DYNAMIC BATCH SIZING FOR DISTRIBUTED DATA STREAM PROCESSING SYSTEMS

[1]Nwe Ni Hlaing

Faculty of Computing,
University of Computer Studies, Yangon
Myanmar
nwenihlaing@ucsy.edu.mm

[2]Si Si Mar Win

Faculty of Computer Science,
University of Computer Studies, Yangon
Myanmar
sisimarwin@ucsy.edu.mm

Abstract

In the data stream pipelines, variations of data stream ingestion rate and static batching are main challenges for processing data streams in real time. When the data stream ingestion rate and batch interval time are high, many events may be pushed into the channel, potentially exceeding transaction capacity, exhausting the agent's memory, and increased transmission time, and longer event queuing times in the channel. In this paper, a dynamic batch interval control algorithm is proposed to handle bursts in data volume, balancing the complex relationship between throughput, transmission time, and channel availability. The proposed method is implemented in Apache Flume for data ingestion and apply Spark Streaming for processing.The experiments indicate that the dynamic batching can prevent memory channel failures, reduce transmission time in data ingestion and processing time in processing. Moreover, the proposed approach minimizes the event queuing time in channel and maximizes the throughput with adjustable processing time for both ingestion and processing phases compared to static batching.

***Keywords*: Data ingestion; Dynamic batch sizing; Stream processing.**

1. Introduction

Nowadays, real-time processing of big data has become increasingly vital, along with technology advances. This leads to generation of vast amounts of data from diverse sources like the web, sensors, and social networking sites. Numerous applications are now requiring analysis of significant data streams to enable immediate actions based on results. In order to provide scalable, fast and resilient stream processing, effective data stream ingestion frameworks and data stream processing, frameworks are used for this purpose. State of the art data stream processing platforms, such Storm, Spark Streaming have received significant attention. Both the stream processing phase and the data stream ingestion phase play crucial roles in minimizing latency within data stream processing systems, thereby enabling faster decision-making. Data ingestion is the initial step in data stream processing system and it acts as a data collector for gathering data from the data producers such as social media, sensors and transmit it to the data consumers such as processing frameworks with high throughput and minimum transmission time. For this work, data stream ingestion framework such as Apache Flume [5] applied for this task.

Due to the fluctuation of data stream ingestion rate, it causes one main challenge for real-time data transfer. Currently, data ingestion methodology such as Apache Flume controls number of delivered events by using static batching technique with stable throughput. Each Flume agent is made up of three main components, the source, the channel and the sink. As a first component, the source is responsible for retrieving data into the flume agent and put these data into channel based on static batch interval time .Second component, channel is serving as a temporary buffer to store data received by the source until it's successfully written out by a sink. Third component, the sink, which extracts events from the channel with static batch size such as 100 in each transaction and directs them either to the topology or to storage systems such as HDFS and others or processing engines such as Spark Streaming.

The transaction capacity controls the maximum number of events that can be placed or retrieved within a transaction from a channel. Due to varying data stream ingestion rates, the number of records in each batch may differ even with the same batch size. High ingestion rates and long batch interval times result in a large number

of events being pushed into the channel at a high throughput or write rate. If the number of records exceeds the channel's transaction capacity, it can lead to channel failures, such as memory overflow and the inability to deliver data to processing engines. In order to solve this problem, channel transaction capacity value is increased by manual configuration and we need to re-run data ingestion component agent from beginning. If the data is time sensitive information, it will lose the data value and may lead data retransmission. If amount of received records in each batch is not exceeded channel transaction capacity, number of records in channel is very high and subsequent result is increasing transmission time and longer waiting time of event in channel. Increasing data ingestion transmission time affects increasing latency and reducing throughput of processing engine such as Spark Streaming .While data ingestion rate is low and small batch size, no of records in each batch is small and it decreases throughput and decreases transmission time and reduce queuing time of event in channel buffer.

Depending on variations of workload, this work aims to handle imbalance relationship between event write and read rates, achieve increasing actual number of drain events into processing engine, decrease complete transmission time in data ingestion stage and decrease channel fill percentage. It also guarantees to attain minimum processing time and high throughput in data processing phase in order to satisfy real time constraints. In order to tackle this problem, dynamic batch interval control algorithm considering the importance of variation of data stream rate and internal structure of channel capacity for applying data stream ingestion phase is proposed. Proposed dynamic batch interval control algorithm is implemented on Apache Flume, data stream ingestion and choose to use Spark Streaming as a processing framework for performing data stream processing job. The algorithm dynamically adjusts the batch size based on fluctuations in the data rate. Our proposed approach utilizes historical information from completed batches to estimate the optimal values for the next batch size .We compare performance of proposed method with default batch size in terms of metrics, such as average throughput between source and collector node, average channel fill percentage and average complete transmission time in milliseconds in data ingestion stage. In order to determine whether proposed algorithm applied in data ingestion stage is also effective for data stream processing phase, two performance metrics, throughput and processing time are also used.

2. Related Works

Across three stages of data stream processing, the critical role of data ingestion possess significant impact on the entire system to reduce end to end latency. Their research was focused on accurately ingesting data from different sources and integrating the resulting data streams into an analytic platform such as Apache Spark, Storm .This study describes the fundamental requirements of data stream ingestion systems and proposed a fault-tolerant and scalable framework for data stream ingestion and integration [3]. They recommend for a combination of Apache Nifi and Kafka, where Apache Nifi manages data stream acquisition, integration, and extraction, while Kafka functions as a message distribution system. This combined methods can serve as a reusable component across multiple feeds of both structured and unstructured input data within a given platform The research of this study does not take into account impact of fluctuation of data stream ingestion rate.

There is a growing interest in efficiently collecting real-time data streams, as shown by the development of various systems for this purpose. However, Apache Flume, a distributed framework created for this task, has limitations in load balancing and channel strategy [5]. The user manually sets the strategy and configuration for channel selection. Its native load balancing is too simplistic to effectively utilize hardware resources, and it lacks a mechanism to choose the appropriate channel based on channel states. To tackle these issues and enhance availability, their studies have introduced two dynamic strategies for load balancing and channel management. Initially, the researchers have integrated a master node to balance loads based on available memory, allowing collectors with more free memory to handle larger amounts of data. Additionally, we have suggested a mechanism to ensure optimal selection among memory channels and multiple file channels, along with transaction failover capabilities. In this study, it fails to consider the effects of fluctuations in batch size caused by shifts in data stream intake speeds. Moreover, there is still challenges that need to perform several optimizations such as considering the network distance between nodes when balancing the load. Additionally, the research of this study need to improve the selection of multiple file channels, as the current time-based method can cause in hash collisions and uneven data distribution. To address this issue, these researchers are still need to solve developing a dynamic algorithm that prioritizes channels with more data for deletion and less data.

