ISSN: 2710-2599

International Journal of Data Science and Big Data Analytics

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Big Data Pipeline Architecture: Building the Reliability of Real-Time Messaging System on Apache Kafka

Thandar Aung[1*] and Aung Htein Maw[2]

[1]University of Computer Studies (Thaton), God Village, Thaton, Myanmar. E-mail: thandaraung@ucstt.edu.mm
[2]NETsys Research Lab, University of Information Technology, Myanmar. E-mail: ahmaw@uit.edu.mm

Article Info

Volume 6, Issue 1, May 2026
Received : 14 February 2026
Accepted : 30 April 2026
Published : 25 May 2026
doi: 10.67191/IJDSBDA.6.1.2026.32-54

Abstract

In big data era, increasing real-time requirements and reliability requirements drive organizations to make time-critical decisions. Reliable messaging systems are essential for innovative infrastructure and business application. Apache Kafka is a widely used distributed real-time messaging platform but ensuring reliable message delivery remains challenging due to message loss risks during server failures. The paper examines the default checkpoint interval for controlling the message loss problems and increases recovery time. The paper emphasizes the checkpoint interval-based message logging approach for the message recovery. The CIMLA (Checkpoint Interval Message Logging Algorithm) enhances the reliable message delivery by defining the optimal checkpoint interval and achieves the message logging to implement the recoverability requirements of Apache Kafka. The paper investigates various comparative studies on different data sizes, server failure cases, and performs the reliability enhancement of Apache Kafka by contributing to better message delivery and faster recovery time of the modified system.

Keywords: *Checkpoint interval, Message recovery, Message logging, Fault tolerance, Reliability*

1. Introduction

Big data is an innovative technology for applying in a business organization. Gathering and analyzing vast amounts of data are the two main issues associated with big data. Many organizations execute big data analytics for collecting valuable information from the large datasets they manage. Information reliability, computational methods, and complexities need to overcome the problems of processing big data. In business and IT industries, innovative infrastructure and creative thinking are an important role to analyze user

* *Corresponding author: Thandar Aung, University of Computer Studies (Thaton), God Village, Thaton, Myanmar. E-mail: thandaraung@ucstt.edu.mm*

behavior data, tracing application performance, activity data such as logs and messages. Real-time big data analytics is a popular trend to analyze and process big data in scientific research and IT infrastructure.

A software capability called real-time big data analytics may analyze huge amounts of incoming data as it has been generated or stored using the IT infrastructure. Real-time big data analytics is a popular tool for IT industries and organizations which produce the product in a very short time. It performs by importing big data stored with a system at run time and executes a big data analysis algorithm over it. The benefit of real-time big data analytics is that it allows businesses of all sizes to gain important intelligence faster than ever before by utilizing insights from large amounts of data. IT organizations most commonly use this technology in industries that produce or capture large amounts of data in a short time, such as logistics, banking, or IT.

Innovative organizations need to enhance the performance of real-time processing more. Fault tolerance is a key factor to develop the reliability of real-time processing. Therefore, the system focuses on the improvement of fault tolerance of the real-time messaging system. The system uses the popular real-time messaging system, Apache Kafka, to solve the challenges of fault tolerance. There are many ways to promote fault tolerance in real-time messaging systems. Among them, the system emphasizes the message delivery for solving the lost message problem in the server failure case of Apache Kafka. There are three types of solving methods in a real-time messaging system: check pointing, replication method and messaging logging. Apache Kafka used the replication method for catching the messages and identified checkpoint intervals to examine the published messages. Apache Kafka establishes the replication method by dividing partitions in Kafka clustering. Apache Kafka receives the published messages by copying among the partitions. However, the weak point of checkpoint interval affects message delivery.

In an asynchronous case, Kafka has the drawback of the recovery method in message delivery because of server failure cases. Default checkpoint interval cannot control any file size and catch up the message with id in message recovery. In this case, the system needs to handle the checkpoint interval in message delivery and to reduce the checkpoint overhead time. Hence, the paper considers solving the problem by using message logging based on check pointing. The optimal checkpoint interval can adjust the problem of check pointing in the message delivery of Kafka. The system creates an efficient message logging algorithm based on optimal checkpoint intervals. Moreover, there are three contributions to solve the three main issues: message delivery process, message recovery and reliability improvement of Kafka.

1. Defining the optimal checkpoint interval in Apache Kafka to handle failure points during message delivery.
2. Creating the message logging system to recover the message loss and the message recovery time.
3. Measuring the reliability factors to prove the improvement of the performance of Kafka.

The rest of the paper is organized as follows: The fault tolerance and the Literature review of Apache Kafka are discussed in Section 2. Section 3 explains Apache Kafka Pipeline Architecture with CIMLA and the algorithm utilization of them. In section 4, we demonstrate the experimental evaluations of our system. Section 5 analyzes the reliability enhancement of experimental results. Finally, Section 6 concludes the system.

2. Literature Review

Most researchers solved the challenges in big data and used various mechanisms with real-time big data processing. The academic research papers related to the problems and solutions of real-time messaging system. Apache Kafka is a popular distributed real-time messaging system for innovation in many organizations. In this approach, the beneficial reasons for real-time processing in big data have been proposed from many aspects in which the performance of the real-time messaging system Wu *et al.* (2020), Wu *et al.* (2019), Wu *et al.* (2019) and Thein *et al.* (2017). The system also discusses the investigated issues to overcome the weak point of the previous researchers.

Wu *et al.* (2020) discussed the impact of batch size effects on the performance and reliability of Apache Kafka. The authors compared both positive and negative network conditions in order to evaluate the suggested batching strategy. The investigation of the authors pointed out how to batch messages properly in Apache Kafka to meet real-time requirements. The authors used Docker containers for analyzing the correlation

between batch size and the distribution of end-to-end latency. The paper emphasizes the optimal checkpoint interval for message recovery in server failure case which is powerful to develop the performance and reliability of Apache Kafka.

Wu *et al.* (2019) discussed about the process and description of different messages delivery of apache Kafka. The author proposed a tool to validate different message delivery on poor network. The authors proved that their proposed tools could control the fault injection of the environment in the network delay. But this paper has the weak point in controlling the message delivery for fault tolerance. Therefore, the paper considers handling the message delivery by defining checkpoint interval-based message logging.

Wu (2019) also discussed the message loss and duplicate messages in the network fail problem and focused on the effectiveness of the prediction model by changing configuration parameter. However, the dynamic configuration cannot handle all situations and cause the challenges of message delivery in server failure case. The paper proves the improvement of message delivery by comparing various experimental evaluations.

Thein *et al.* (2017) described the need for enforcing security policies to protect sensitive data in big data security and scalability. The system used sticky policy to control the bank transaction messages. The author proved the enhancement of the security of bank transactions by using real-time big data pipeline architecture. Although the author evaluated the development of fault tolerance, they need to consider the server failure case in pipeline architecture. The paper establishes the ability of Kafka by dealing with the check pointing mechanism and shows the reliability of real-time messaging system is also important for any use cases.

Most researchers proved the fault tolerance in their point of view (Kumar *et al.*, 2011; Daly, 2003). Message logging based on an optimal checkpoint interval is used to illustrate the message recovery strategy. Researchers demonstrated the effectiveness of messaging logging and emphasized the significance of rollback recovery in the development of fault tolerance (Rao and Naid, 2008).

Researcher (Kumar *et al.*, 2011) described the different techniques for tolerance fault in the real-time distributed system. The author explained that a fault can occur due to link failure, resource failure or any other reason is to be tolerated for working the system smoothly and accurately. The author proved effective methods to detect and recover faults from the real-time distributed system. Fault tolerance depends on the occurrence of failures. The author confirmed that the system is scalable, reliable and feasible by applying the fault tolerance method. The paper considers the effective fault tolerance method for solving the failure case in real-time processing.

Daly (2003) described by comparing the modified two models which provide accurate high order approximations to the optimal checkpoint interval. The author proposed to derive a result, in terms of a simple analytic approximation, easily accessible to the application user, with well-defined error bounds. The author proposed the model based on the first approximation optimal checkpoint interval for quantifying the optimum restart interval that minimizes the total application run time (Liu *et al.*, 2008).

A new system that combines sender-based message recording with an active interval mechanism to minimize output commit latency and calculate lost messages was proposed by Rao and Naidu (2008). The method significantly reduces message logging overhead by only logging messages within a defined interval. Improved fault tolerance and effective message recovery for real-time messaging systems are demonstrated by the evaluation.

Different from the previous papers, the system is implemented to define smart interval based on coordinated check pointing and proved the reliability of message delivery based on CIMLA (Checkpoint Interval Message Logging Algorithm).

3. System Architecture for Real-Time Big Data Pipeline Architecture

This section focuses on in detail the main components of Kafka, the processing flow, the main characteristics of Kafka, message transmission and the system architecture of Kafka. In addition, the current fault tolerance in Apache Kafka and check pointing techniques are discussed in the system. Apache Kafka is a fault tolerant,

reliable messaging system for data transferring in real-time. Apache Kafka differs from the traditional messaging system in four cases. Apache Kafka can scale out by constructing as a distributed system and offer high thoughts for both publishing and subscribing. Then, it handles multi-subscribers and automatically balances the consumers during failure. Kafka (Garg, 2013) can be used for stream processing, website activity tracking, metrics collection, monitoring and log aggregation. There are five main components of Kafka architecture: producer, broker, topic consumer and zookeeper. Producers are the important part of Kafka messaging system which is responsible for publishing messages. The producer can publish messages into the Kafka topic by using two modes: synchronous and asynchronous. In synchronous mode, as soon as a message is published, the producer sends a request to the broker. Synchronous mode reduces the throughput and acts as the blocking operation that is better to run producers in asynchronous mode. In asynchronous mode, the producer can send messages by registering a completion listener to monitor message send success or failure. Asynchronous sending can increase performance in certain circumstances. The producer identifies two parameters for optimal performance: batch size and linger time. Batch size is the size of data to be sent in one batch, measured in bytes. In the system, the producer emphasizes message transmission to brokers by the asynchronous types. Kafka Topic is also considered as a message category or feed name to which messages are published. The topic is a historical log of events. An event is attached to the end of a topic when an external system submits it to Kafka.

