International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Fuzzy-Based Event Clustering for Semantic Load Shedding of Real-Time Data Streaming

Shubham Vyas[1]*, Dr. Rajesh Kumar Tyagi[2], Dr. Shashank Sahu[3], Dr. Rajesh Kumar Tyagi[4]

Submitted: 01/10/2023 **Accepted :** 22/11/2023 **Accepted:** 02/12/2023

Abstract*:* In real-time data stream processing, load shedding is used to manage data overload. Fuzzy-based event grouping and load shedding optimize Apache Kafka's performance in this study. This study presents a hybrid load-shedding strategy with high recall rates that retains the throughput and cost models needed to calculate the value of matched events to shed. It also shows that deleting a constant fraction of input events can reduce latency without losing recall. The study also shows that state-based methods had the highest recall rates and input-based procedures the highest throughput. As time slices become more significant, the hybrid technique, which employs four or more slices, is best for high recall rates and acceptable throughput. These findings can enhance machine learning algorithms and load-shedding tactics for many applications. This study is dynamic and will test the method's flexibility by employing automated algorithms to determine the system's ideal sampling rate. Workload, data flow, and resources comprise this environment

Keywords: *Semantic Load Shedding, Fuzzy Clustering, K Nearest Neighbor (KNN), Apache Kafka, Real-Time Data Streaming.*

1. Introduction

Apache Kafka handles most real-time processing methods; it is common practice to use them in conjunction with a messaging system that allows for the entry and product of various stream continuity. To alleviate the starving issue, a load shedding (LS) strategy is required, restricting the amount of arriving information and keeping the system's performance stable, even when the system is very busy. LS is a technique created to mitigate the negative impacts generated by the high velocity and volume of the processing of big data streams. LS is a method used by stream processing systems to manage unanticipated spikes in the input load [1]. Apache Kafka is a framework for the real-time delivery of data streams operating as a distributed message queuing system. Because it uses a distributed processing technique, Kafka has the benefit of providing extremely massive data streams in a very short amount of time [2]. However, when the data explosion takes place, the message latency significantly rises, and the system may be disrupted. Compared to other systems of messaging working in log processing in real-time, Kafka has a greater output per second cycle [3]. Additionally, unlike other messaging systems, each consisting of a single cluster, Kafka is designed specifically for use in distributed settings, making it simple to extend into several clusters [4]. As an add-on, currently available systems keep data in memory, which results in a decrease in the system's speed [5].

1.1. Apache Kafka

In the traditional integration system, when there are multiple sources and target systems, creating integrations between them can lead to various difficulties, such as selecting protocols and handling different data formats like CSV, binary, hash, etc. However, Kafka's primary purpose is to perform analytics for streaming real-time data. It has more applications, including website traffic tracking, error recovery, and message narrating and replaying. Kafka is also known for being user-friendly and providing fast throughput and consistent replication [6].

Kafka employs a pull-based paradigm for clients to receive data whenever they need it, which leads to high throughput when the platform is used for big data streaming. Kafka's architecture includes various key components like producers, zookeepers, consumers, partitions, brokers, and logs [7]. Record streams are categorized into topics, and the topics are stored on drives as logs. The entries that form a topic log are arranged chronologically and stored in partitions that are numbered by offset [1]. These partitions are dispersed among many brokers to ensure high throughput. Consumers subscribe to topics of interest and obtain data using offsets. With the cooperation of consumer organizations, the strain on Kafka can be spread more evenly. Multiple consumers within a consumer group can obtain data in parallel from various partitions of a subject. To carry out this procedure, both the API Kafka consumer and Kafka producer are used. A zookeeper oversees coordinating the activities of all the brokers in a cluster [8].

1.2. Window-Aware Load Shedding

Window-Aware Load Shedding is a valuable technique for managing the processing of data streams, especially in scenarios where there is a high volume of data and limited processing resources. It helps to ensure that data processing remains efficient and effective without compromising the accuracy or quality of the results. The main objective of this technique is to prevent overloading and reduce the processing load on the system [9].

1.2.1. The Window Drop Operator

The Window Drop operator, also known as WinDrop, is a technique used in load shedding for aggregating queries that use windows of time while processing data streams. It takes six parameters, including S, T, N, ω, δ, p, and B, along with the stream

[1] *Ph.D. Research Scholar, Amity Institute of Information Technology, Gurugram, Haryana, India, (r.shubhamvyas@gmail.com)*
[b]*Ph.D., Professor - Department of Computer Science and Engineering, Amity School of Engineering and Technology, Gurgaon, Haryana, India, (rktyagi@ggn.amity.edu)*
[3]*Ph.D., Professor - Department of Computer Science & Engineering, Ajay Kumar Garg Engineering College, Ghaziabad, India, s (sahushashank75@gmail.com)*
[4]*Ph.D., Professor - Department of Computer Science and Engineering, Amity School of Engineering and Technology, Gurgaon, Haryana, India, (rktyagi@ggn.amity.edu)*
** Corresponding Author Email: r.shubhamvyas@gmail.com*

S as its input. The variables T, N, δ, and ω are obtained from the attributes of downstream aggregate operators, and the load shedder determines "p" based on the amount of weight to be discarded. "B" is calculated depending on the application's needs [10].

WinDrop probabilistically keeps or drops B successive windows using a Bernoulli drop probability "p." It splits the input stream "S" into time windows of size, starting a new window every time unit. Each tuple contains a system-assigned window specification property with a default value of "-1" to enable a downstream aggregation to open a window. This window's specified value may initiate in that tuple, and by this "T" input, consecutive tuples would be kept in the stream to maintain the opened window's integrity. "t" is set as its specification property by 0 to prevent a downstream aggregation from opening a window [11].