In recent times, businesses have been faced with a large volume of streaming data that requires real-time processing. To handle this surge in data, a data stream processing framework called Spark Streaming has emerged. It is designed to handle real-time stream data analytics using a micro-batch approach. The unified programming model of Spark Streaming offers several advantages over traditional streaming systems, including quick recovery from failures, improved load balancing, and efficient resource usage. By treating the continuous stream as a series of micro-batches, Spark Streaming can continuously process these jobs. However, effectively processing these micro-batch jobs to achieve high throughput and low latency is a significant challenge due to the complex data dependencies and dynamic nature of streaming data. In a Spark Streaming system, fluctuations in data ingestion rates and cluster conditions can lead to challenging behavior. Maintaining a stable batch interval becomes

particularly difficult in such scenarios. Therefore, it is crucial to develop an adaptive approach that can dynamically adjust the batch size while meeting real-time constraints. Ideally, each batch size should ensure completion before the arrival of a new one. Due to variations in data stream ingestion rates, the relationship between throughput and transmission time, as well as the interplay between batch intervals and processing rates, can be complex.

To address this issue, our previous study described in [4] proposed an adaptive batch size tuning algorithm based on isotonic regression model that combines the model-independent expert fuzzy control (EFC) technique[7] with the A-scheduler. The EFC technique relies on two important parameters: workload fluctuation between time slot t and t-1 and the processing rate of the server system. By using the EFC technique, real-time decision-making based on server operations and historical data becomes possible, overcoming the difficulty of designing controls without precise performance models in complex micro-batch stream processing systems. However, we evaluated the performance of our algorithm in terms of only throughput and processing time in the Spark streaming processing engine, without stating the potential improvements in data ingestion performance. The algorithm also overlooked the importance of channel capacity and transaction capacity parameters, leading to memory channel availability issues and excessive queuing time of event. To fill the gap of previous work [4], this study focuses on minimizing the latency of the data processing phase and decreasing buffering time of events in channel to avoid channel fail problem.

In summary, the contribution of this work is three folds:

(i) Create a historical dataset using the collected transmission parameters from completed batches, applying various batch intervals to facilitate the prediction of future batch intervals.

(ii) Competitive, adjustable dynamic batch interval control algorithm that adjusts the length of batch intervals dynamically based on data ingestion rates, thereby optimizing the data stream ingestion layer.

(iii) Implement the proposed algorithm in Apache Flume and apply dynamic batch interval in both data stream ingestion and processing phases by increasing the throughput and decreasing the overall latency.

The rest of the paper is organized as follow: Section 2 presents the closely related work paper. Section 3 explains the architecture of the proposed system. Section 4 summarizes the results from our analysis and highlights the implications of our results. Conclusions are made in Section 5.

3. Proposed System Architecture

The typical structure of proposed system includes three layers such as a data stream access layer, data ingestion or data cache layer and stream processing layer. The system design is presented in Fig. 1.

3.1. *Data Access Layer*

The data access layer is in charge of handling the gathering and retrieval of external data, which includes transmitting data streams. The task of collecting source data from clients is assigned to Apache Flume. In this work, the pre-existing data is viewed as a stream as it undergoes ingestion via utilizing Exec Source from Apache

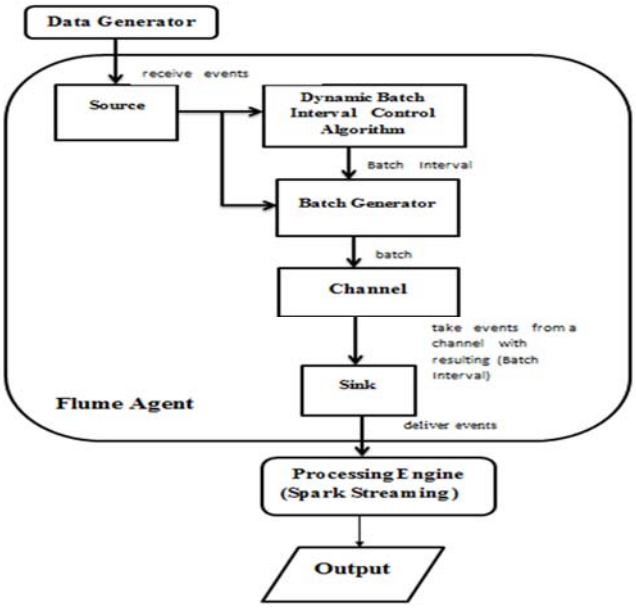

Fig. 1. Proposed System Design

Flume. Exec Source can monitor changes and anticipates a steady data output via stdout. Leveraging this feature, the system can generate a data stream from a pre-existing dataset, facilitating adaptable and timely data processing from Apache Flume. Exec Source can monitor changes and anticipates a steady data output via stdout. Leveraging this feature, the system can generate a data stream from a pre-existing dataset, facilitating adaptable and timely data processing.

3.2. *Data Ingestion Layer*

This layer is in charge of managing message queue and transmitting data streams. The middleware needs to cache the data stream in order to handle large amount of external data sources. Data Stream ingestion framework, Apache Flume is tasked for caching messages and handling message queues and transmitting data streams. Using a fixed batch size in the data ingestion layer, regardless of changes in data ingestion rate, can lead to issues like channel failures and complex relationships between throughput and transmission time. This paper introduces a new approach to adjust batch interval length based on fluctuation of data rate and to solve speed imbalance between data producers and data consumers. This approach is depicted in Algorithm (1).

3.2.1. Process Flow of Proposed Dynamic Batch Interval Control Algorithm

The process flow of proposed work is composed of three parts namely create data set D, building regression model and applied control method in Algorithm (1).

Algorithm 1. Proposed Data Ingestion Algorithm

Require: Gather all necessary data entries such as r_i,, ec, c, pc, tc, dc, t, etc. using various batch intervals and create dataset D.
Train regression model based on the dataset D.
Begin
Step 1: While the data input stream is NOT NULL
Step 2: e = read the current input stream
Step 3: i_b = DBIC (D, Model, e)
Step 4: startTime = currentTime;
Step 5: While (currentTime-startTime<ib)
5.1: Batch-up the input stream due to i_b.
Step 6: Send the batch to Channel buffer.
Step 7: GOTO step 1.
End

Algorithm 2. Proposed Dynamic Batch Interval Control (DBIC) Algorithm

Input: Data entry set D, Trained Regression Model, Transaction capacity t_c and current input entry e
Output: the best batch interval in milliseconds, i_b
Begin
Step 1: Initialize the channel fill %, chfill = 0%
i_b = Model (e) // predict the batch interval using current data entry
Step 2: S= RetrieveEntries (D, i_b)
//Adjust and Modify the batch interval depending on user defined transaction capacity