Consumer is an external application that subscribes for the consumption of messages on a specific topic on the Kafka broker. A consumer is a member of one consumer group. Each message in the topic delivers to one consumer from a consumer group (with the same group ID). Differ from the traditional messaging/publish-subscribe systems, Kafka can maintain long-term storage of messages because Kafka does not delete messages after delivery. Apache Zookeeper (Andor Molnar, 2022) is a centralized service for configuring devices, namely providing distributed synchronization, and delivering group services. Zookeeper is also a distributed application coordination service with outstanding performance.

Message Transmission: When compared to traditional messaging/publish subscribe systems, reading data from Kafka works a little differently. Brokers do not discard messages after delivery. This design decision provides several benefits: consumers' reading progress does not need to be tracked by brokers. Consumers must recognize which messages they have successfully processed if brokers track read progress. As a result, if an earlier message is unsuccessfully processed but a later message is successful, the initial message is resent out of order. Consumers can go back in time and reprocess previous data because brokers do not track progress and do not erase data after delivery.

Kafka employs a producer-consumer pattern to work with streaming data. Producers are responsible for sending messages while others are consumers for receiving and processing them. There are three types of message delivery guarantees supported by the Apache Kafka: at most once, at least once and exactly once. It is critical to know the type of asynchronous message delivery because each will influence the configuration of producers and consumers and the performance that Kafka can provide. Therefore, the system discusses the challenges in the message delivery process of Kafka and how to improve the message delivery of Kafka.

The main challenge of Kafka processing is message delivery which has three message delivery semantics. The producer receives messages in batches that are only available in asynchronous mode. It publishes either the fixed number of messages or a specified latency defined by producer configuration in asynchronous mode. So, the producer focuses on the fixed number of messages by sending a batch-based transmission scheme. The paper analyzes the message delivery at most once in the case of a server failure because it concentrates on the asynchronous mode. The producer sends each message to the Kafka topic once without waiting for any acknowledgment from the Kafka topic. There is no guarantee that a message is successfully delivered. This offers a high throughput and makes good use of available bandwidth but there is no mechanism to avoid message loss.

The paper intends to handle the configuration of message delivery for solving at most one message delivery weakness. The process examines depending on the message delivery state from producer to Kafka topic. The system describes the message delivery state of the message by processing in Figure 1. In Figure 1, the system assumes the message delivery state before sending to the Kafka topic as the initial state marked as the ready to send state.

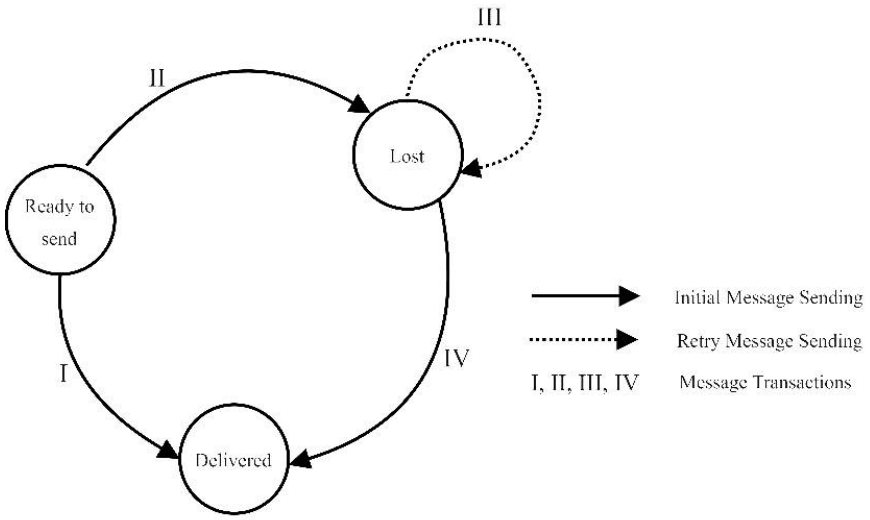

Figure 1: Message Delivery State

The message flows from the initial state to the target state assumed as the delivered state which is identified as Case 1. Case 1 represents the successful case of the message delivery under normal circumstances. Case 1 and Case 2 occurred by applying at most once message delivery because the producer sends each message once and does not respond from the brokers. The producer retries sending the message without receiving an acknowledgment until a timeout expires. If the server still fails, the message is delayed in the lost state. Since the producer can be configured to check message count in the failure case, it can keep retrying before receiving the acknowledgment. So, the system considers defining the interval time for checking the message reached or not. The system defines the interval time based on the publishing batched-based scheme. The system assumes the checkpoint interval time as the parameter ôi. Case 3 recovers the lost messages causing the server failure, the ôi reaches the checkpoint interval time. Then the producer recovers the lost message in a lost state. Case 4 examines the successful message delivery state at the end. All of message delivery states occurred in the system which is described in Table 1.

Table 1 explains the message transactions according to the message delivery state described in Figure 1. Table 1 demonstrates the successful message delivery state in Case 1 and Case 4. The system analyzes based on all the possible cases in message delivery in Apache Kafka. The system describes the message transactions for Case 2 and Case 3 as the possibility of message loss failure. The weak point of message delivery caused in Case 2 and Case 3 is the main process of the failure for sending the message from the producer and Kafka topic.

Table 1: Possible Cases of Message Delivery in Apache Kafka

No	Message Transactions
1	I
2	II
3	$II \rightarrow III.\tau_i$
4	$II \rightarrow III.\tau_i \rightarrow IV$

Asynchronous replication is used by Apache Kafka topics to process data across multiple partitions, and producers use flexible partitioning techniques to allocate messages to each partition. The performance of producers and consumers under various partition configurations and datasets is assessed in this study. The findings emphasize message delivery constraints and insufficient fault tolerance in the underlying broker infrastructure by demonstrating that asynchronous replication may prevent the default Kafka pipeline from recovering all messages after server outages.

In message delivery of Kafka, the producer publishes the messages continuously without waiting for the acknowledgment from the Kafka topic occurring the server failure case. Therefore, the system needs to check the failure cases in time. The frequency of these commits is determined by the configuration parameter in the broker. The frequency with which the system updates the persistent messages of the last flush which acts as the log recovery point as a log. flush.offset. checkpoint.interval.ms. In default Apache Kafka, the constant value is defined as log.flush.offset.checkpoint.interval.ms which acts as the same interval time on any data sizes and situation. Therefore, the default interval cannot manage on server failure cases which occurs the message loss problem and decreases the Kafka performance.

Figure 2 explains the improvement of message delivery by applying the checkpoint interval messages logging algorithm in Apache Kafka. In Figure 2, the producer receives the messages file from external sources as the batch-based transmission scheme. The producer needs to know the producer properties topic name, interval time and message information (message_id, values). The producer defines message_id and counts the number of messages in the producer message log and stores into the buffer. All messages were published to the Kafka topic. The consumer receives messages from the Kafka topic. The system handles the message delivery process according to the configuration setting. The system then checks the server failure. If the server failure occurs, it moves to the callback state because of checking the messages reach or not. If the server does not occur failure, publish messages store to the messages log. After that, the callback state performs the confirmation by checking the message log from the producer side. The producer processes the retry state for recovering the lost messages in message delivery. All messages delivery processing depends on the optimal checkpoint interval. By combining these advantages of CIMLA with message delivery, the modified Kafka can perform fault tolerance efficiently.

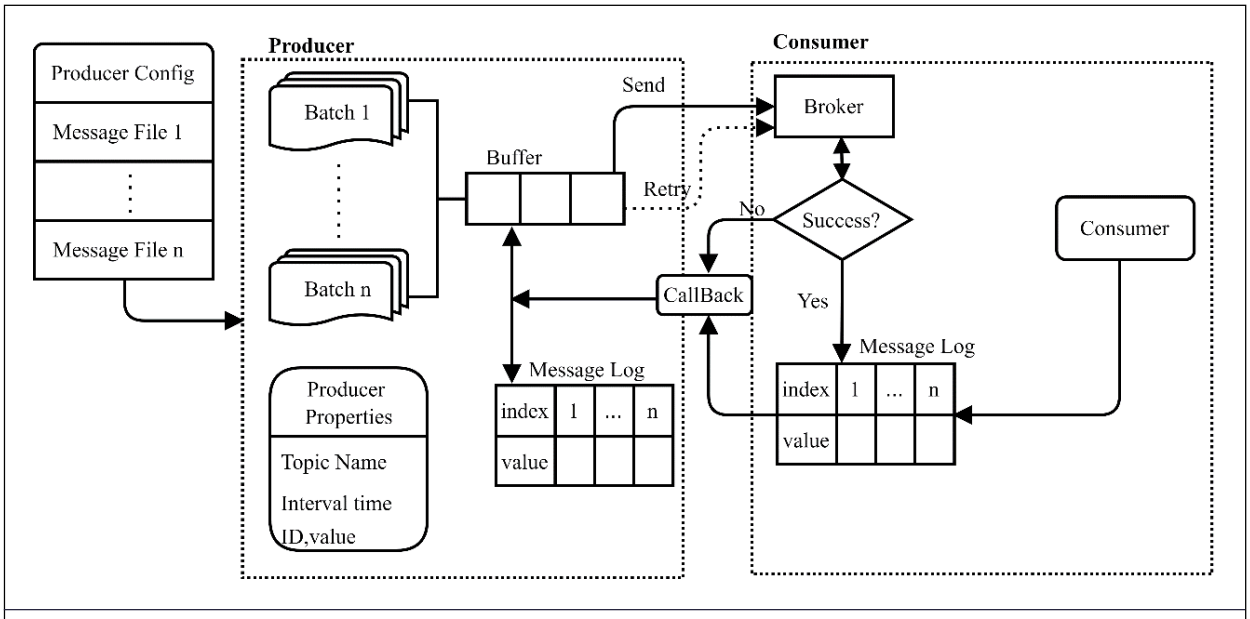

Figure 2: Message Delivery Process with CIMLA

The system focuses on effective message delivery with CIMLA to prove the improvement of message delivery of Apache Kafka. According to the analysis of the message delivery, the system handles the checkpoint to the disk in Kafka. Defining checkpoint intervals becomes the core part of solving the lost message problem in Kafka. Therefore, the system creates CIMLA based on the optimal checkpoint interval. In Table 2, the system explains the processing steps of Kafka processing which focuses on the message recovery occurring failure cases and improvement of the fault tolerance of Kafka.