According to Tatbul et al. (2004) [1], WinDrop encodes window keep/drop choices into stream tuples for downstream aggregation operators. Table 2 provides the semantics for the window-specified variables with an A. O. Agg (F, ω, δ) 2 and assumes that it can be used to put a window drop before Aggregate. By using the initial two variables of WinDrop as a crow file from Aggregate, WinDrop can classify the input stream into windows in a similar pattern as Aggregate.

Table 1 provides a summary of the semantics for the window-specified variables using the notation A.O. Agg(F,ω,δ)2, and it suggests that it can be used to place a window drop before the Aggregate operation. To eliminate the "p" portion based on the output of the Aggregate operation, WinDrop (ω,δ,p, B) is inserted at the input of the Aggregate operation. Note that the first two variables of WinDrop serve as a crow file from Aggregate, allowing WinDrop to partition the input stream into windows in a similar manner to Aggregate [1].

Table 1. Semantics for the window specification attribute

Window instructions	**Details**
-1	Uncommitted
0	Discontinued by window
T	Continued by a window; must take care of tuples with $\tau > T$

2. Literature of Review

In recent years, the issue of load shedding in distributed stream processing systems has become a topic of great interest to researchers. In this literature review, we discuss several proposed solutions to address this problem.

Bang et al. [12] proposed a load-shedding technique to prevent famine in Apache Kafka. The authors gathered metrics from the JMX interface and designed an engine to solve the issue of starvation. They also presented a method to prevent all message producers from sending data when necessary. Results from experiments showed that the load-shedding engine can effectively address the problem of starvation by reducing the number of messages in the broker data queue.

Basaran et al. [13] suggested a simple load-shedding approach to resolve the per-stream backlog issue. This technique considers the per-stream backlog and query selectivity to enable smooth and fine-grained load shedding. The authors also conducted experiments using high-rate internet traces and demonstrated that their approach can accurately support the length of the queue for each stream.

Wang et al. [14] presented a prediction model that combines the Fuzzy C-Means (FCM) clustering algorithm and the Fuzzy Network (FN) technique. The suggested model's Mamdani rule-based structure is determined using a hybrid of the FCM algorithm (which uses data) and the expert-system approach (which uses expert knowledge) brought together using the FN technique. The parameters of the fuzzy set are optimized using a particle swarm optimization (PSO) technique. The suggested model was validated on 6 independent datasets, and the findings were compared to those obtained using the FCM technique. Based on these findings, the model is the most accurate, transparent, and interpretable option.

Liu et al. [15] demonstrated cutting-edge methods of data categorization. The goal of this clustering is to generate a structure in the data that is in line with how people naturally think about classifying things. In comparison to traditional clustering techniques, the proposed method elucidates the underlying structure and reasoning behind cluster formation in greater detail. The results of Axiomatic Fuzzy Set (AFS) theories can be better understood when this is considered in the representation.

Xie et al. [16] observed that beginning from a large-scale priority approach, fuzzy iteration with granular balls is used to improve iteration efficiency, with data membership degrees only considering the two granular balls in which they are placed. More processing techniques are available for the produced fuzzy granular-balls set, increasing the applicability of fuzzy clustering computations in a wide variety of data circumstances.

Mi et al. [17] developed the fuzzy-based concept learning model (FCLM) to address these two problems by making use of the hierarchical structure of concepts represented in concept lattices. The concept similarity measure in concept lattices is derived using object-oriented and attribute-oriented fuzzy concept similarities, and this paper first demonstrates some new related concepts for FCLM in a standard fuzzy formal decision setting. Additionally, a unique fuzzy idea learning framework is constructed, along with learning algorithms to go along with it. Lastly, undertake tests on a variety of real-world datasets to prove that the proposed method can outperform other similarity-based learning techniques in terms of classification accuracy. Further, one can test the approach on the MNIST dataset to ensure its efficacy in idea identification.

Hayat et al. [18] present modern methods and key characteristics of cloud computing based on fuzzy logic. An introduction to cloud computing and a classification of existing studies come first. Secondly, they summarize relevant research papers and provide some of the most important methods described in the current literature. The discussion concludes with some suggestions for where the field may go from here.

Maratha et al. [19] estimate that prolongs the first node death as much as feasible and delays the frequent re-clustering procedure to save energy consumption by solving a linear optimization problem to maximize the lifespan of devices such as CHs. The study also employs the CH uniform distribution to guarantee consistent power usage across all IoT gadgets. For IoT, the proposed clustering method ECFEL (Efficient Clustering using Fuzzy logic based on Estimated Lifetime) outperforms the existing protocols Low Energy Adaptive Clustering Hierarchy (LEACH), Novel-PSO-LEACH, FM-SCHEL, Modified LEACH (MOD-LEACH), Dynamic k-LEACH (DkLEACH), and M-IWOCA. The simulation's findings reveal that ECFEL has a longer first-node death (FND), half-node death (HND), and last-node death (LND) than other similar devices. The trials also show that ECFEL uses less power while keeping a constant packet delivery ratio over a longer period.

Mozafari et al. [20] proposed a unified engine that integrates analytics, transaction processing, and stream processing in a single cluster. The authors combined Apache Spark with Apache GemFire to construct this approach, which they named Snappy Data.

Rivetti et al. [21] presented a load-aware shedding technique (LAS) for distributed stream processing systems. LAS works with operator load that is affected by both input rate and tuple data. The authors designed a load-shedding mechanism based on the operator load to keep average queuing latencies near a specific threshold. The authors demonstrated through experiments that LAS is a close approximation to the optimal method and can deliver performance that closely approaches a specified objective while dropping some individual tuples if the specific load caused by each tuple is considered.

The issue of load shedding in distributed stream processing systems has prompted several suggested solutions from the scientific community. Some of these methods are straightforward, considering the per-stream backlog and query selectivity, while others are more complicated, like combining different shedding algorithms into a single model. The proposed techniques have shown promising results in experiments and can significantly improve the performance of distributed stream processing systems.