Step 3: $\overline{ec} = (\sum_{i=1}^{N} e_c)/N$ //calculate mean value of event receive count entries in S
Step 4: Calculate chfill using Eq. (2)
Step 5: // reduce the batch interval based on the ratio of transaction capacity and event received count
if tc < $\overline{ec}$ or chfill >50% then
5.1 $i_b = i_b * (t_c / \overline{ec})$
Step 6: return batch interval i_b
End

Build-Regression-Model
Begin
Input, pre-collected dataset D
Output: Regression model
Step 1: Read data dataset
Step 2: // Select 80% of the data from D to address the class imbalance issue
S = 80 % of data entries from D
Step 3: Train Regression Model on the entry set S
Step 4: return Regression Model

Function RetrieveEntries (D, r) // retrieve the data entries from pre-collected data D that match with batch interval values r
Step 1: $S = \emptyset$// Initialize the entry set S
Step 2: for each entry e in D
2.1: if batch interval value of D equals r then S = S ∪ {e}
end for
Step 3: if S is e*qual to* $\emptyset$ // if S is empty
3.1:Calculate σ(r) // Standard deviation of r
3.2: $r_l = r_i - \sigma(r)$ // lower bound of r
3.3: $r_h = r_i + \sigma(r)$ // upper bound of r
Step 4: // Retrieve data entries based on r_l and r_h
4.1 S_l = RetrieveEntries (r_l)
4.2:S_h = RetrieveEntries (r_h)
4.3: $S = S_l \cup S_h$
Step 5: return S. // return the entry set S

Our proposed algorithm is designed to learn from completed job statistics without prior knowledge of workload characteristics. To begin with, this algorithm gather all necessary parameters such as data ingestion rate r_i, number of events received ec, channel size c, number of event put success count pc, number of event take success count tc, number of event drain success count dc and complete transmission time t for various static batch interval range from 1100 milliseconds to 9000 milliseconds and create dataset named D.

The term "event receive" denotes the cumulative count of events received by the source up to the current moment. "channel size" indicates the total number of events presently within the channel. "event put success count" represents the total number of events successfully written and committed to the channel. "event take success count" signifies the total number of events successfully retrieved by the sink. "event drain success count" denotes the total number of events successfully transferred by the sink to storage or a processing engine. Lastly, "transmission time" refers to the duration required to transmit data in each batch from the source to the processing engine.

In this work, current data ingestion rate is a key parameter. The foundation of our algorithm lies in five regression analysis techniques, namely Isotonic regression (Iso), Linear regression (Linear),Pace (Partially Adaptive Competitive Equilibrium) regression, Support Vector Regression (SVR) and Multilayer Perceptron (MLP). In our algorithm, regression model is trained using entry set S resulting by selecting 80% of the data from D. While data input stream is not null, at first stage, read the current data entry such as ingestion rate, event received count, channel size of current data entry and then resulting data entry used for input in regression model for prediction of batch interval. In order to determine whether obtaining batch interval is optimized ,this study retrieves all relevant records that match with resulting batch answer and calculate mean value of event receive count in S. If no matches are found, we include entries with batch interval values within the standard deviation of batch interval values from D, both above and below the mean, and then calculate mean value of event receive count in S. If average number events available in this batch answer is greater than user defined transaction capacity or channel fill percentage is greater than 50% then reduce the batch interval based on ratio of transaction capacity and event received count.

The purpose of reducing the batch interval based on channel transaction capacity is to prevent channel failures and reduce event queuing time. The resulting batch interval limits the records sent to the channel, controlling the write rate from the Source. However, controlling only the write rate is insufficient. The read rate from the Sink is managed with a static batch size. Given the complex relationship between throughput, event drain success, transmission time, channel availability, and event queuing time, both write and read rates need adjustment. Therefore, our work also applies the predicted batch interval at the Sink level.

4. Experiments and Performance Evaluation

To conduct the ablation experiments, the proposed dynamic batch interval algorithm is implemented in Apache Flume with release version 1.11.0 and Spark Streaming. In this experiments, we simulate the fluctuation of data streaming arrival rate by controlling speed of writing data to csv file by generating random interval [1, 60] seconds and monitor incoming data in this file using Apache flume's Exec Source in real time nature with the amount of data size 6 Mb and 25 Mb. In Apache Flume's agent, transaction capacity 100000 and memory capacity 1000000 are used.

This work evaluates proposed control algorithm using two well-known data processing workload such as Word Count and stopword removal based on three metrics such as throughput, transmission time and channel fill percentage. Collector throughput is used to measure performance between collector node, Apache Flume and source node. Collector's throughput is measured the average number of events in a batch using Eq. (1). In this equation, "N" represent total number of batches. Transmission time is used to measure the average time each event from its arrival at channel to the time point when the event is transmitted successfully to the processing engine. Channel Fill percentage is used to measure availability of channel and this formula is described in Eq. (2). Lower channel fill percentage leads to improved system performance by reducing event queuing time in the channel , more free space available to maintain many events and preventing channel failure issues .An increase in the channel fill percentage raises the probability of facing channel fail exceptions and results in extended waiting times for channels in the buffer.

$$Throughput = \frac{\sum_{i=1}^{N} no\ of\ events\ in\ batch\ \ i}{N} \qquad (1)$$

$$ChannelFillPercentage = \frac{actual\ event\ in\ channel}{channel\ capacity} \times 100 \qquad (2)$$

In order to prove effectiveness of our proposed algorithm in data processing phase, two performance metrics, throughput and processing time are also used. Throughput in processing layer measures average number of records that is successfully processed in each batch. Processing time measure average time taken to process number of records in each batch.

Batch Interval Control Scheme	Data Ingestion			Data Stream Processing	
	Throughput (events/batch)	Transmission time	Channel Fill (%)	Throughput (records/ batch)	Processing Time(ms)
static (2000 ms)	7574	2014	6.11	2517	171
static (5000 ms)	11891	5348	52.73	1700	97
static (7000 ms)	11210	5844	63.46	1225	83
static (9000 ms)	25625	13763	54.61	1454	127
dynamic (ISO)	**36589**	**3138**	**3.53**	**2677**	**95**
dynamic (Linear)	**38468**	**3090**	**3.61**	**2968**	**90**
dynamic (MLP)	**38383**	**3181**	**3.67**	**3016**	**94**
dynamic (SVR)	**42341**	**3264**	4.05	**4040**	**109**
dynamic (Pace)	**37022**	**3162**	**3.52**	**3203**	**96**
dynamic (Iso) with static sink	37491	11486	3.76	2942	99
dynamic (Linear) with static sink	37252	11021	3.67	3006	94
Dynamic (MLP) with static sink	38812	9478	4.69	2419	95
Dynamic (SVR) with static sink	37817	10316	5.41	3007	101
Dynamic (Pace) with static sink	38794	7141	3.82	3388	96

Table 1.Average Performance Metrics under word count workload(1s)

The effectiveness of the proposed dynamic batch interval control algorithm is assessed by examining changes in data ingestion speeds with a constant batch size in Apache Flume. During the trials, constant batch time durations in milliseconds, like 2000, 5000, 7000 , 9000 milliseconds, are utilized. Moreover, this proposed dynamic batch interval control mechanism is evaluated with two options. The first option is that the proposed

batch interval control algorithm apply regression models such as (Iso), Dynamic(Linear), Dynamic(MLP), Dynamic(SVR), Dynamic(Pace) to guess batch interval on fluctuation of data ingestion rate and batch up incoming events based on resulting batch interval and then send these events to channel. Events that are currently arrived in channel are taken by sink component using resulting batch interval time in order to avoid channel full problem and increasing queuing time in channel buffer. The second option considers the dynamic scheme is apply in source and static scheme is applied at sink level. It means predicting batch interval time using proposed batch interval control method with five regression models and then sends incoming events to channel after waiting resulting batch interval time. Then current events available in channel are taken by sink using static default batch size 100.