The algorithm identifies the parameters for recovery processing such as the number of checkpoint interval T and number of lost messages L. The system defines three message counters: MSCi[j] represents the message counter from process Pi to process Pj on the producer side, MRCi[j] counts the messages on the consumer side and left-cntj[i] counts the messages occurring the interval time on the consumer side. Msg[m] and Msg[m1] perform for saving the message_id in processing.

Table 2: CIMLA (Checkpoint Interval Message Logging Algorithm)

```
Message Counter send from Pi to Pj , MSCi[j] =0
tag, Number of Checkpoint Interval T, Number of Lost Messages L
Messages receive count MRCi[j]=0
Messages counter Pi to Pj left-cntj[i] =0
Msg[m]=message_id from producer
Msg[m1] =message_id from consumer
Step I: This step is executed at initiator process Pinitiator
        a. Calculate optimal checkpoint interval
        b. Define number of Interval by using optimal checkpoint
           interval T
Step II: If Producer of the message is active Interval,   then
                    tag=1;

          else       tag=0;

           While (T! =NULL)
              Begin
                 If Message send Pi to Pj   then
                    MSCi[j]= MSCi[j] +1;
                    Save-state (Msg[m]);
                End
Step III:  If Consumer message is active Interval then
            While (T! = NULL)
           Begin
                 If  Message receive Pi to Pj   then
                     MRCj[i] = MRCj[i] +1;
                 If  Message receive Pj to Pi  then
                     left-cntj[i] = left-cntj[i] +1;
                     Save-state (Msg[m1]);
           End
Step IV: This Step is calculated Message Lost
            L = MSCi [j] – (MRCj[i] + left-cntj[i] )
           While (MRCi[j]! =NULL)
                Begin
                   If (Msg[m]! = Msg[m1])
                  Reply[m] = Msg[m]
                End
Step V: Recovery State is executed for message recovery
           Resend Reply[m] from Step II to Step III
```

Firstly, the system defines the optimal checkpoint interval based on the number of text files and estimates the number of intervals to identify in Kafka processing. All processing steps perform depending on the checkpoint interval. Therefore, the system needs to know the main parameters for optimal checkpoint calculation. The system explains the optimal checkpoint interval calculation from Equation 1 to Equation 4.

Firstly, the system calculates the Mean Time Between Failure (MTBF) to find the optimal checkpoint interval. MTBF is the predicted time interval between system failures during real-time operation. The system

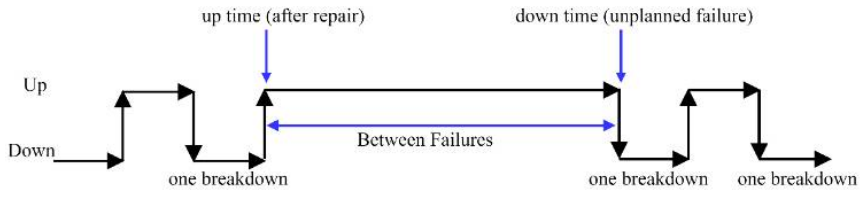

Figure 3: Calculation of Mean Time between Failure

extracts the total uptime based on the batch sizes of publishing messages. Total uptime has calculated the summation of time between failure in Kafka processing. And then, the time between failure is the difference of uptime time and downtime during Kafka processing. Secondly, the system defines the number of breakdowns. The number of breakdowns is the estimation of failure time which depends on batch size defined by the producer side. The calculation of meantime between failure is described in Figure 3.

$$Time\ Between\ Failure = T_{dt} - T_{ut} \quad ...(1)$$

Downtime T_{dt} means the instantaneous time the process goes down and Uptime T_{ut} means the time the process goes up after downtime. The total uptime means the summation of uptime of Kafka processing.

$$Total\ Uptime = \sum (T_{dt} - T_{ut}) \quad ...(2)$$

The system then calculates the MTBF based on the above Equations (1) and (2). The number of breakdowns assumes the number of failures among the batch sizes. Therefore, the number of breakdowns depends on the batch sizes classification from the producer.

$$MTBF = \frac{Total\ Uptime}{number\ of\ breakdown} \quad ...(3)$$

The optimal checkpoint interval is calculated using the mean time between failures. The system uses the Kafka configuration's save time T_{sd} to compute the optimal checkpoint interval T_{opt}. In Kafka processing T_{sd} refers to the default saving time on the local disk. MTBF represents Mean Time Between Failure in Equation (3). Equation (4) describes calculating the optimal checkpoint interval T_{opt} for Kafka.

$$T_{opt} = \sqrt{2 \times T_{sd} \times MTBF} \quad ...(4)$$

After defining interval in Step I of Table 2, the producer publishes the messages to the Kafka topic and records the message information (id and count) in Step II. On the consumer side, the Kafka topic receives messages in two states: the normal transmission state and during interval state. The two message receiving states receive the message_id and count the number of messages. Then, all the received messages save to the save state in Step III. Step IV describes the callback state for checking whether the messages reach or not then the lost message extracts for recovery. The recovery processing is performed in Step V. Message delivery develops the fault tolerance of Apache Kafka. The system proved the reliability of Kafka by promoting the fault tolerance factor.

3.1. Modified System Flow of Apache Kafka

Figure 4 demonstrates the processing flow in Apache Kafka by applying a checkpoint interval message logging algorithm as the modified system. In Figure 4, the producer predefines the interval based on the publishing messages and processes the saving messages with id. The producer sends all the message files to the Kafka topic which receives messages information and runs in the role of predefined configuration. The system divides the messages according to the identified number of partitions and also checks the server's state concurrently. All messages are temporarily stored in the message logging queue in any server state. The message logging state sends the messages to the callback state to check whether all of the messages are reached or not. If the lost message problem is in a callback state, the system extracts the lost messages with id. And then, the system recovers the messages by retrying the publishing state from the producer to Kafka topic. If the system has a successful state by checking message-id, all the information is sent to the ISR. The consumer pulls the messages from the ISR list and informs the zookeeper. Zookeeper handles the information between the Kafka topic and the consumer group which has one or more consumers. After handling the message delivery flow, the system needs to measure the fault tolerance of Kafka. The system uses the reliability factors in Apache Kafka for measuring the current system performance. In this system, the development of system performance enhances the reliability in the modified system.

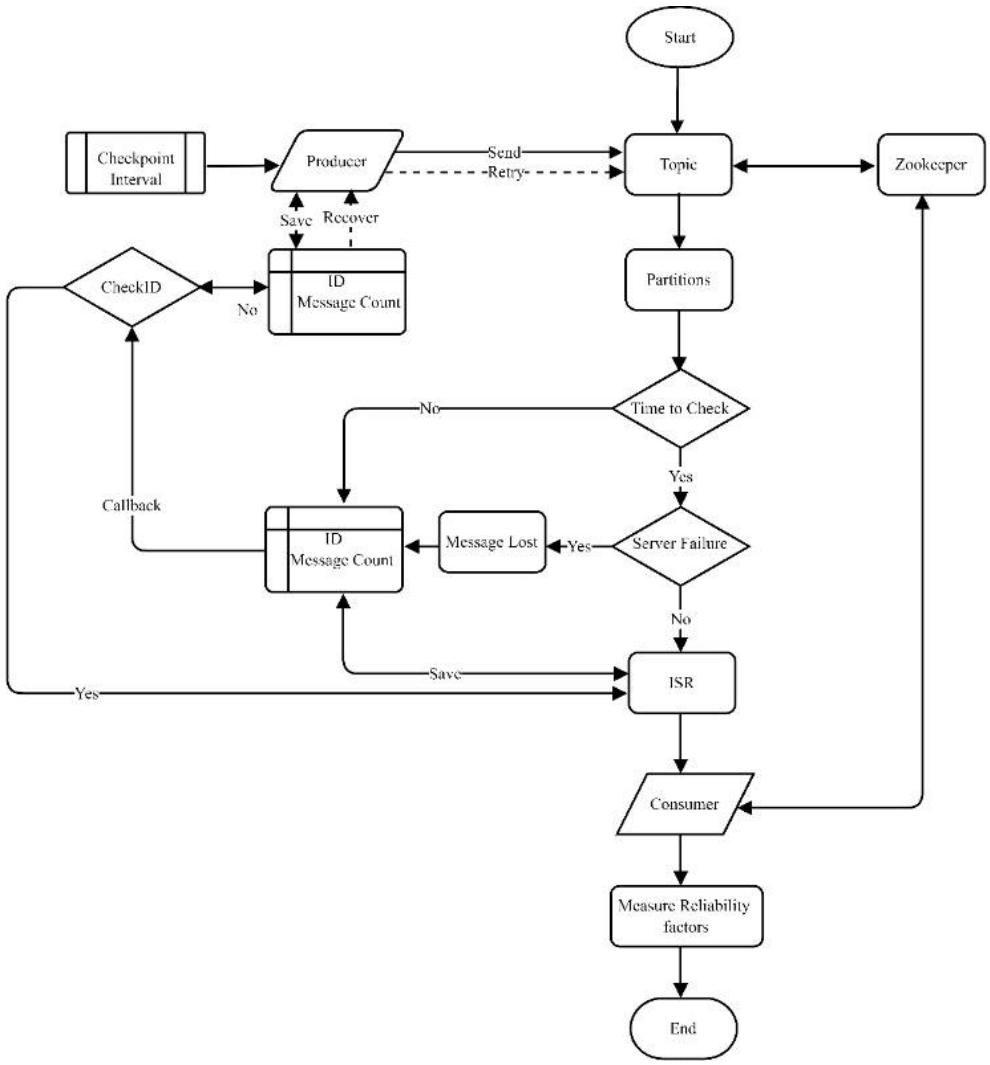

Figure 4: Modified System Flow of Apache Kafka

Figure 4 illustrates how to apply the CIMLA to Apache Kafka's message delivery configuration and how to advance the message delivery of Apache Kafka. According to the survey of Kafka configuration parameters, the system analyzes several experiments on how to configure those parameters to gain the best performance for various applications. On default Kafka pipeline architecture, the system mentions a major weakness in the message delivery process. The analysis of Kafka processing based on several partitions and servers indicates that managing message delivery between producer and consumer is essential for Kafka. The default system's recovery procedure is unable to process all the messages in the event of a server failure.