3. Background Study

The field of complex event processing (CEP) involves computer systems that evaluate queries by applying them to streams of events. These systems often have high input rates and query selectivity, and load shedding is commonly used in traditional data stream processing during short periods of high demand. However, this approach may not always be suitable for CEP queries, as certain stream elements may hold significant value for the query answer. To address this issue, a hybrid model of input-based shedding and state-based shedding has been proposed, which can deliver high-quality results while working with minimal resources. Experimental results indicate that using hybrid shedding instead of baseline approaches can improve recall by up to 14 times for synthetic data and up to 11 times for real-world data [22].

4. Problem statement

The problem addressed in this research is the efficient processing and analysis of real-time data streams using complex event processing (CEP) techniques. Specifically, the research focuses on the topic of semantic load shedding, which involves determining when and how much load to shed to manage the flow of data from source to receiver. The challenge of prioritizing events for load shedding is complex and requires special attention to be effectively solved. To address this problem, a new method called fuzzy-based clustering of events has been developed. This method is designed to prioritize the scheduling of events based on clustering using the KNN technique. The proposed solution involves several phases, including data buffering, data processing, data clustering via KNN, evaluating window drop size and slide, fuzzy-based priority scheduling of events, and feature selection/optimization using PSO.

5. Research Objectives

- To design and develop an optimized framework of semantic load shedding using fuzzy-based priority scheduling and evaluate the impact of buffer size and KNN clustering on system performance.
- To implement the proposed enhancements to producer and consumer APIs in Apache Kafka and assess the impact on throughput, latency, recall, and ratio of shed events.
- To compare the performance of the proposed framework with related work and evaluate its effectiveness in improving system performance.
- To investigate the effectiveness of the Particle Swarm Optimization algorithm in feature selection and feature set optimization for real-time data streaming in the proposed framework and determine the optimal feature set for maximizing system performance.

6. Technique used

In this section, the technique which has been referred to is described as follows:

6.1. Fuzzy Clustering

According to research [11], it has been discovered that the Fuzzy C Means method is superior to other algorithms like Ward's clustering and the K-means algorithm when it comes to machine learning and image processing. The Fuzzy C Means method has certain potential flaws due to its heavy dependence on prototypes and the optimization process itself. This algorithm makes use of a minimization function to optimize its performance:

$$j_m = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^{m} \left\| x_i - c_j \right\|^2 , \ 1 \leq m \leq \infty \qquad (1)$$

Jm is evaluated difference using other parameters, where xi represents the ith point in the database, cj represents the jth center allocated for the cluster, and $||*||$ means the distance between the dataset or point to the center. The idea of randomness in fuzzy logic is where the fuzziness index known as "{m|m {m|m R>1}}" comes from. This index indicates the degree of ambiguity associated with an event. The iterative procedures below are required to get a result for the goal function located above [23].

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} 〚 \left(\frac{\left\| x_i - c_j \right\|^{\frac{2}{m-1}}}{\left\| x_i - c_k \right\|} 〛 \right)} \qquad (2)$$

$$C_j = \frac{\sum_{i=1}^{n} u_{ij}^{m} . x_j}{\sum_{i=1}^{n} u_{ij}^{m}} \qquad (3)$$

In Equations 2 and 3, the degree of membership is "***u*ᵢⱼ**" of individual ***x*ᵢ** belonging to cluster ***j*; *C*ⱼ** is the center. Both are taken as though. case by case [12].

The computation follows these steps:

The process of Fuzzy c-means involves the following steps:

- Initialization of cluster center: In this step, the distance of each data point from the centers is calculated using Euclidean distances.
- Membership degree evaluation: Each data point is evaluated for its membership degree, with the total membership degree being equal to 1.
- New cluster centers calculation: The new cluster centers are calculated using the estimated membership degree in equation (4).
- Repeat the above steps until convergence: The above steps are repeated until the gap between the previous and current generations of "jm," "cj," or "uij" is less than a predetermined threshold value (ε). The value of ε can be an intuitively tiny amount or a previously established value, depending on the situation.

One limitation of Fuzzy c-means is its reliance on prototypes and the optimization process itself, even though it has been shown to outperform other algorithms in machine learning and image processing, such as Ward's clustering and the k-mean algorithm [13].

6.2. K Nearest Neighbor

The author proposed a fast K-nearest neighbor (KNN) method called FKNN, with the aim of addressing the drawbacks of large calculations. However, it was found that FKNN failed to improve accuracy. In this study, clustering was utilized to reduce the amount of computing required, and the centers of clusters were selected as the representative locations. To overcome the fault of no difference between distinctive terms and reduce the high calculation complexity [24], the authors used KNN, which is a supervised learning predictable classification non-parameter technique in pattern recognition. KNN is used to classify training samples without the need for further data.

6.3. Proposed Methodology

The complete strategy of the proposed architecture, as shown in Figure 1, is elevated in this section. The initial step of the architecture is the streaming of data, which is explained as follows:

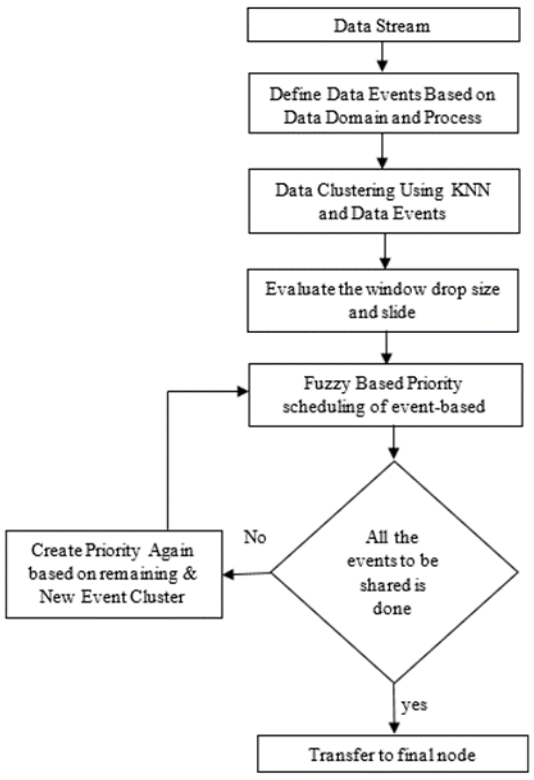
Data Stream
Define Data Events Based on Data Domain and Process
Data Clustering Using KNN and Data Events
Evaluate the window drop size and slide
Fuzzy Based Priority scheduling of event-based
All the events to be shared is done
No
Create Priority Again based on remaining & New Event Cluster
yes
Transfer to final node

Fig 1. Proposed Architecture

6.4. Proposed Algorithm

Algorithm: Prioritize **E**vent **S**cheduling via **KNN Clustering.**
Start

```
Phase I – Collection of Data Stream from Steps 1 – 3.
Step 1:  Set buffer_size to an integer value
Step 2:  data_buffer ← []
Step 3:  Start a loop to collect data from the data stream
    new_data ← Read and Extract major data from the data stream
    data_buffer ← new_data

    if length (data_buffer) = buffer_size do
        Process "data_buffer" as a batch
        Clear "data_buffer"
Phase II – Data-Events on Data-Domain & process from Steps 4 – 11
Step 4:  Input ← Data Event and associated Data Domain and process
Step 5:  Identify the Data Domain and process associated with the Data Event.
Step 6:  Extract the relevant Data from the event.
Step 7:  Update the relevant Data in the system based on the process.
Step 8:  Trigger any relevant actions or decisions based on the Updated Data.
Step 9:  Store the Updated Data and any relevant information about the event in a Log or DataBase.
Step 10: Return the Updated Data.
Step 11: Output ← Updated Data and Relevant Actions or Decisions
Phase III – Data Cluster via KNN & Data Events from Steps 12 – 17.
Step 12:
        o Read the data into a matrix, X
        o Choose a value for k, the number of nearest neighbors to consider
Step 13:
        o Normalize the data if necessary.
        o Compute the distance between each pair of data points.
Step 14:
        o For each data point, find its k nearest neighbors using the computed distances.
        o Assign each data point to a cluster based on the majority class of its k nearest neighbors.
Step 15:
        o Repeat Steps 13 – 14 until convergence, i.e., no data points change cluster assignment.
Step 16:
        o Return the final cluster assignments for each data point.
Step 17:
        o Visualize the clusters to check the quality of the clustering results.

Phase IV – Evaluate the Window Drop Size and Slide.
Step 18: Define Evaluate_Window_Drop_Size_and_Slide with Data & Desired_Performance_Metric.

best_window_drop_size ← 0
best_slide ← 0 ; best_performance ← -inf

For j in possible_window_drop_sizes, do
    For i in possible_slides, do
        result ← run_sliding_window_algorithm(data, j, i)
        performance ← evaluate_performance(result, desired_performance_metric)

        if performance > best_performance do
            best_window_drop_size ← j
            best_slide ← slide
            best_performance ← performance
Phase V – Fuzzy-based Priority Scheduling of Event-based from Steps 19 – 17.
Step 19: Input events and their fuzzy priorities
Step 20: Define a fuzzy inference system to calculate the crisp priority of each event.
Step 21: Sort the Events Based on their Crisp Priorities
Step 22: Schedule the Events in the order of their Crisp Priorities
Step 23: Repeat the process for new incoming events and update the schedule accordingly
Step 24: Cluster event scheduling priority Determines event Prioritisation after Window Drop Size and Slide analysis.
Step 25: Then is shared with the Transfer Node by checking the if-else condition for completion.
Step 26: The Highest Priority event from End-User input will be Shared if any events remain.
Step 27: Repeat until all event clusters reach the transfer node.

Phase VI – Feature Selection / Feature Set Optimisation using PSO from Steps 20 – 21.
Step 28: LS manages data flow and preserves system performance when overloaded, solving the starvation problem.
Step 29: Kafka distributes massive volumes of real-time log data.
Step 30: Apache Software Foundation created it.
Step 31: A fuzzy-based event grouping system has been made for semantic load shedding in real-time data streaming.
End
```

7. Result and Discussion

7.1. Constant used to calculate results.

There are some constants or parameters used to calculate results at the time of implementation, which are shown in Table 2 as given below:

Table 2. Apache Kafka Streams

Parameter	Value	Default
Instances Count	6	/
No. of threads	5	/
Java heap /Memory	20GB/ 22GB	/
Producer Compression	lz4 (L: none)	none
Producer Batch Size	300 KB (L: 16KB)	20 KB

Results

According to Figure 2 (a), the hybrid load-shedding approach achieves the highest recall. This strategy achieves a 100 percent recall rate for latency constraints ranging from 900µs to 500µs, which is consistent with previous research. In contrast, the baseline solutions lose their effectiveness quickly as the latency boundaries become more restrictive. Accordingly, it seems that the suggested method can accurately evaluate input event significance and deal with incomplete matches.

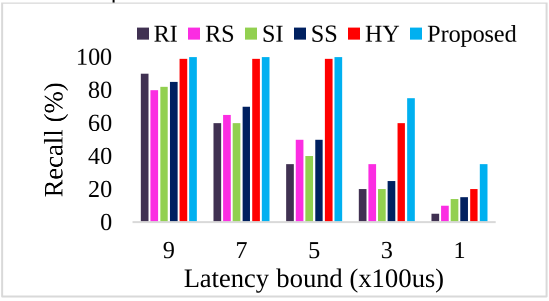

Fig 2 (a). Recall of latency bound.