In first experiment, under a StopWord removal using a batch interval time of 100 ms in Spark streaming, this study compares the performance statistics of our proposed dynamic batch interval control method with static batch interval times (2000 ms, 3000 ms, 5000 ms,7000 ms, and 9000 ms) , with the creation of data size amounting to 5 MB and random different write speeds to a CSV file locally, monitored using Apache Flume Exec Source. The impact of performance between data ingestion rate and static batch interval time are described in Fig.2,Fig.3,Fig.4 ,Fig.5 and Fig.6.It is evident no of events in each batch cannot be adjustable depending on fluctuation of data rate in static batching method. Our proposed dynamic batch interval method can adjust number of events in each batch depending on variation of data rate with high throughput in Fig.7, Fig.8,Fig.9,Fig.10 and Fig.11.

Performance comparison results of data ingestion phase under stop word removal workload in terms of average number of events, average number of transmission time and channel fill percentage are shown in Fig.12.In static batch interval(2000 ms,3000 ms,5000 ms,7000 ms and 9000 ms) ,it can be seen that high throughput in source node and Apache Flume can lead to maximum transmission time and increasing channel fill percentage. Low throughput in source node and Apache Flume can achieve minimum transmission time and decrease channel fill percentage. Minimum throughput is observed except for the static batch interval of 9000 ms when compared to other methods. Dynamic (Iso), Dynamic (Linear),Dynamic (Pace), Dynamic (SVR), Dynamic (MLP), and the dynamic batching technique with a static sink of 100 achieve higher throughput than the static batch intervals. Although the maximum throughput is found in Dynamic (Pace), the performance of dynamic batching depending on other regression models is also assumed to be effective. Static batch interval (9000 ms), Dynamic (SVR) with static sink 100 take the maximum transmission time with other approaches due to higher throughput conditions. Our desired condition that satisfies higher throughput and lower transmission time is the proposed dynamic batch interval with the Pace regression model using the resulting batch interval time at the sink side.

The channel fill percentage metric is highly correlated with throughput during the data ingestion stage. In Fig. 12, this study demonstrates that if the throughput between the source node and the collector node is higher, the channel fill percentage is also higher. In other words, this indicates that the queuing time of events in the channel is longer. The static batch interval time of 9000 ms consumes a higher channel fill percentage than other approaches. If the channel fill percentage is high, it may lead to the problem of channel memory becoming full. Consequently, this prevents uploading events available in each batch to the required channel. Optimized channel fill percentage with minimum transmission time can be observed in the dynamic batch method, particularly Dynamic (Pace).Therefore, our proposed dynamic batch interval control method with dynamic sink can adjust higher throughput, minimum transmission time and less channel Fill percentage.

Performance statistics in terms of throughput and processing time in the data processing stage are described in Fig.13.This experiments show that high throughput in processing engine lead to longer processing time and lower throughput attain minimum processing time in static batching technique. Dynamic (Pace) achieves the highest throughput compared to other static batch intervals and the proposed dynamic batch interval control algorithm with a static sink of 100.Our proposed dynamic batching method with dynamic sink adjust maximum throughput and minimum transmission time. Among them, the proposed dynamic batch interval control using dynamic sinks, such as Dynamic (Pace), is more efficient because it takes a small amount of time and achieves the highest throughput. Additionally, our proposed dynamic batch interval control algorithm utilizes dynamic batch results at both the source and sink levels, resulting in higher throughput, reduced transmission time, and increased free space in the channel buffer compared to using a static batch size of 100 at the sink side.

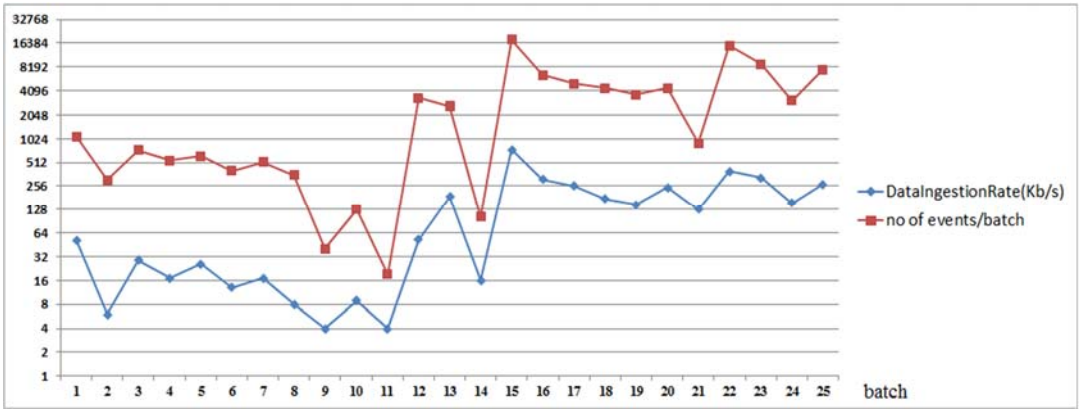

Fig. 2. Performance Analysis between data ingestion rate and static batch interval (2000 ms)