In default Kafka, the checkpoint interval check message log in every **60 seconds** which is the main challenge to catch up on all of the messages in the server failure case. Default checkpoint interval time is not sufficient for catching up all the messages in various data sizes and batch sizes. Message lost problems occur in the server failure cases. It becomes the problem of insufficient fault tolerance of the basic messaging system and infrastructure. Message lost problems occur in the server failure cases. Therefore, the system focuses on the message delivery which depends on the interval time for message recovery process. Changing the interval configuration parameter for message log guarantees message delivery of Kafka. The system architecture aims to develop Apache Kafka's fault tolerance by describing the impact of the CIMLA algorithm.

Figure 5 demonstrates the modified system architecture according to the modified system flow of Apache Kafka. In Figure 5, CIMLA handles the checkpoint interval in message delivery and demonstrates how to advance the message delivery of Apache Kafka.

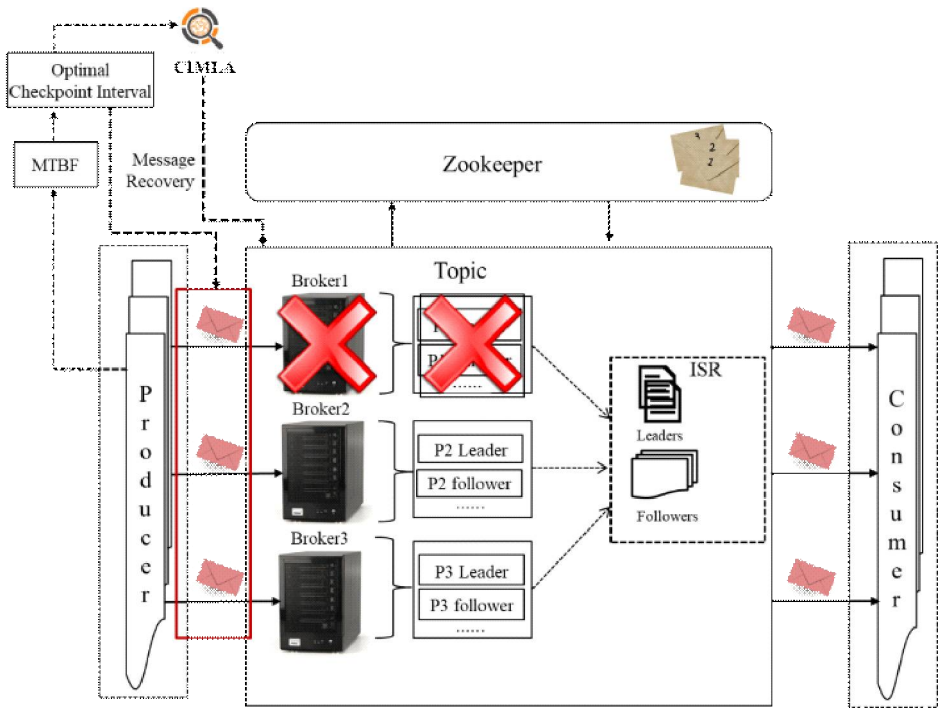

Figure 5: Modified System Architecture of Apache Kafka

The system considers the total overhead cost for defining the optimal checkpoint interval in Kafka configuration and needs to validate the reliability factors of the modified system. To test the performance of determining optimal checkpoint interval in Kafka, the system highlights the fixed checkpoint interval method. Compared to other methods, the fixed checkpoint interval method's benefits enhance the computation of overall overhead costs.

The optimal checkpoint interval method provides for recovering the lost messages in a real-time messaging system. Then, the system calculates the number of checkpoints which depends on the total uptime and optimal checkpoint interval. Figure 6 shows how the fixed checkpoint interval method performs and how many checkpoints there are in the ith cycle. Based on T_{opt} (the ideal checkpoint interval size) and T_{sd} (save time to disk), the system determines C_i (the start time of ith checkpoint).

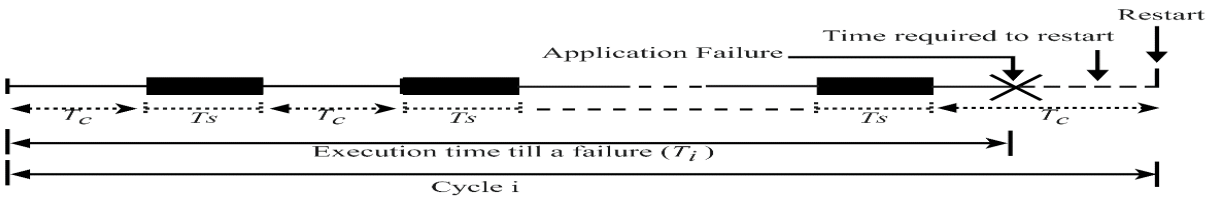

Figure 6: Fixed Checkpoint Interval

$$N_i = \left\lfloor T_{if} \Big/ \left(T_{opt} + T_{sd}\right) \right\rfloor \quad \text{...(5)}$$

In the fixed checkpoint interval method, the length of each checkpoint interval is fixed. During failure recovery, T_{opt} system calculates the total overhead cost due to check pointing and restarting. T_{if} (time of failure), (optimal checkpoint interval size) and T_{sd} (save time to disk) are used in Equation (5) to compute N_i (the number of checkpoints taken in the ith cycle) (save time to disk).

$$CC_i = N_i T_{sd} \quad \text{...(6)}$$

In Equation (6), the system calculates CC_i (checkpointing cost in the i^{th} cycle) by using N_i (the number of checkpoints taken in the i^{th} cycle) and T_{sd} (save time to disk). Checkpoint cost measures based on the number of checkpoints.

$$Rb_i = (T_{if} - N_i(T_{opt} + T_{sd})) \quad ...(7)$$

In Equation (7), Rollback cost is conversely proportional to the number of checkpoints. Rb_i (rollback cost for the ith cycle) is calculated using parameters N_i (the number of checkpoints taken in the ith cycle), T_{if} (the time when the i^{th} failure occurred), T_{opt} (optimal checkpoint interval size) and T_{sd} (save time to disk). N_i is the number of checkpoints to be taken in ith cycle (before ith failure occurs) and ($T_{opt} + T_{sd}$) optimal checkpoint interval and save time in Kafka. Rollback recovery has measured the difference of current server failure and the number of checkpoints to be taken in i^{th} cycle and each checkpoint interval cost.

The system needs to know the restart time T_{rt} after the failure case, Kafka defines the restart time by default. The overall waste of time in the i^{th} Kafka processing cycle $E(T_{cost})$ is determined by rollback, checkpoint and restart costs. The restart time restarts the state that caused the previous failure in Kafka processing. Equation (8) is a critical part of demonstrating the evolution of our system.

$$E(T_{cost}) = \left(\left\lfloor T_{if} \Big/ (T_{opt} + T_{sd}) \right\rfloor \times T_{sd}\right) + (Tif - \left(\left\lfloor T_{if} \Big/ (T_{opt} + T_{sd}) \right\rfloor \times (T_{opt} + T_{sd})\right) + T_{rt} \quad ...(8)$$

3.2. Reliability Measurement of Apache Kafka

The system describes the measurement of reliability factors for applying CIMLA and investigates the performance improvement of Kafka. The most important metrics for measuring overall system performance are message loss rate, producer throughput, consumer throughput and latency. In Equation (9), the possibility of message loss rate P_1 is calculated by the system using the number of lost counts C_{1m} and producer publishing messages P_m. Message loss rate is the main point for evaluating the reliability of message delivery.

$$P_1 = \frac{C_{1m}}{P_m} \quad ...(9)$$

Then, the system measures the throughput of the messages published from the producer to the topic in a specified time interval of Kafka. The system evaluates the producer throughput for measuring the performance of the producer side. Defining the messages as the batch-based scheme effects on measuring producer throughput. It is easy to get good throughput in MB per second if the messages are large but much harder to get good throughput when the messages are small, as the overhead of processing each message dominates. For the analysis of the producer throughput experiment, the system tests the total published messages. Throughput measures how many messages arrive within a specific amount of time. A batch-based transmission strategy is typically used by Kafka producers, and throughput is determined by the number of messages stored in memory and the amount of time that Kafka processing takes. Based on the difference between the system's start and end times, the system determines the producer processing elapsed time. Equation (10) emphasizes the number of messages per second α, β represents total publishing message count and γ represents elapsed time of Kafka processing.

$$\alpha = \frac{\beta \times 1000}{\gamma} \quad ...(10)$$

Latency L measures how long it takes to process one event. Based on the start time T_s and end time T_e of Kafka processing, the system determines the latency. Equation (11) calculates the latency L in Kafka processing by using the latency calculation. The system defines the maximum latency among the number of latencies by using Math.max (max Latency).

$$L = T_e - T_s \quad ...(11)$$

Additionally, the system uses Kafka to measure user throughput. The difference between the start and end times of consumer processing is calculated by the system.

$$\theta = \frac{\lambda}{\delta} \quad ...(12)$$

Equation (12) describes the total consumed messages in one second. Consumer throughput θ calculates the total number of messages λ divided by the elapsed time δ in consumer processing.

The system examines reliability improvement based on the performance evaluation results of Apache Kafka. The system reliability can be defined as the time it takes for a system to operate correctly before it breaks down, without the requirement for maintenance or outside action to solve problems with the system. There are two types of reliability: serial reliability and parallel reliability. There are different actions depending on the processing of the system and deciding how reliable parts of the system are. The system emphasizes the parallel system reliability for processing many brokers in concurrently. Mathematically parallel system reliability is described as follows, with more components enhancing the system. Conceptually, in a parallel configuration the total system reliability is higher than the reliability of any single system component. Equation (13) describes 'n' parallel components including the system processing and R_s represents the reliability of the system.

$$R_s = 1-(1-R_1)\times(1-R_2)\times\ldots\times(1-R_n) \quad ...(13)$$

The system considers the parameters based on the reliability factors of Apache Kafka. The parameters of parallel system reliability calculated depend on the performance evaluation results of the test cases. The system decides the whole system's reliability by evaluating the parallel system reliability.