The results indicate that state-based techniques, as shown in Figure 2(a) (fine granular shedding), provide better recall, whereas input-based methods, as illustrated in Figure 2(b), generate higher throughput (instant resource savings). It is worth noting that, given the memory results shown earlier, the proposed approach is almost as effective as the input-based solutions. The impact of shedding on the accuracy and speed of Fuzzy EC under a strict latency constraint is also assessed.

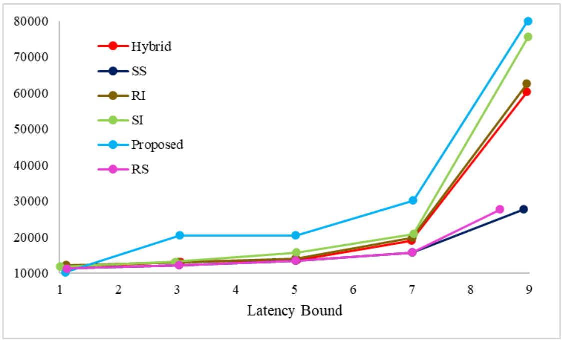

Fig 2 (b). Throughput of latency bound.

The efficiency of the suggested method is shown by the ratio of shed events to partial matches, as shown in Figure 2 (c) and Figure 2(d). The described approach discards a fixed fraction of input events up to 500µs. The increase in shed partial match, which does not affect recall, is used to achieve the required decrease in latency, as seen in Figure 2(a). A more consistent proportion of rejected partial matches occurs when more input events must be discarded to fulfill the latency limitation. Reduced stress during the shedding process is achieved using input-shedding to avoid the development of incomplete matches.

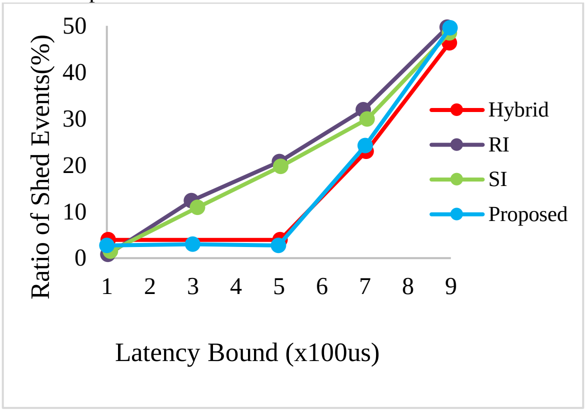

Fig 2 (c). The ratio of Shed event of latency bound.

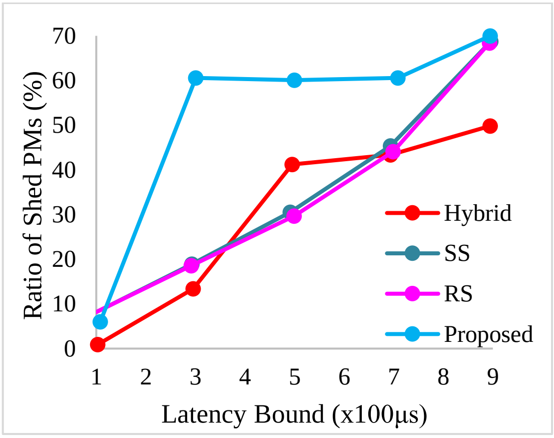

Fig 2 (d). The ratio of Shed PMs of latency bound.

It is essential to measure the efficiency of input events or incomplete matches that do not influence recollection. Figure 3 (a) and Figure 2 (b) show the success of selecting and evaluating their quality. Input-based shedding using the cost model Hyl maintains a higher recall compared to random input (RI) and selectivity-based input (SI) shedding at the cost of a slightly lower throughput. This indicates that the cost model can provide a reliable estimate of the value of matches and events to shed.

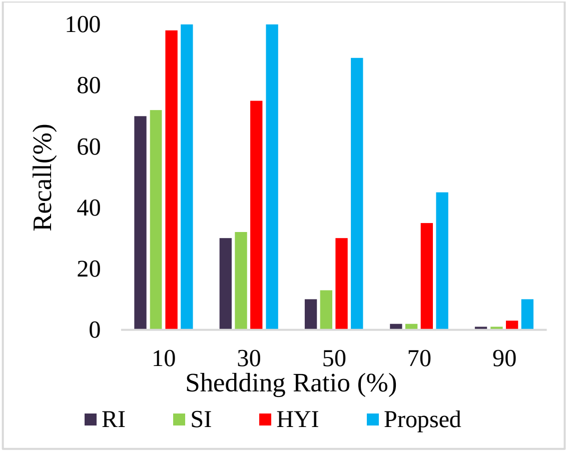

Fig 3 (a). Recall (Input-based)

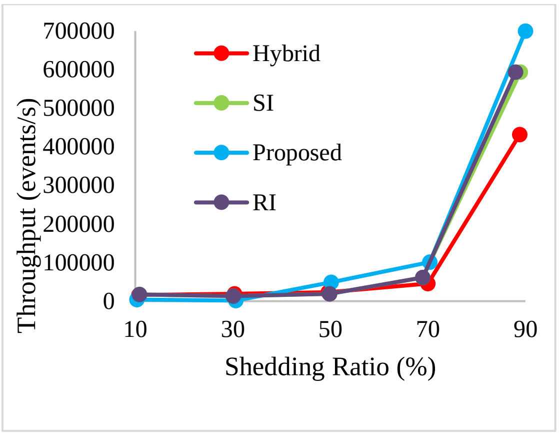

Fig 3 (b). Throughput (input-based)

Figure 3 (c) depicts the recall rates of state-based techniques, and it is evident that the proposed HyS method outperforms both random strategies (RS) and selectivity-based strategies (SS) in terms of recall. Despite discarding 50 percent of the partial matches, the proposed method maintains a perfect 100 percent recall rate, whereas the baseline solutions achieve only a 30 percent recall rate.