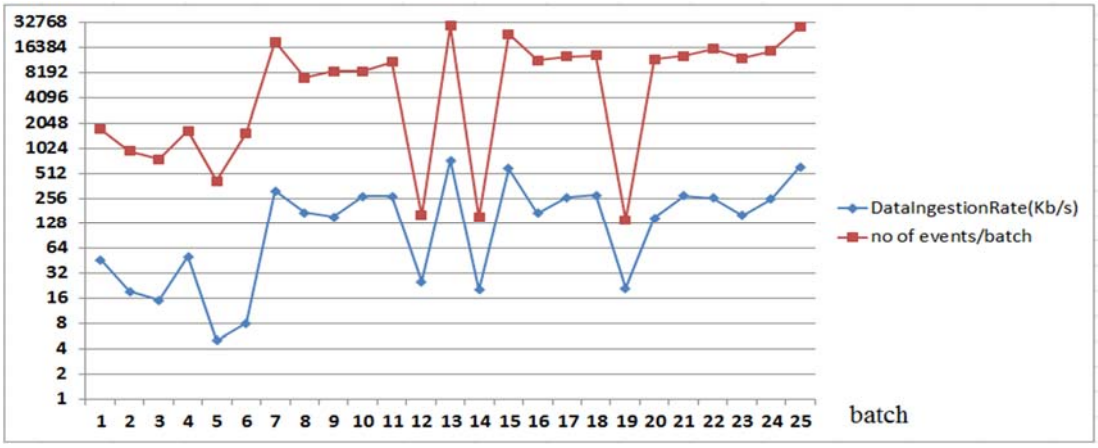

Fig.3. Performance Analysis between data ingestion rate and static batch interval (3000 ms)

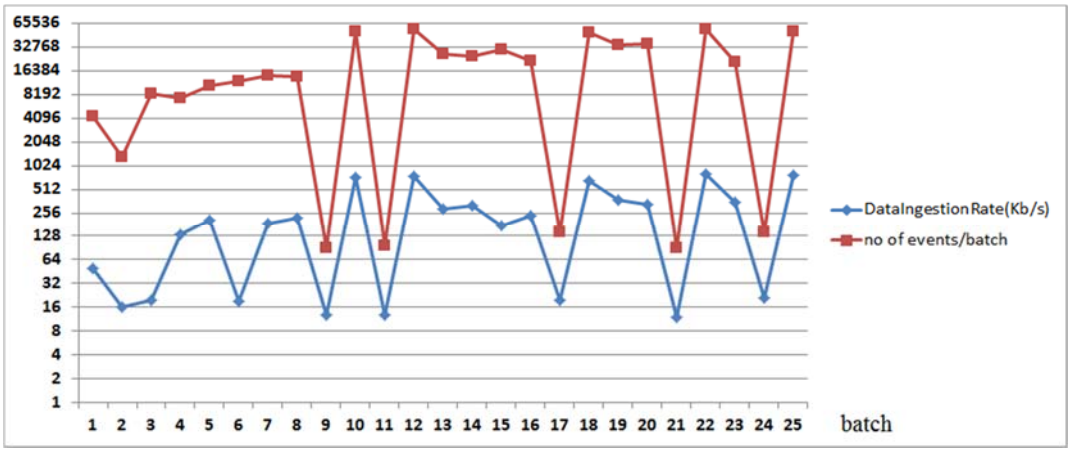

Fig.4. Performance Analysis between data ingestion rate and static batch interval (5000 ms)

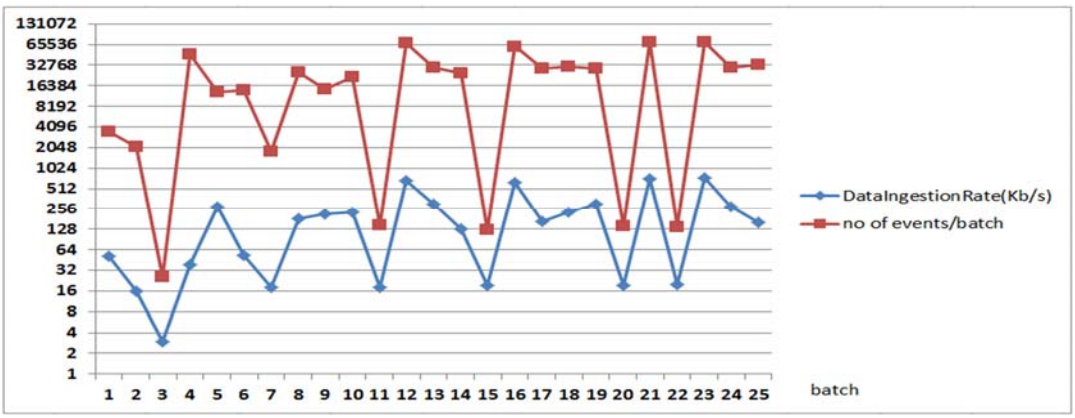

Fig.5. Performance Analysis between data ingestion rate and static batch interval (7000 ms)

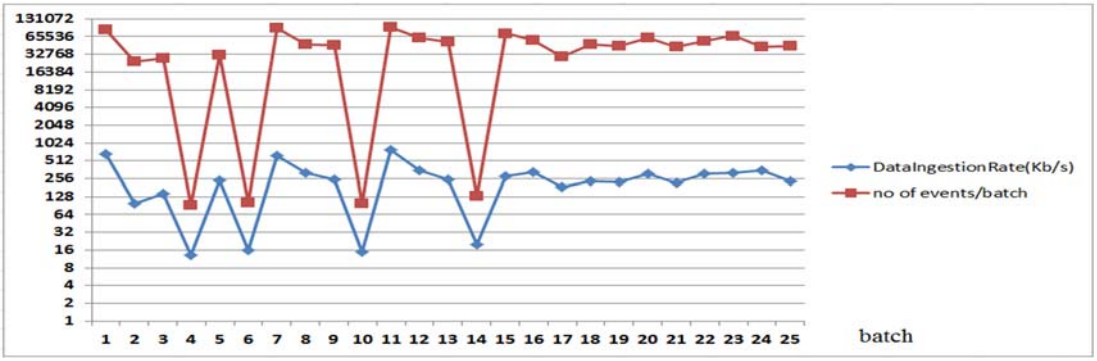

Fig.6. Performance Analysis between data ingestion rate and static batch interval (9000 ms)

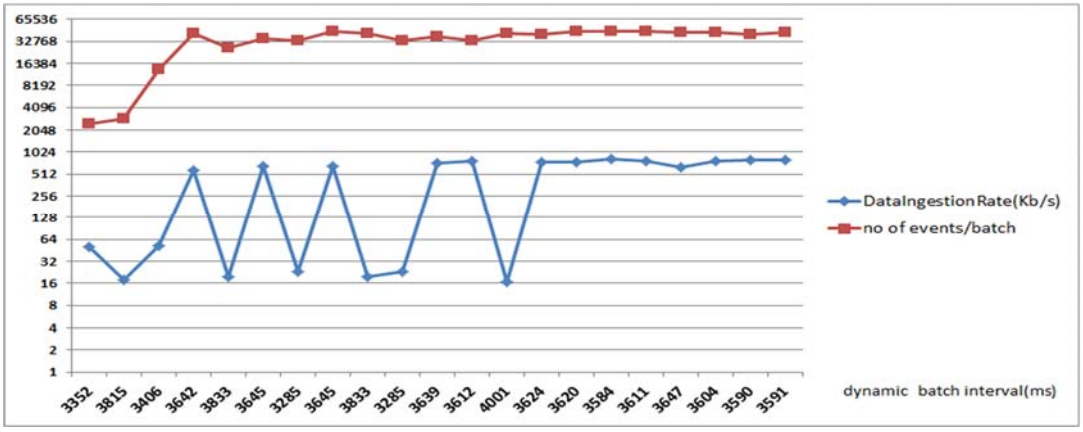

Fig.7. Performance Analysis between data inestion rate and dynamic batch interval (Iso)