4. Experimental Setup and Performance Evaluation

In this section, the system the system measures the performance evaluation to prove the reliability improvement of Apache Kafka. The experimental setup for performance evaluation of original Apache Kafka, Apache Kafka with CIMLA, the total overhead cost and reliability factors are explained. The system establishes the configuration parameters for evaluating efficient message delivery processing in Apache Kafka. Performance evaluations are conducted to prove that Apache Kafka with CIMLA can improve the message delivery processing than the original Apache Kafka.

Then, the mathematical equations of defining checkpoint interval and reliability factors are explored in this section. The system measures the total overhead cost and reliability by applying CIMLA. The modified system reduces the total overhead cost than the default system. As the result of performance evaluation, the system proves the reliability improvement of Apache Kafka by measuring the reliability factors.

4.1. Experimental Setup

In Table 3, the performance of the real-time big data pipeline has been evaluated on the Window 10 Enterprise. The system tested five local servers and two types of datasets which focus on the effect of the messages. The system validates the performance investigation of equal data size on spam messages and various data sizes on tweets. of each partition. The system extends to 8 GB of the RAM memory for running many partitions and batch sizes.

Table 4 shows the required parameters for testing the message delivery flow of the modified system. The system identifies the client port which the clients will connect to the zookeeper, the id of the broker and the port number in the socket server. This must be set to a unique integer for each broker. The system defines the list of directories under which the log files are stored. The system identifies replication factors for the number of copies.

4.2. Key Parameters for Coordinate Check Pointing and Reliability Measurement

In this section, the system describes the key parameter for checkpoint interval message logging algorithm and values for these parameters. These parameters are assumed from the default values of Kafka processing. The

Table 3: Environment Setting for Performance Evaluation

Platform	Specification
Hardware	RAM 8.00 GB, Hard Disk 2TB, Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
Software	OS: Windows 64-bit Operating system Message System: Apache Kafka_2.11-0.9.0.0 Storage System: Zookeeper 3.4.7 Java 1.8
Dataset	Mobile phone spam messages from a Kaggle Web site and messages (tweets, news, blog, swiftey) form twitter

Table 4: Parameters Setting for Performance Evaluation

Local Servers		No of Partitions (Optimal)	Replication Factors	No of Batch Size (Files)	No of Partition
Zookeeper server (Local host: 2181)	Broker ID 1 (Localhost: 9092)	40	5, 4, 3, 2, 1	40, 50	1
	Broker ID 2 (Localhost: 9093)	40	5, 4, 3, 2, 1	40, 50	1
	Broker ID 3 (Localhost: 9094)	40	5, 4, 3, 2, 1	40, 50	1
	Broker ID 4 (Localhost: 9095)	40	5, 4, 3, 2, 1	40, 50	1
	Broker ID 5 (Localhost: 9096)	40	5, 4, 3, 2, 1	40, 50	1

main values are calculated based on the actual running time in Kafka processing. Firstly, the system calculates the time between failure based on uptime and downtime and number of breakdowns that depend on the publishing batch size (file size). After calculating the MTBF, the system calculates the optimal checkpoint interval, the default values and MTBF value. The system estimates the number of checkpoint intervals in processing according to Equation 1 to Equation 4. The above equations can vary based on the publishing file size (batch size). And then, the system considers the key parameters of the total overhead cost for the real-time messaging system with CIMLA. In Equation 5 to Equation 8, the system assumes the save time (T_{sd}) 5 seconds from the default save time from the original Apache Kafka and accepts failure time (T_{if}) from dynamic server failure time in the running process. Restart time (T_{rt}) assume the default restart time 5 seconds from the default Apache Kafka. The number of checkpoints in i^{th} cycle (N_i), checkpointing cost in i^{th} cycle (CC_i) and rollback recovery in i^{th} cycle (Rb_i) accept from the calculation of above equations. The system assumes the parameters of the parallel system reliability based on Test Cases. The system demonstrates key parameters and symbols for measuring the system reliability in Table 5.

Table 5: Key Parameters for System Reliability

Key Parameters	Symbols
System Reliability	R_S
Reliability of Dataset Sizes in Test Case 1	R_{DS1}, R_{DS2}, R_{DS3}, R_{DS4}, R_{DS5}
Reliability of Number of Server Failures in Test Case 2	R_{SF1}, R_{SF2}, R_{SF3}, R_{SF4}

4.3. Experimental Results for Test Cases

The system provides the producer client that accepts input from the user and publishes them as a message to

the Kafka cluster. The system arranges 40 files divided the equal size of mobile phone spam messages. Firstly, the system loads the message files from the user as soon as the system calculates the upload time of each file and calculates the MTBF. The producer defines optimal checkpoint intervals for message delivery processing in Kafka. Then, the system publishes the messages and receives from the consumer continuously. The system counts the messages and records the start time and end time of the process. The system catches up the failure time and failure messages with id in the running process of Kafka. After extracting the number of lost messages in the running state, the system calculates the reliability factors for the method and the whole system.

The system compares how well the original Kafka (Default system) and Kafka with CIMLA (Modified system) recovers lost messages. Next, using a batch-based strategy, the system examines asynchronous situations involving different file sizes. The system evaluates the comparison of lost messages and the comparison of optimal checkpoint intervals between the default system and the modified system. The system experiments with the reliability factors the total overhead cost, producer throughput, latency and consumer throughput for all Test Cases and compares the difference between the default and modified system.

In Table 6, real-time big data pipeline analytics is implemented on two Test Cases and performance evaluations are handled on different broker failures and data sizes for one topic.

Table 6: Parameters Setting for Test Cases

No	Test Cases	Number of Brokers	Number of Partitions	Number of Batch Size (File Size)	Dataset Size
1	Test Case 1	5	40	50	230 MB, 450 MB, 550 MB, 700 MB, 900 MB, 11 GB
2	Test Case 2	5(SF1, SF2, SF3, SF4)	40	40	500 MB

4.3.1. Experimental Studies on Test Case 1

In Test Case 1, there are six equal data sizes in phone spam messages 230 MB, 450 MB, 550 MB, 700 MB, 900 MB and 11 GB data sizes which are named DS1, DS2, DS3, DS4, DS5 and DS6 in the system. The system achieves the fault tolerance of Apache Kafka by recovering the message loss problem. Figure 7 demonstrates the message recovery by comparing the default system and modified system for Test Case 1. In Figure 7, the system pointed out the number of lost messages on the difference between the default system and the modified system.

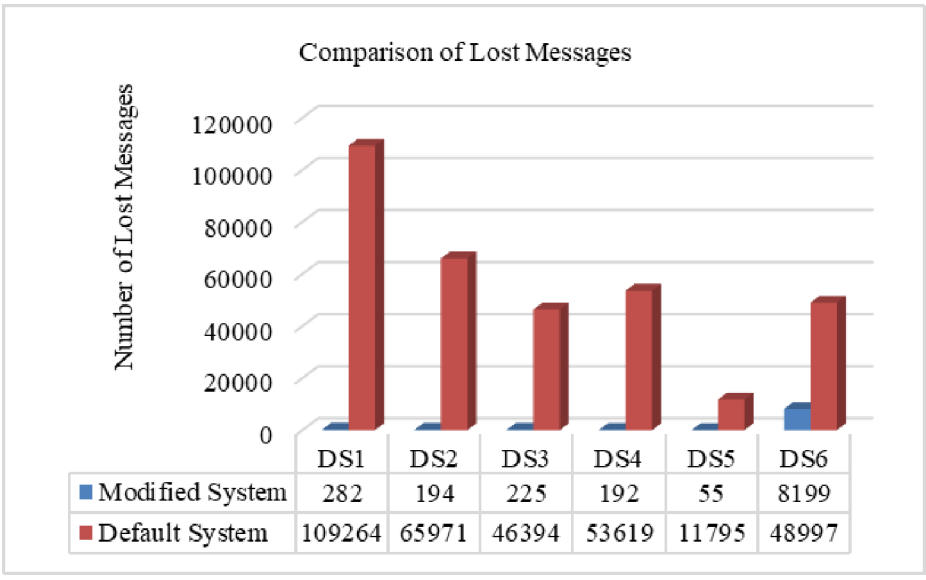

Figure 7: Comparison of Lost Messages for Test Case 1

The modified system can improve the message recovery in any publishing message size with five replication factors. Both systems can handle the message loss rate in DS5 by defining maximum replication factors. The modified system substantially decreases the number of lost messages compared to the default system as the strength of the optimal checkpoint interval. According to the comparison, the modified system significantly recovers the message loss problem compared to the default system.

The system demonstrates the difference between checkpoint interval time in the Default system and the Modified system in Figure 8. As the Default system defines the checkpoint interval of 60 seconds for any data size, the default system occurs in time consuming problems and message check problems.

Figure 8 emphasizes the optimal checkpoint interval in the modified system which alters publishing the number of batch sizes and the probability of mean time between failure occurring in messages. Figure 8 mentions the strong point of optimal checkpoint interval calculation in the modified system. As the modified system decides on the optimal checkpoint interval, it reduces the time complexity of the system. There is highly different optimal checkpoint interval, so the system checks the message delivery process and reduces the total overhead time. Figures 7 and 8 explain the improvement of the modified system by applying the CIMLA algorithm in the system.