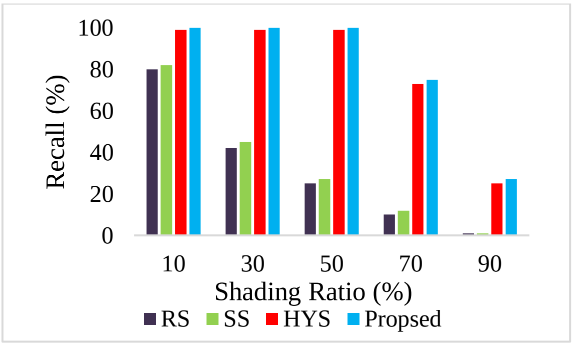

Fig 3(c). Recall (State-based)

Figure 3 (d) indicates that all methods have similar throughput at that moment. However, the baselines can achieve better throughput only when they have high shedding ratios, which result in extremely low recall. Therefore, in practical terms, this approach has limited utility.

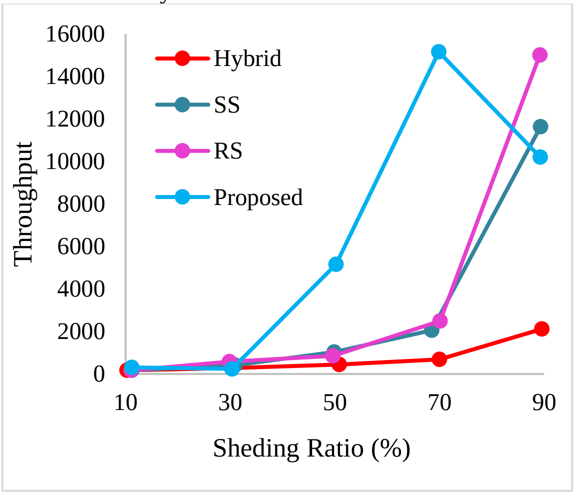

Fig 3 (d). Throughput (state-based)

Figure 4(a) demonstrates the impact of query selectivity variation, indicating that the suggested method achieves the best results even when the 99th percentile latency has a 50% constraint, and the recall is not affected. This finding aligns with the initial expectations.

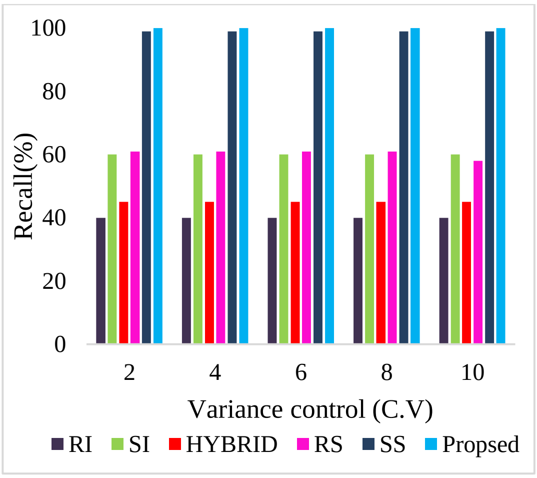

Fig 4 (a). Recall of variance of Query selectivity.

In Figure 4 (b), the impact on throughput is evident. The hybrid approach can effectively evaluate the significance of input events and discard those that are deemed unimportant when selectivity exhibits low variances. However, when dealing with high variance, the method employs a very fine level of partial matches to achieve comparable throughput to the baseline technique.

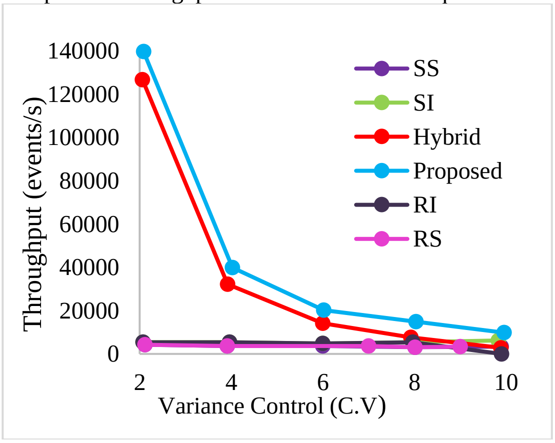

Fig 4 (b). Throughput of variance of query selectivity

The number of partial matches growing at a fixed input rate as a function of time window size is shown in Figure 5. The number of partial matches generated is based on the period of the query. The 99th percentile delay is limited to 50% of the total. As shown in Figure 5 (a), the proposed technique consistently achieves the highest recall, and recall improves for all methods as the window size increases. This improvement in recall could be attributed to an enhanced cost estimate.

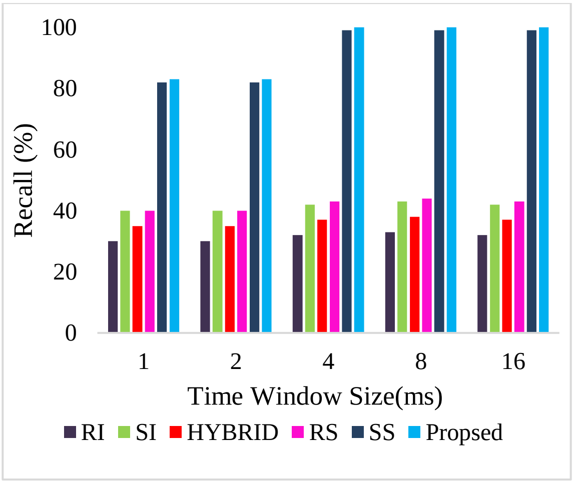

Fig 5 (a). Recall of window size.