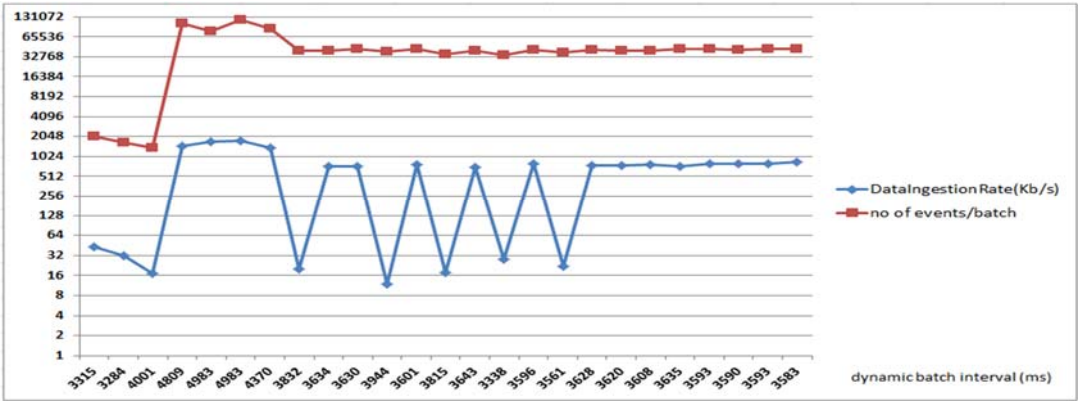

Fig. 8. Performance Analysis between data ingestion rate and dynamic batch interval (Linear)

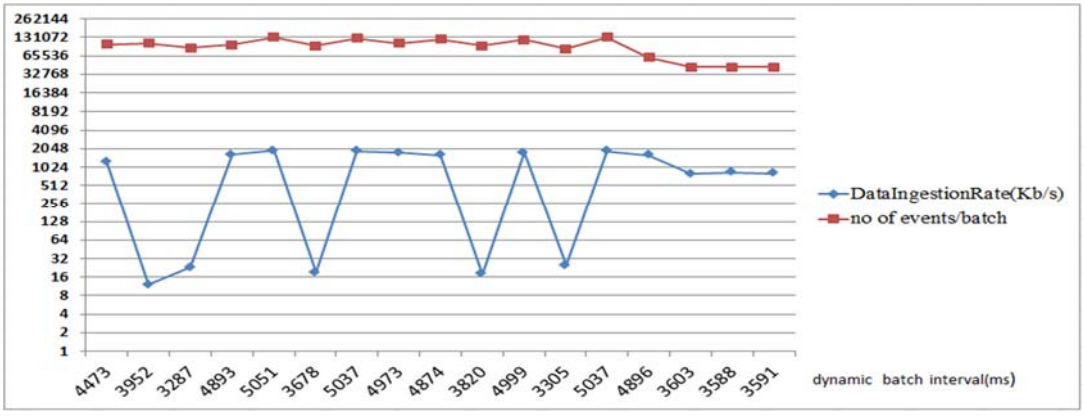

Fig. 9. Performance Analysis between data ingestion rate and dynamic batch interval (MLP)

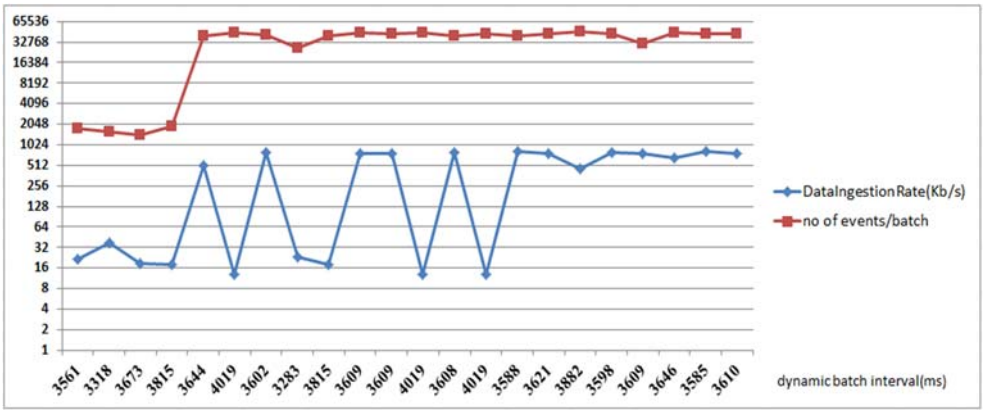

Fig. 10.Performance Analysis between data ingestion rate and dynamic batch interval (SVR)

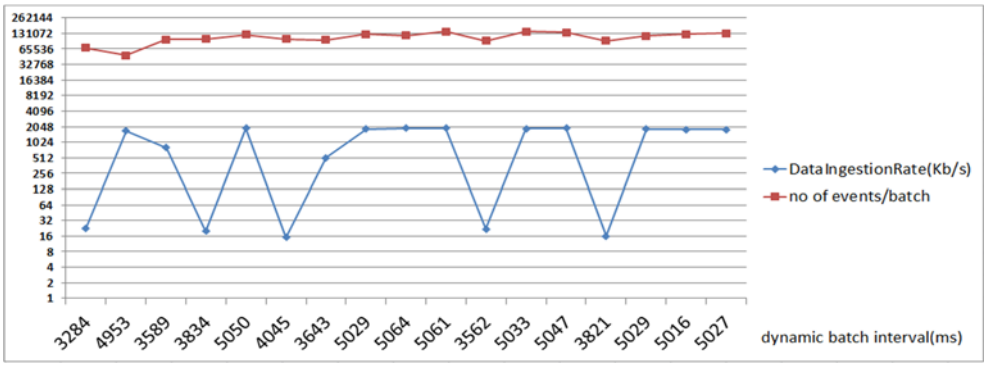

Fig. 11.Performance Analysis between data ingestion rate and dynamic batch interval (Pace)