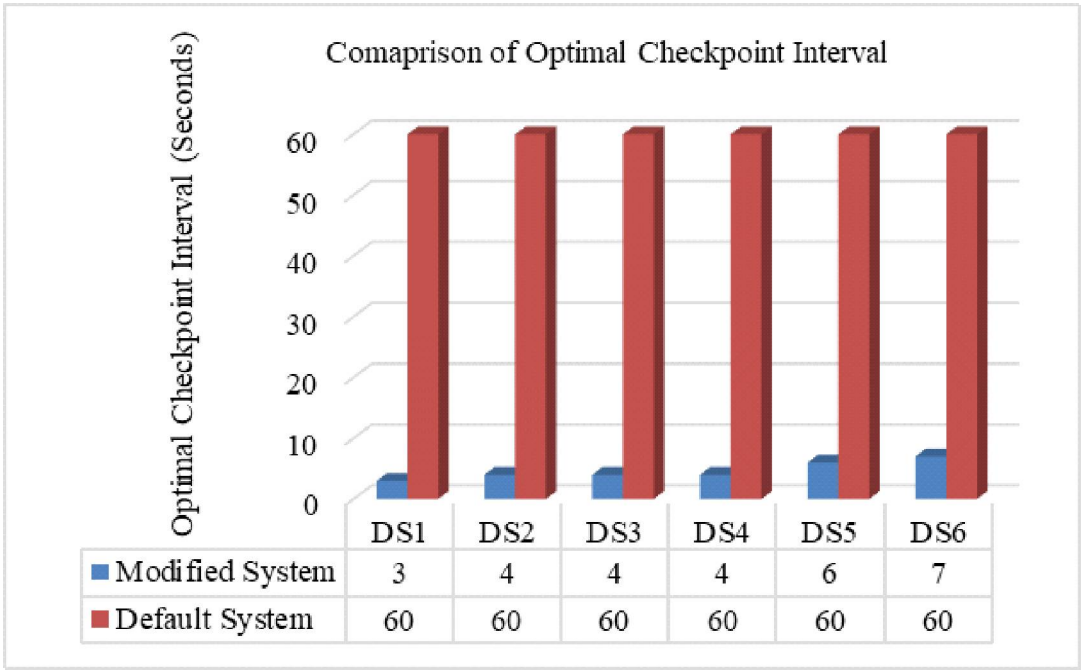

Figure 8: Comparison of Checkpoint Interval Time Test Case 1

Figure 9 illustrates the total overhead cost on various dataset sizes by comparing the default and the modified system. According to different dataset sizes in Test Case 1, the system defines the DS1, DS2, DS3, DS4, DS5 and DS6 as input dataset sizes respectively. Test Case 1 considers 50 batch sizes, five servers and five replication factors for all experiments. In the case of evaluation on total overhead cost in Figure 9, the total overhead costs of both systems depend on the server failure time. According to the experimental results, the modified system can adjust the total overhead time to the default system. There is a significant difference in the total overhead cost for the default system and modified system. The modified system can provide better performance, although server failure occurs in processing. According to the experimental evaluation, the total overhead cost of the modified system is slightly lower than the default system. Figure 9 proved the improvement of reliability of the modified system for any data sizes.

Figure 10 illustrates the comparison of the modified system and default system on five types of data sizes. The system measures the producer throughput by measuring in both systems asynchronously flush messages to its persistence store.

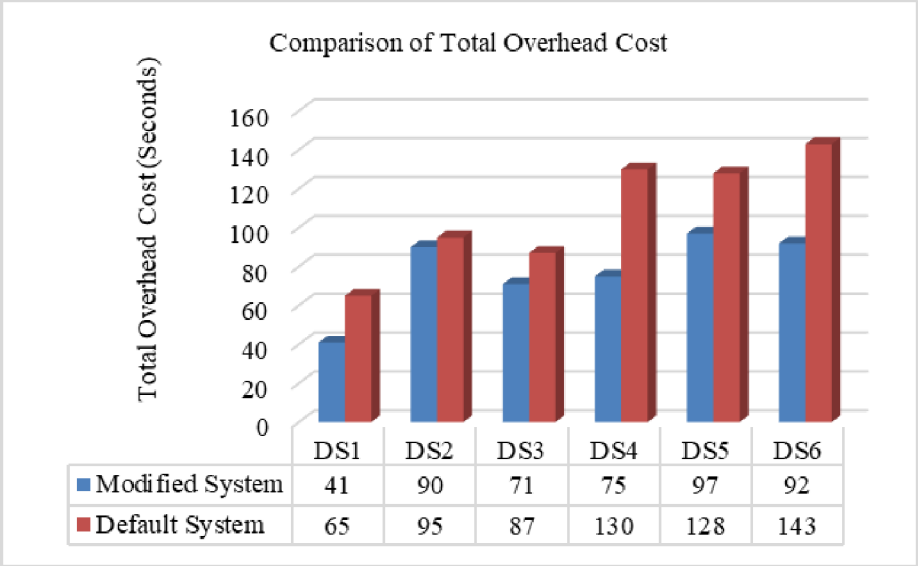

Figure 9: Comparison of Total Overhead Cost for Test Case 1

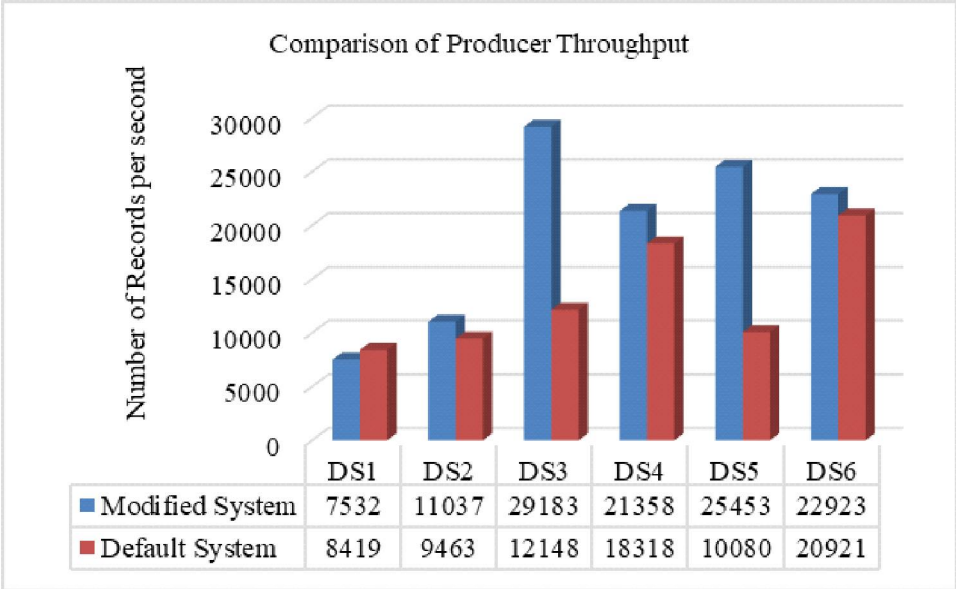

Figure 10: Comparison of Producer Throughput for Test Case 1

Figure 10 dictates the number of fewer than five servers for DS1(230 MB) because of decreasing the producer's performance of the modified system. As the experimental other results, the modified system is about two times better records per second in more data size than the default system. According to the comparison, the modified system improves the producer throughput of more data sizes than the default system.

Then, Figure 11 compares the maximum latency of the system between the default system and the modified system. The message recovery process affects the producer latency of the system. Therefore, the system estimates the maximum latency of five types of data sizes tested on five servers. The producer throughput illustrated in Figure 10 is directly proportional to the producer latency in Figure 11.

The modified system decreases the maximum latency is nearly two times in more data sizes than the default system. The latency of the modified system takes 39 seconds more than the default system. Although there are significant differences in maximum latency in more data sizes, the latency in DS1 has a slight gap

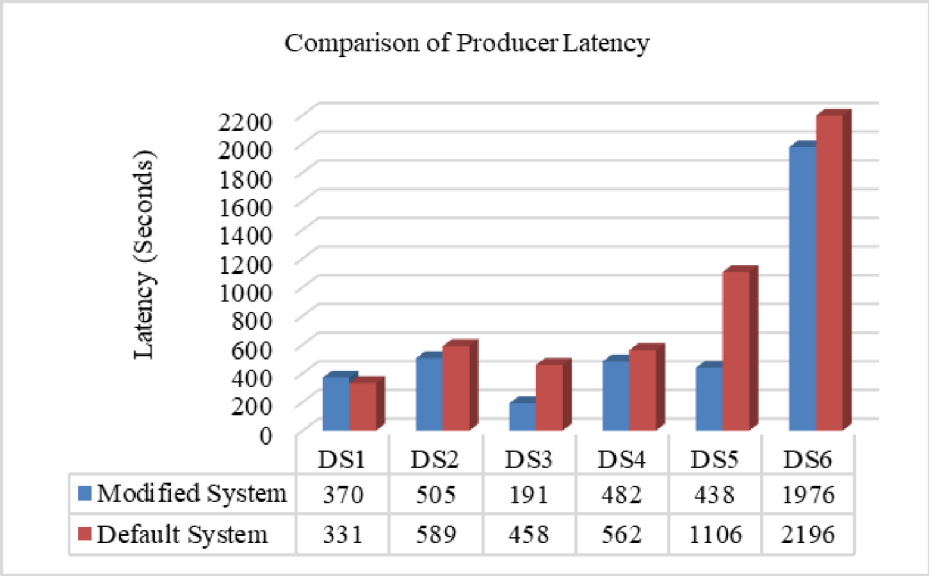

Figure 11: Comparison of Producer Latency for Test Case 1

between the default system and the modified system. According to the fact, the less dataset size is directly proportional to the less server. The modified system handles the latency of more data sizes significantly. By confirming comparative analysis, the modified system improves the latency than the default system.

In Figure 12, the system measures consumer throughput by using the consumer performance tools in Kafka. The system analyzes the difference in consumer throughput in various data sizes and compares the default and the modified systems on these dataset sizes. Figure 12 explains the comparison of the default system and modified system by comparing the records per second on the consumer side of Kafka. According to the results, the consumer throughput of the modified system significantly improves than the default system. By confirming the experimental results, the modified system increases the consumer throughput by approximately more records per second for different data sizes.