As the window size increases, both partial and full matches are considered to provide an estimation. Figure 5(b) shows that input-based baseline approaches achieve the maximum throughput. The hybrid approach is just as effective as state-based approaches. Differences become less noticeable as the window size grows due to the exponential rise in the number of partial matches and their increased lifetime.

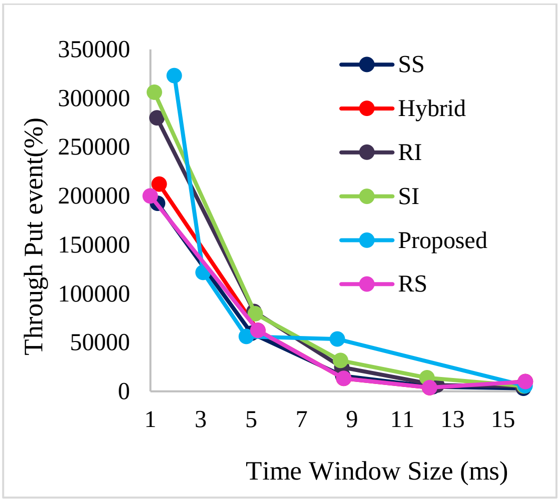

Fig 5 (b). Throughput of window size.

Figure 6 illustrates the effects of Query Pattern Length. The figure shows that as the pattern length increases, the recall remains unchanged, but the throughput decreases significantly. Interestingly, the proposed method shows a smaller reduction in throughput compared to the other methods. As a result, the proposed method can perform better with more complex queries.

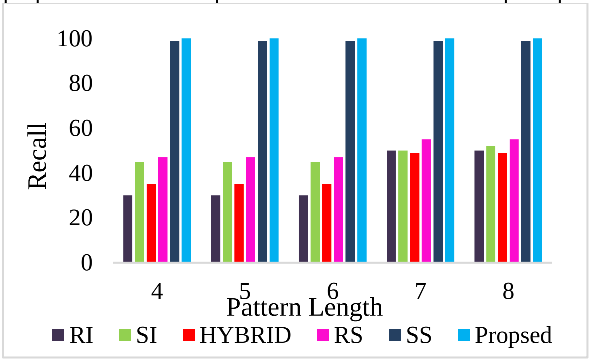

Fig 6. Recall of pattern length.

Figure 7 demonstrates the impact of improved temporal resolution. As seen in the figure, an increase in the number of time slices results in a decrease in throughput and an increase in overhead. However, the hybrid approach provides better recall than RI and SI (one slice) while maintaining similar throughput. Similar results are obtained with state-level baseline strategies. The best recall is obtained with four or more slices.

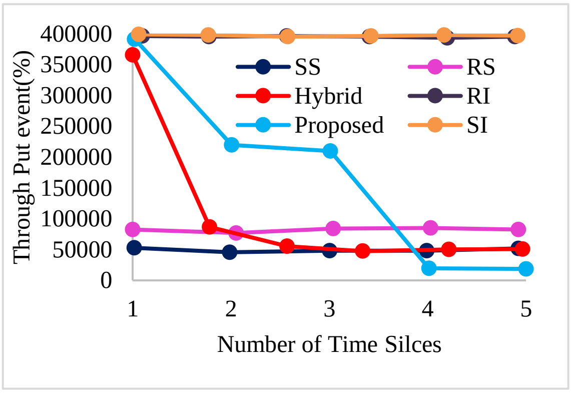

Fig 7. Throughput of temporal granularity.

Discussion

There are several ways in which the suggested model is preferable than existing ones. Figure 6 shows that the throughput of a system reduces dramatically as the length of a query pattern grows, but the recall stays about the same. However, our model stands out by exhibiting a lesser reduction in throughput than previous approaches. Because of this special quality, our model can answer even the most difficult problems with remarkable speed and accuracy. Figure 7 shows that when the number of time slices is increased, the throughput decreases and the overhead rises, indicating the influence of enhanced temporal resolution. In contrast, while retaining the same throughput as the baseline, our hybrid technique improves by providing greater recall than approaches employing only one time slice. This pattern is true even at the state level, where baseline techniques with four or more slices have the highest recall. while it comes to data-intensive applications, the suggested approach stands out because of the excellent balance it achieves between preserving throughput under complicated query patterns and offering great recall while managing higher temporal resolution.

8. Conclusion and future work

In conventional data stream processing, this task is accomplished using a technique referred to as load shedding. In conclusion, the proposed method is a promising approach for improving the efficiency of Apache Kafka. The hybrid load-shedding technique has demonstrated high recall rates while maintaining comparable throughput and cost models for estimating the value of matches and events to shed. Additionally, shedding a constant proportion of input events can be an effective strategy for achieving the desired decrease in latency without sacrificing recall rates. Furthermore, state-based tactics have shown superior recall rates, while input-based procedures generate higher throughput. In terms of the number of time slices used, the hybrid method with four or more time slices achieves the highest recall rates while maintaining reasonable throughput. These findings can guide the development of more efficient and effective machine-learning algorithms and load-shedding techniques for various applications. Future works will focus on making the method dynamic by incorporating an automated procedure to compute the best sample rate based on the system's environment, including workload, data flow, and available resources. Overall, it contributes to advancing the state of the art in machine learning algorithms and load-shedding techniques.

DISCLAIMER: This paper is for research purposes only. The views expressed in this paper are personal views of the author (s) and not those of Optum/UHG. Optum/UHG or any of their affiliates shall not be responsible for the statements made or views presented by the author (s) in this paper.