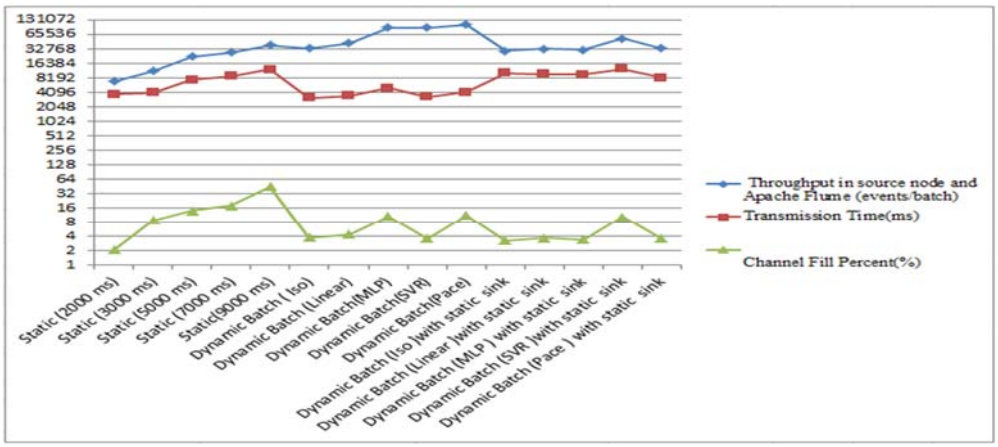

Fig. 12. Comparison Results in data ingestion under Stopword Removal workload (100 ms) in terms of throughput , transmission time and channel fill percentage

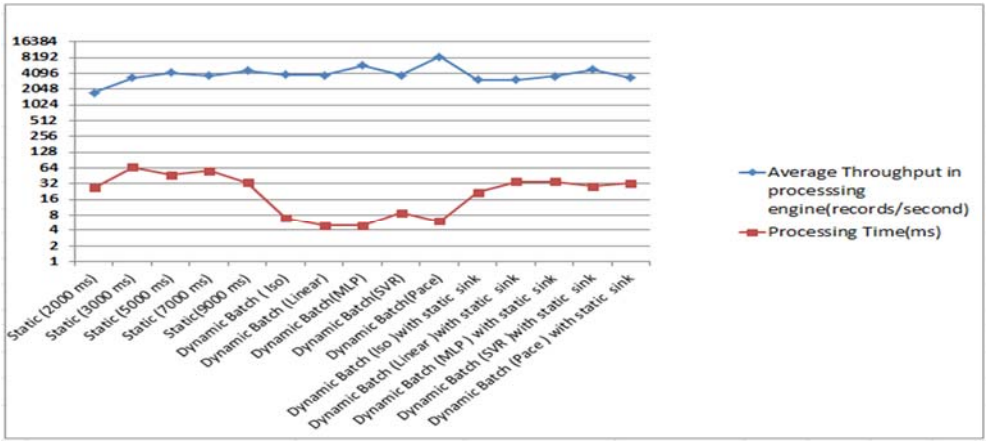

Fig. 13. Comparison Results in data processing under Stopword Removal workload (100 ms) in terms of throughput and processing time

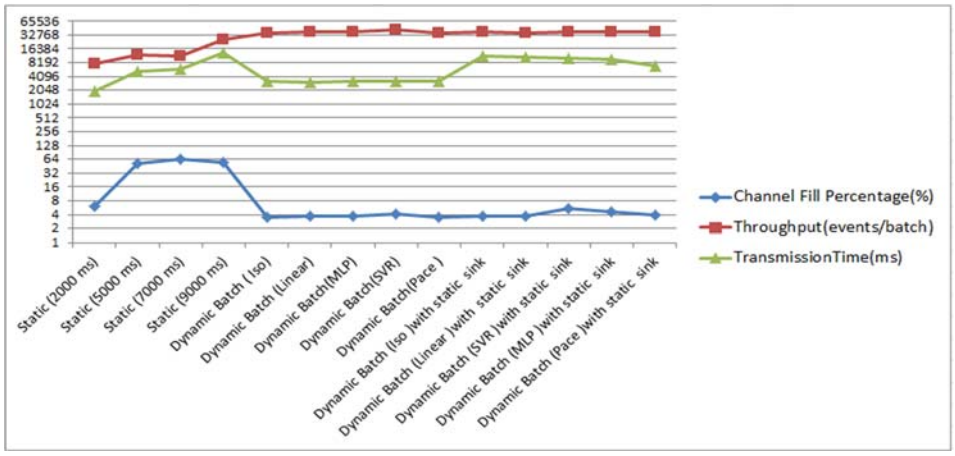

Fig. 14. Comparison Results in data ingestion under WordCount workload (1 s) in terms of throughput , transmission time and channel fill percentage

In second experiment, under a Word count workload using a batch interval time of 1 s in Spark streaming, this study compares the performance statistics of our proposed dynamic batch interval control method in data ingestion with static batch interval times (2000 ms5000 ms,7000 ms, and 9000 ms) with simulation of 24.5 Megabytes file by controlling write rate of data to Apache Flume Exec Source . In Table 1 and Fig.14, it is evident if amount of throughput is high, static batch method takes longer transmission time and increase channel fill percentage Small throughput decreases transmission time and decreases channel fill percentage. This experiments show that throughput of static batching techniques gain lower throughput than proposed dynamic methods using Iso, Linear, SVR, MLP, and Pace

When comparing throughput metrics of these dynamic methods with proposed dynamic batch interval control with static sink, the amount of throughput is slightly different because of applying same technique in data ingestion stage before grouping data, the highest throughput is achieved in Dynamic (SVR). Dynamic approach with a dynamic batch interval at the sink level achieves the minimum transmission time, outperforming the dynamic approach with a static batch size of 100. Although retrieving 100 events from the channel to the sink takes only a few milliseconds, retrieving all events in each batch consumes more transmission time, increasing waiting time in the channel buffer and leading to memory overflow issues. In Dynamic methods with static sink, it maintain to achieve high throughput but increased transmission time and increased channel fill percentage. Proposed method with dynamic sink can adjust to balance higher throughput along with minimum transmission time and minimize channel fill percentage especially Dynamic(SVR).

If amount of throughput in processing engine is high,it attain longer processing time and amount of throughput

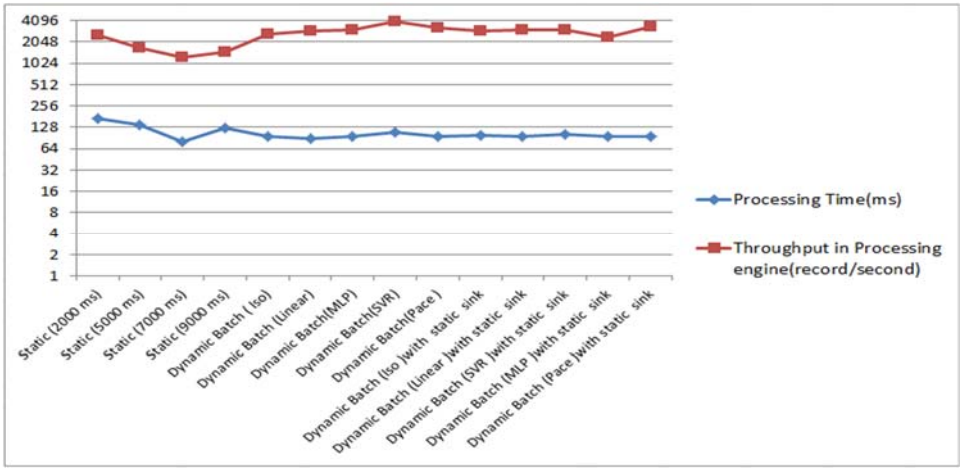

Fig. 15. Comparison Results in data processing under WordCount workload (1 s) in terms of throughput , processing time