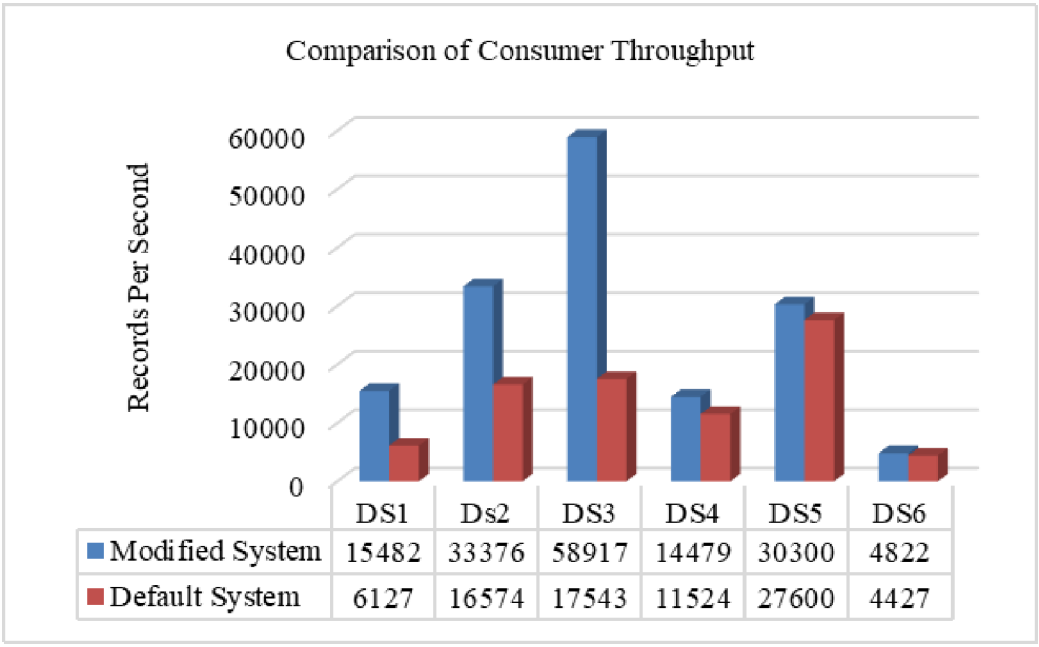

Figure 12: Comparison of Consumer Throughput for Test Case 1

4.3.2. Experimental Studies on Test Case 2

In Test Case 2, the system demonstrates 500 MB with various servers in Figure 13. Test Case 2 analyzed the comparison of one server failure, two server failures, three server failures and the four server failures which represent SF1, SF2, SF3 and SF4 respectively. The system intends to make a server failure case, so the system defines a maximum of five servers.

Figure 13 pointed out messages recovery by comparing the number of lost messages on the default system and modified system for Test Case 2. In Figure13, the system analyzed the server failure case by performing parallel running of many servers and testing on the same partitions on the same batch sizes. According to the comparison of server failures, the modified system decided to use five servers for data sizes 500 MB.As shown in Figure 13, the strength of the modified system efficiently recovers the number of lost messages compared to the default system. By the experimental results, the modified system can handle the number of server failure based on the same dataset size and improve the message recovery by 500 MB.

In Figure14, the system measures the total overhead cost of Test Case 2 by comparing the default system and modified system on various servers. Figure 14 demonstrates the difference in total overhead time on different server failures.

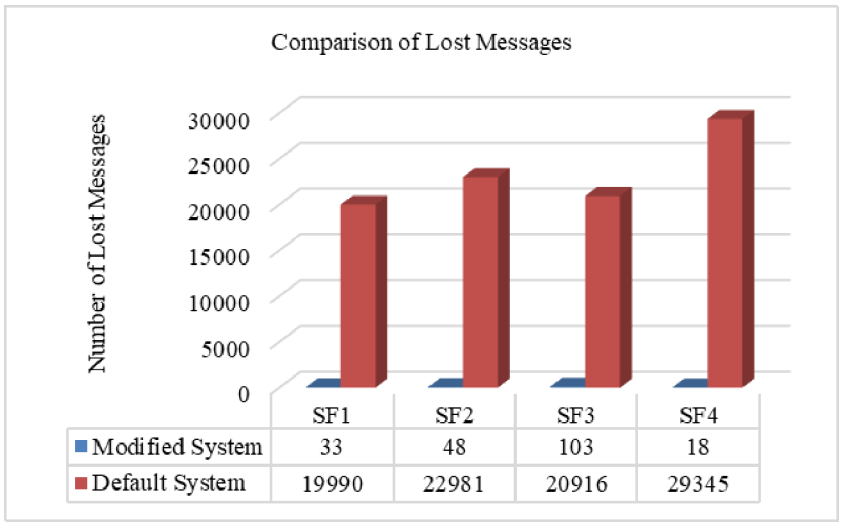

Figure 13: Comparison of Lost Messages for Test Case 2

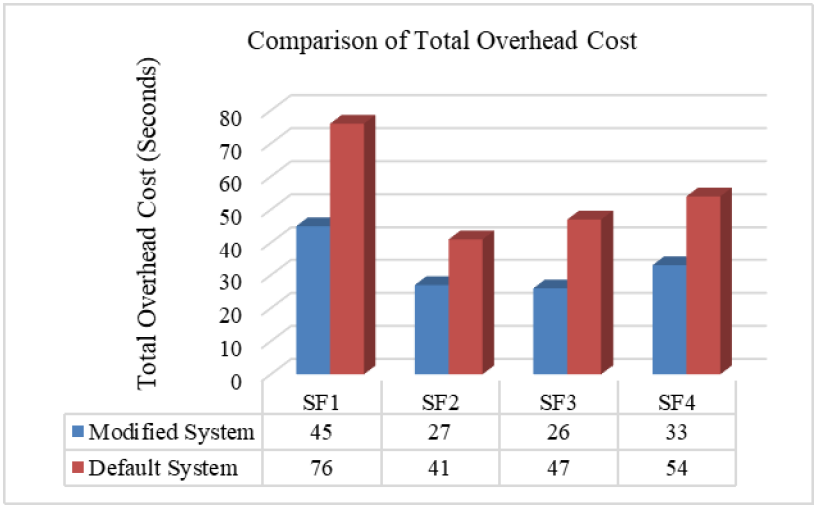

Figure 14: Comparison of Total Overhead Cost for Test Case 2

As the experimental results, the total overhead cost of the modified system recovers more than the default system. The modified system reduces the total overhead cost by nearly 50% of the default system.

Then, the system tested the producer throughput between the default system and the modified system.

Figure 15 explains the comparison of the producer throughput among different server failures. According to the experimental results, the system pointed out that less server failure more handle for data size 500 MB. The producer throughput of the modified system increases more records per second than the default system. As the comparison results, the modified system can improve the producer throughput on any number of server failures for data size 500 MB. According to the producer throughput comparisons, the modified system affects the decision of any server failure and replication factors usages.

And then, the system measures the producer's latency on various server failures in Figure 16. The system analyzed the comparison of the default system and modified system. The modified system less reduces the latency than the default system in any different server failure cases. The latency of the modified system improves on any server failure for dataset size 500 MB. According to the experimental results, the system points out the different number of server failures affected by the data sizes 500 MB.

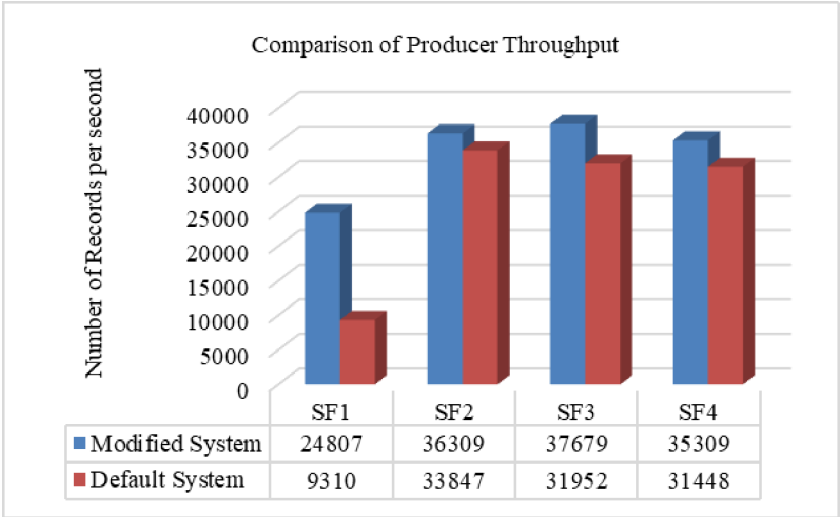

Figure 15: Comparison of Producer Throughput for Test Case 2

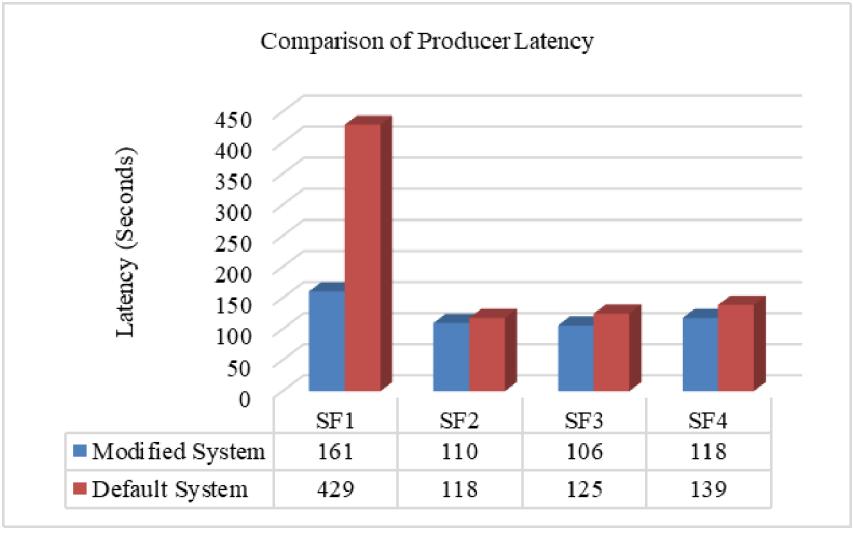

Figure 16: Comparison of Producer Latency for Test Case 2

In Figure17, the system measures consumer throughput testing on various servers by comparing the default system and modified system. The consumer throughput of modified system increases more records per second than the default system. By comparative analysis, the modified system proves the effectiveness of consumer throughput by testing on different server failures.

The system analyzes the batch sizes, various server failures, various data sizes and methods. To summarize the Experimental results, the system emphasizes the strong point of the modified system.

