874.9

Table 3 summarizes experimental results. The table compares the number of messages per one second between two servers and three servers. The results indicate that the number of messages per second on two types of the server is different. The number of messages tested on three servers is slightly different than two servers. The processing time of Figure (3) and (5) correlate to the number of messages per second in Table 3.The experimental results calculated based on Equation (1) to (4).According to the result, producer performance on three servers improves than two servers.

Table 4. Experimental Summary for Consumer

Data size (MB)	success /fail	No of messages per second on two Servers	No of messages per second on three Servers
9.2 MB	S	34117.9	1205194.6
9.2 MB	F	1287541	68171.4
13.9 MB	S	29157.1	163776.2
13.9 MB	F	1442182.6	2374692.9
18.4 MB	S	102626.2	35514.5
18.4 MB	F	1587103.5	29775.3
36.8 MB	S	1715096.2	1922068.9
36.8 MB	F	244064.5	54285.2

Table4 summarizes 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 Figure (4) and (6) correlate to the number of messages per second in Table 4.The experimental results calculated based on Equation (1) to (4). According to the experiments, the system needs to control the performance on many servers. We need to reduce completion time and recover lost messages.

In the future, Message logging based check pointing 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. Check pointing is utilized to limit log size and recovery latency.

5. Conclusion

In this paper, the comparison of the processing time between the successful process and failed process on two types of the server was evaluated. The analysis of the results has shown that the comparison of four types of data set with servers. Any application that needs to process a large number of messages is tolerant lost a few. If the system handles lost messages, Kafka will get the more reliable messages. We conducted several experiments to determine higher performance and to get reliable streaming

data. To develop the reliability of pipeline architecture, we need to evaluate the higher performance of processes in Apache Kafka.

As the future direction, we intend to handle the performance of Kafka processing in real time pipeline architecture by using Message logging based check pointing. By analyzing many processes, we can achieve more reliable data in Kafka-storm pipeline architecture.

6. References

[1] Jay Kreps,NehaNarkhede, Jun Rao,"Kafka: a Distributed Messaging System for Log Processing", May , 2015.

[2] Hassan Nazeer, Waheed Iqbal, Fawaz Bokhari,Faisal Bukhari, Shuja Ur Rehman Baig ,"Real-time Text Analytics Pipeline Using Open-source Big Data Tools" ,December ,2017.

[3] Martin Kleppmann, "Kafka, Semza and the Unix Philosophy of Distributed Data" Bulletin of the IEEE computer Society Technical Committee on Data Engineering, July, 2016.

[4] Paul Le Noac'h, Alexandru Costan , Luc Boug´e ," A Performance Evaluation of Apache Kafka in Support of Big Data Streaming Applications" , 2017 IEEE International Conference on Big Data (BIGDATA), December ,2017.

[5] Wenjie Yang, Xingang Liu and Lan Zhang, "Big Data Real-time Processing Based on Storm", 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013.

[6] Mohit Maske, Dr. Prakash Prasad, "A Real Time Processing and Streaming of Wireless Network Data using Storm ", International Journal of Advanced Research in Computer Science and Software Engineering, January, 2015.

[7] SeverinSimko ,"Performance testing of Apache Storm framework", Masaryk University, Faculty of Informatics, Brno, Autumn 2015

[8] Muhammad Syafrudin 1 ID , Norma LatifFitriyani 1 ID , Donglai Li 1, GanjarAlfian 2 ID , Jongtae Rhee 1 and Yong-Shin Kang 3, "An Open Source-Based Real-Time Data Processing Architecture Framework for Manufacturing Sustainability" ,*Sustainability* Open Access Journal 2017 ,20 November 2017

[9] NishantGarg, "Apache Kafka", PACKT Publishing UK,October , 2013

[10] Ankit Jain, AnandNalya, "Learning Storm" PACKT Publishing UK, August, 2014.

[11] Tanenbaum Andrew, Tanenbaum-Distributed operating system, Wikipedia, p-100, 1994.

[12] http://zookeeper.apache.org/.