Acknowledgment

The author would like to express her sincere gratitude to supervisors from the "Electronics and Communications Department " for their invaluable contributions to this research paper. Their expertise and insights have been instrumental in shaping the direction of this study. The author is also grateful for the support and resources provided by the " Department of Computer Science and Engineering, Amity School of Engineering and Technology, Gurgaon, Haryana, India ". The author extends our heartfelt thanks to all those who have contributed to the successful completion of this research.

Conflict of Interest

The authors declare that they have no conflict of interest

References

[1] N. Tatbul and S. Zdonik, “Window-aware load shedding for aggregation queries over data streams” in VLDB, vol. 6, 2006, pp. 799-810.

[2] R. Guo et al., “Bioinformatics applications on Apache Spark,” GigaScience, vol. 7, no. 8, p. giy098, 2018. (doi:10.1093/gigascience/giy098).

[3] R. Shree et al., “KAFKA: The modern platform for data management and analysis in the big data domain” in 2nd international conference on telecommunication and networks (TEL-NET). IEEE, 2017, pp. 1-5. (doi:10.1109/TEL-NET.2017.8343593).

[4] A. Floratou et al., “Dhalion: Self-regulating stream processing in heron,”, Proc. VLDB Endow., vol. 10, no. 12, pp. 1825-1836, 2017. (doi:10.14778/3137765.3137786).

[5] G. Van Dongen and D. Van den Poel, “Evaluation of stream processing frameworks,” IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 8, pp. 1845-1858, 2020. (doi:10.1109/TPDS.2020.2978480).

[6] P. Le Noac’H et al., “A performance evaluation of Apache Kafka in support of big data streaming applications” in IEEE International Conference on Big Data (Big Data). IEEE, 2017, pp. 4803-4806. (doi:10.1109/BigData.2017.8258548).

[7] B. R. Hiraman et al., “A study of Apache Kafka in big data stream processing” in International Conference on Information, Communication, Engineering and Technology (ICICET). IEEE, 2018, pp. 1-3. (doi:10.1109/ICICET.2018.8533771).

[8] K. M. Thein, “Me. ‘Apache Kafka: next generation distributed messaging system.’,” Int. J. Sci. Eng. Technol. Research, vol. 3, no. 47, pp. 9478-9483, 2014.

[9] Y. Chen et al., “Fast density peak clustering for large scale data based on kNN,” Knowl. Based Syst., vol. 187, p. 104824, 2020. (doi:10.1016/j.knosys.2019.06.032).

[10] B. Mozafari and C. Zaniolo, “Optimal load shedding with aggregates and mining queries” in 26th International Conference on Data Engineering (ICDE 2010). IEEE. IEEE, 2010, pp. 76-88. (doi:10.1109/ICDE.2010.5447867).

[11] B. Zhao et al., “Eires: Efficient integration of remote data in event stream processing” in Proc. 2021 International Conference on Management of Data, 2021, pp. 2128-2141. (doi:10.1145/3448016.3457304).

[12] J. Bang et al., “Design and implementation of a load shedding engine for solving starvation problems in Apache Kafka” in Noms IEEE/IFIP Network Operations and Management Symposium, vol. 2018. IEEE, 2018, pp. 1-4. (doi:10.1109/NOMS.2018.8406306).

[13] C. Basaran et al., “Adaptive load shedding via fuzzy control in data stream management systems” in Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA). IEEE, 2012, pp. 1-8. (doi:10.1109/SOCA.2012.6449438).

[14] X. Wang et al., “Fuzzy-clustering and fuzzy network based interpretable fuzzy model for prediction,” Sci. Rep., vol. 12, no. 1, p. 16279, 2022. (doi:10.1038/s41598-022-20015-y).

[15] X. Liu et al., “Fuzzy clustering with semantic interpretation,” Appl. Soft Comput., vol. 26, pp. 21-30, 2015. (doi:10.1016/j.asoc.2014.09.037).

[16] J. Xie et al., “Research on efficient fuzzy clustering method based on local fuzzy granular balls,” Arxiv e-Prints, 2023: arXiv-2303.

[17] Y. Mi et al., “Fuzzy-based concept learning method: Exploiting data with fuzzy conceptual clustering,” IEEE Trans. Cybern., vol. 52, no. 1, pp. 582-593, 2022. (doi:10.1109/TCYB.2020.2980794).

[18] B. Hayat et al., “A study on fuzzy logic-based cloud computing,” Clust. Comput., vol. 21, no. 1, pp. 589-603, 2018. (doi:10.1007/s10586-017-0953-x).

[19] P. Maratha and K. Gupta, “Linear optimization and fuzzy-based clustering for WSNs assisted internet of things,” Multimedia Tool. Appl., vol. 82, no. 4, pp. 5161-5185, 2023. (doi:10.1007/s11042-021-11850-8).

[20] B. Mozafari et al., “SnappyData: A unified cluster for streaming, transactions and interactive analytics” in CIDR, vol. 17, 2017, pp. 8-11.

[21] N. Rivetti et al., “Load-aware shedding in stream processing systems” in Proc. 10th ACM International Conference on Distributed and Event-Based Systems, 2016, pp. 61-68. (doi:10.1145/2933267.2933311).

[22] K. Tang et al., “DRS+: Load Shedding Meets Resource Auto-Scaling in Distributed Stream Processing” 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE. IEEE, 2020, pp. 292-301. (doi:10.1109/HPCC-SmartCity-DSS50907.2020.00036).

[23] H.-Y. Wang et al., “A survey of fuzzy clustering validity evaluation methods,” Inf. Sci., vol. 618, 270-297, 2022. (doi:10.1016/j.ins.2022.11.010).

[24] S. K. Jha et al., “A hybrid machine learning approach of fuzzy-rough-k-nearest neighbor, latent semantic analysis, and ranker search for efficient disease diagnosis,” J. Intell. Fuzzy Syst., vol. 42, no. 3, pp. 2549-2563, 2022. (doi:10.3233/JIFS-211820).