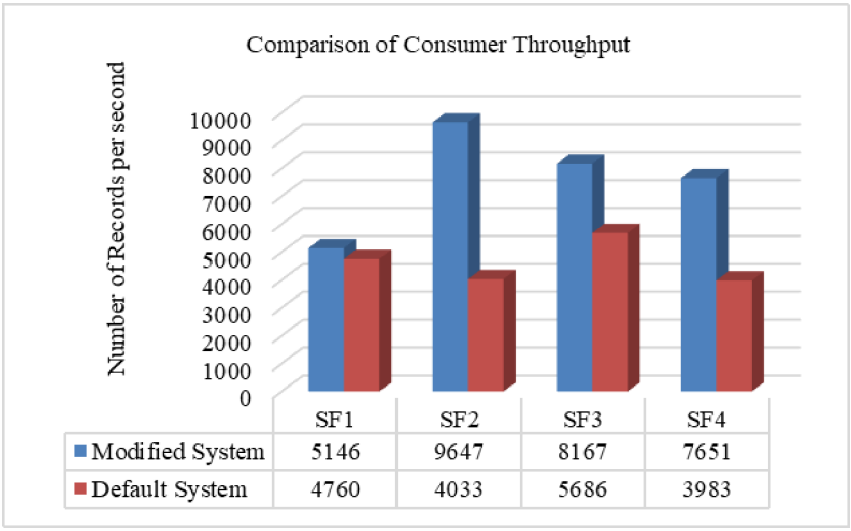

Figure 17: Comparison of Consumer Throughput for Test Case 2

5. Reliability Enhancement of Experimental Results

The experimental results for Test Cases of modified Apache Kafka and original Apache Kafka showed as message recovery, checkpoint interval time, total overhead cost and reliability factors. The system determines the whole system reliability by considering the experimental results of Test Cases. From Table 7 to Table 9 summarize the experimental performance of the modified system based on five test cases and percentage results.

In Table 7, the system calculates the message loss rate based on the number of lost messages in Figure 7. As a result of message loss rate, the modified system provides nearly 100% message recovery for DS1, DS2, DS3, DS4, DS5 and DS6 compared to the default system.

In Table 7, the system compares the checkpoint interval of the default and the modified systems based on Figure 8. Table 7 indicates that the modified system's optimal checkpoint intervals differ depending on the size of the dataset. The optimal checkpoint interval time of the modified system decreases 95% for DS1, 93%

Table 7: Message Recovery and Checkpoint Interval for Test Case 1

Dataset Size	Message Loss Rate (Default System %)	Message Loss Rate (Modified System %)	Checkpoint Interval (Default System)	Checkpoint Interval (Modified System)
DS1	3.92%	0.010%	60s	3s
DS2	1.18%	0.003%	60s	4s
DS3	0.45%	0.002%	60s	4s
DS4	0.01%	0.001%	60s	4s
DS5	0.02%	0.000%	60s	6s
DS6	0.10%	0.000%	60s	7s

for DS2, DS3, DS4, 90% for DS5 and 89% for DS6 than the default checkpoint interval time of the default system. The optimal checkpoint interval of the modified system affects the message recovery rate. Due to the optimal checkpoint interval calculation based on the dataset size, the modified system recovers the lost messages. As a result of Table 7, the modified system achieves the message recovery in any dataset size.

The system calculates the system reliability based on the results of Table 7. The message recovery of the system reliability improves nearly 100% in Test Case 1. The optimal checkpoint interval of the modified system increases 99% than the default system according to the results of Table 7. The results of system reliability prove the improvement of the whole system's reliability.

Table 8 compares the reliability performance on message recovery in CIMLA and the processing of Apache Kafka. Table 8 calculates reliability performance based on the experimental results of Figure 9, 10, 11 and 12. The total overhead cost of the modified system reduces 36% for DS1, 0.6% for DS2, 19% for DS3, 43% for DS4, 25% for DS5 and 36% for DS6 than the default system. This evidence indicates that the modified system is more reliable for message recovery than the default system. The parallel system's reliability of modified system enhances 87% than the default system according to the results of all total overhead cost.

Table 8: Reliability Performances for Test Case 1

Dataset Size	Total Overhead Cost	Producer Throughput	Consumer Throughput	Latency
DS1	36%	0%	61%	0%
DS2	0.6%	15%	51%	15%
DS3	19%	59%	71%	88%
DS4	43%	15%	21%	15%
DS5	25%	61%	9%	61%
DS6	36%	9%	9%	11%

According to the results, the producer throughput of the modified system is better performance than the default system. The producer throughput of DS1 has slightly decreased because of using more servers on less dataset size. The reprocessing time and the message recovery process affect the producer throughput measurement when the server fails. The producer throughput of the modified system increases 15%, 59%, 15%, 61% and 9% for DS2, DS3, DS4, DS5 and DS6 than the default system. According to the results of producer throughputs in Table 8, the system is 89% more reliable than the default system.

As the results of Table 8, the modified system has better consumer throughput for all dataset sizes than the default system. The consumer throughput of the modified increases 61% for DS1, 51% for DS2, 71% for DS3, 21% for DS4, 9% for DS5 and 9% for DS6 than the default system. According to the results of consumer throughput in Table 8, the system reliability increases by 96% than the default system.

The maximum latency of the modified system reduces 15%, 88%, 15%, 61%, 11% for DS2, DS3, DS4, DS5 and DS6 than the default system. Because more servers are used for smaller datasets, the modified system's maximum latency is higher than the default system in DS1. According to the latency of Table 8, the system's reliability is 96% better than the default system. The system proved reliability enhancement confirming the results of system reliability calculation in Test Case 1.

Table 9 demonstrates the experimental results of Figure 13, 14, 15, 16 and 17. Test Case 2 examines the reliability performance of Dataset 500 MB on the comparison of different server failures. In comparison to the default system, the modified system recovers nearly 100% of messages, according to experimental data. The modified system saves 59% better total overhead cost for SF1, 65% better total overhead cost for SF2, 55% better total overhead time for SF3 and 61% better total overhead cost for SF4 respectively.

The producer throughput of the modified system raises 73% more performance for SF1, 7% more performance for SF2, 16% more performance for SF3 and 11% more performance for SF4 than the default system, As the

results, the modified system significantly improves the producer throughput in one server failure (SF1). Compared to the default system, the modified system's consumer throughput rises by 8% records per second for SF1. By increasing more server failures SF2, SF3 and SF4 increases the modified system's consumer throughput by 59%, 31% and 48%, respectively. The latency of the modified system decreases by 37% seconds for SF1, 7% seconds for SF2, 16% seconds for SF3 and SF4 than the default system. The system determines the number of servers' failures based on a comparison of Table 9.

The system determines the reliability improvement of the whole system by calculating the message recovery. The message recovery of the modified system increases nearly 100% as the results of Test Case 2.

According to Table 9, the system's reliability enhances total overhead cost by 97%, producer throughput by 81%, consumer throughput by 86% and latency by 59%.

Table 9: Reliability Performance for Test Case 2

Number of Server (500 MB)	Message Loss Rate (Default System %)	Message Loss Rate (Modified System %)	Total Overhead Cost	Producer Throughput	Consumer Throughput	Latency
SF1	0.50%	0.0008%	59%	73%	8%	37%
SF2	0.57%	0.0012%	65%	7%	59%	7%
SF3	0.52%	0.0025%	55%	16%	31%	16%
SF4	0.73%	0.0004%	61%	11%	48%	16%

6. Conclusion

The IT infrastructure faces a major difficulty in providing reliable messages and a real-time messaging system across the business infrastructure. The development of Kafka's fault tolerance relies on reliable message delivery. The system focuses on the CIMLA (Checkpoint Interval Message Logging Algorithm) to be reliable message delivery of Apache Kafka. The checkpoint interval, recovering message loss problems and message recovery time and evaluating reliability improvement are highlighted in this dissertation. According to the investigation, the modified system greatly recovers the message loss problem compared to the default system. The evaluations prove that the modified system is better than the message recovery and reduces the time complexity of the system. As a result, the system points out that the modified system achieves message recovery in any number of server failures. The modified system significantly improves the default system according to the measurement of reliability factors. According to the evaluation analysis, the system improves the performance and enhances the reliability of real-time big data pipeline architecture.

7. Future Work and Limitation

The proposed algorithm can control the message delivery and enhance the fault tolerance occurred server failure cases. Broker failure scenario in message delivery is a limitation of this system. It can investigate more failure scenarios including the network failure for message delivery of real-time big data pipeline architecture. As the future suggestion of this work, the development of an innovative and optimized open-source framework: Apache Storm that improves the scalability of real-time big data pipeline architecture.

References

Andor Molnar. (2022). Apache Zookeeper. Retrieved from http://zookeeper .apache.org Andor Molnar on Aug 14, 2019

Daly, J. (2003). A Model for Predicting the Optimum Checkpoint Interval for Restart Dumps. In *International Conference on Computational Science*, June, 3-12, Springer, Berlin, Heidelberg.

Garg, N. (2013). Apache Kafka, 30-31, Packt Publishing, Birmingham, UK.

Kumar, A. *et al.* (2011). Fault Tolerance in Real Time Distributed System. *International Journal on Computer Science and Engineering*, 3(2), 933-939.

Liu, Y. *et al.* (2008). An Optimal Checkpoint/Restart Model for a Large Scale High Performance Computing System. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, April, 1-9, IEEE.

Rao, C.D.S. and Naidu, M.M. (2008). A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging. In *2008 IEEE/ACS International Conference on Computer Systems and Applications*, March, 444-447, IEEE.

Thein, K.M.M. *et al.* (2017). Security of Real-Time Big Data Analytics Pipeline. *MERAL Portal*.

Wu, H. (2019). Research Proposal: Reliability Evaluation of the Apache Kafka Streaming System. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, October, 112-113, IEEE.

Wu, H. *et al.* (2019). Trak: A Testing Tool for Studying the Reliability of Data Delivery in Apache Kafka. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, October, 394-397, IEEE.

Wu, H. *et al.* (2020). A Reactive Batching Strategy of Apache Kafka for Reliable Stream Processing in Real-Time. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, October, 207-217, IEEE.

Cite this article as: Thandar Aung and Aung Htein Maw (2026). Big Data Pipeline Architecture: Building the Reliability of Real-Time Messaging System on Apache Kafka. *International Journal of Data Science and Big Data Analytics*, 6(1), 32-54. doi: 10.67191/IJDSBDA.6.1.2026.32-54.