is small,it get small processing time in static batch interval and this situation is occurred in Fig.15 and Table1.In proposed dynamic batching technique with dynamic sink and static sink adjust higher throughput and minimum

processing time in Spark streaming. Considering all important metrics in terms of throughput in data ingestion, transmission time, channel fill percentage, processing time, and throughput in the processing engine, the proposed dynamic batch interval using the resulting batch interval from channel to sink is more effective. Among these approaches, Dynamic (SVR) adjust better throughput, minimum transmission time, higher channel availability in data ingestion (Apache Flume), increased throughput with lower processing time in the stream processing phase.

5. Conclusion

Enhancing the efficiency of the data stream intake phase is crucial for data stream processing systems. This research introduces a dynamic batch interval control strategy that adjusts the batch interval based on changes in data stream intake rates. Our objective is to ensure peak system performance under varying demands, such as high data transfer rates, reduced transmission times, and prevention of channel buffer overflow.

By considering historical data rates and the internal structure of the data ingestion network, our algorithm accurately determines the optimal batch interval. It handles sudden increases in data volume, reduces transmission times, and minimizes channel buffer time and fullness. Implementing this method enhances data stream processing performance, increasing throughput and reducing processing times. Proposed regression-based algorithm outperforms the standard batch interval control approach in data ingestion, particularly under Word count and stop word removal tasks. Our proposed method, employing a dynamic batch interval control strategy, has the capability to adjust the number of events in each batch according to variation of data ingestion rate, the transmission speed, and the channel's utilization percentage and it can handle complex relationship between throughput, processing time and channel fill percentage.

Conflicts of Interest

The authors have no conflicts of interest to declare.

References

[1] Pal, G.; et.al. (2018): Big Data Real Time Ingestion and Machine Learning, 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP), Lviv, Ukraine, 2018, pp. 25-31.
[2] Tandel, U.; Deshmukh, S.;Sharma ,V.(2017): Spark Streaming through Dynamic Batch Sizing , International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), Volume 6, Issue 2, March - April 2017.
[3] Isah, H.; Zulkernine, F. (2018) : A Scalable and Robust Framework for Data Stream Ingestion, IEEE International Conference on Big Data (Big Data), 2018.
[4] Hlaing, N.; Win, S.(2024): Improvement of Data Stream Processing using Adaptive Ingestion , IEEE 21st International Conference on Computer Applications,Myanmar,2024.
[5] Shu, B.;Chen, H.; Sun, M.(2017) : Dynamic Load Balancing and Channel Strategy for Apache Flume Collecting Real-Time Data Stream, IEEE International Symposium on Parallel and Distributed Processing with Applications and IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC),2017.
[6] Liao ,X.; et.al.(2015): An enforcement of real time scheduling in Spark Streaming, 2015 Sixth International Green and Sustainable Computing Conference (IGSC), Las Vegas, NV, 2015, pp. 1-6.
[7] Cheng ,D.; et.al.(2018): Adaptive Scheduling Parallel Jobs with Dynamic Batching in Spark Streaming, IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 12, pp. 2672-2685.
[8] Li, W.;et.al.(2017): Wide-Area Spark Streaming: Automated Routing and Batch Sizing, 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, USA, 2017, pp. 33-38.
[9] Jun ,J.(2015): Distributed Data Processing Based on flume /Kafka/Spark,3rd International Conference on Mechantronics and Industrial Informatics(ICM 2015),2015.
[10] Marcu,O .; et. al.(2018): KerA: Scalable Data Ingestion for Stream Processing, IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2018, pp. 1480-1485.
[11] Hu, L.; Schwan , K.; Amur, H ;Chen, X .(2014):Elf: efficient lightweight fast stream processing at scale, Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference. USENIX Association,2014, pp. 25–36.
[12] Xu ,L.; Hu, H.; Ma ,Y. (2019):Improvement Design for Distributed Real-Time Stream Processing Systems, Journal of Electronic Science and Technology, VOL. 17, NO. 1,2019, pp. 3-12.
[13] Zhang ,X. –C .; Xiao, N.; Liu , F.; Yu ,S.(2016): “Review of memory file system,” Journal of Computer Research and development, vol. s2, 2016,pp. 9-17.
[14] Liu ,Q.; et.al.(2017): Architectural design of data stream-based big data real-time analysis system, Proceeding of Joint Intl. Information Technology, Mechanical & Electronic Engineering Conf, 2017,pp.153-156.
[15] Liu, F.(2016):Performance comparison and optimization scheme of message queue based on massive data ,Computer Engineering & Software, vol. 37, no. 10, 2016,pp. 33-37.
[16] Chen, R.; Lyu, Y.; Hou, B.(2015):Design and implementation of distributed log stream processing system based on Flume/Kafka/Spark, Proc. of ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2015,pp. 2323-2324.
[17] Hoffman, S.(2018) :Apache Flume: Distributed Log Collection for Hadoop, Birmingham: Packt Publishing Ltd., 2016.
[18] Dagdia, Z.; Zarges, C.; Beck, G.(2017): A distributed rough set theory based algorithm for an efficient big data pre-processing under the spark framework, Proc. of IEEE Intl. Conf. on Big Data, 2017 ,pp. 911-916.
[19] Sirisakdiwan , T.; Nupairoj, N.(2019):Spark Framework for Real-Time Analytic of Multiple Heterogeneous Data Streams, 2nd International Conference on Communication Engineering and Technology,2019
[20] Lee, C.;Paik, I. (2017):Stock market analysis from Twitter and news based on streaming big datainfrastructure, IEEE 8th International Conference on Awareness Science and Technology (iCAST),2017.
[21] Alwidian,J.;et.al.(2020): Big Data Ingestion and Preparation Tools, Modern Applied Science,Vol. 14, No. 9,2020.

Authors Profile

Nwe Ni Hlaing, holds a Master of Computer Science degree from University of Computer Studies in Yangon in 2017.She also received B.C.Sc and B.C.Sc(Hons:) from University of Computer Studies(Pyay) in 2013 and 2014.She is currently a research candidate at University of Computer Studies,Yangon.Her research includes machine learning, data mining and Big Data analysis.

Si Si Mar Win is currently a professor in Faculty of Computer Science, University of Computer Studies,Yangon and supervise doctoral students. She is interested in machine learning, data mining ,cloud computing and big data analytics.