Some New Observations on SLO-aware Edge Stream Processing

Amna Shahid[1], Peng Kang[2], Palden Lama[2], and Samee U. Khan[1]

[1]Mississippi State University
[1]as5120@msstate.edu and skhan@ece.msstate.edu
[2]The University of Texas at San Antonio
[2]{peng.kang, palden.lama}@utsa.edu

***Abstract*—The emergence of edge stream processing has created a new way of processing real-time data from the Internet of Things (IoT), which comprises a plethora of geographically dispersed physical devices equipped with sensors and actuators that exchange data with the Cloud. Nevertheless, edge stream processing systems face new challenges, including dynamic workloads, resource limitations, and multi-tenant application hosting. Adaptive resource management has been proposed to address these issues. However, this technique may lead to Service Level Objective (SLO) violations when the system encounters resource constraints. To mitigate this problem, we investigate the benefits of using priority-based stream data to reduce the SLO violations associated with adaptive resource management. Our findings demonstrate that segregating data according to their priority levels and processing them accordingly can significantly enhance the efficiency and stability of the system. We implemented this technique on the Storm Streaming system and used RIOT as a benchmark, employing VRebalance and other approaches to adjust system resources dynamically.**

***Index Terms*—Internet of Things, Stream Processing, Priority-based Scheduler.**

I. Introduction & Motivation

The advent of the Internet of Things (IoT) has brought about a pressing need for processing continuous data streams generated by devices, enabling integration and control operations within intelligent systems [1]–[3]. This data must often be processed under strict time constraints to extract meaningful insights for future actions, and in some cases, it must be processed in near real-time to identify patterns of interest. To address this need, *stream processing* has emerged as a leading paradigm for processing high-volume and continuous data streams in real-time as shown in Fig. 1. Unfortunately, popular stream processing engines such as Apache Storm [1] and Apache Spark [2] are primarily designed for resource-rich cloud environments and may not perform optimally at the edge. A recent study [4] showed that Storm's default resource scaling mechanism is slow and can lead to inefficient usage in resource-constrained environments.

As edge computing becomes more prevalent, it is crucial to develop stream processing solutions that can effectively operate in such environments while minimizing data transfer costs over the wide area network (WAN). However, edge stream processing poses a significant challenge in promptly responding to dynamically changing input loads, as any delay could result in catastrophic outcomes. For instance, IoT-based traffic control systems in smart cities require real-time analysis of live video streams captured from cameras deployed at various intersections [5]. Unfortunately, scenarios like a sudden surge in user traffic resulting from a natural disaster or the death of a prominent individual cannot be predicted. One potential solution is to apply a priority-based resource scheduling technique [6], [7] to a traditional stream processing system, allowing users to prioritize certain input data items or nodes in the processing graph. However, these techniques have some limitations when used in edge environments, as they do not adequately address dynamic resource allocation or competition between multiple applications.

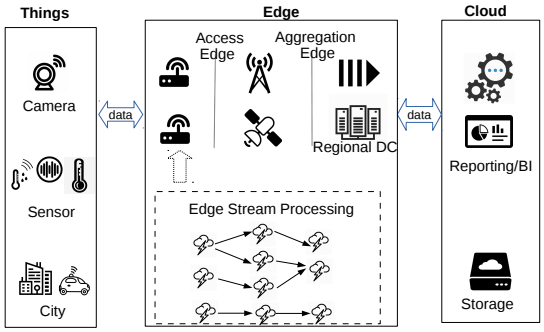


Fig. 1: IoT: Device to Cloud.

In our previous study [4], we presented VRebalance, a virtual resource orchestrator that ensures end-to-end performance for concurrent stream processing workloads at the edge. VRebalance employs Bayesian Optimization ($\mathcal{BO}$) to identify near-optimal resource configurations rapidly. However, this approach did not account for data priority and resulted in SLO violations when resource constraints were reached. To address this limitation, we explored using a data priority scheduler to sort input data with different priorities, ensuring strict latency for high-priority data while sacrificing low-priority data. We conducted experiments using the open-source IoT

[1]http://storm.apache.org/
[2]https://spark.apache.org/

benchmark RIoTBench [8] and the Apache Storm stream processing engine to implement our priority scheduler. Our study suggests that stream processing systems based on data-priority provide superior SLO guarantees for high-priority data compared to traditional stream processing systems.

In conjunction with existing resource allocation techniques, our priority scheduler consistently achieves 0% SLO violation rate for high-priority data in concurrently running applications even for highly dynamic workloads. In particular, priority scheduling with $\mathcal{BO}$-based resource allocation achieves the best overall performance with only up to 8.5% SLO violations even for low-priority data and up to 3% SLO violations for medium-priority data. Furthermore, we analyzed the impact of micro-batching with different batch sizes on the performance of priority-based stream processing.

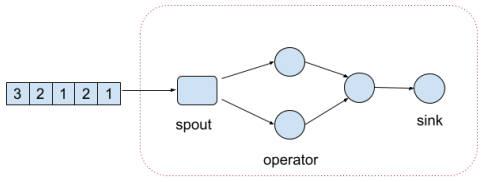
The paper is structured as follows. Section II-A provides an overview of the stream processing model. In Section II-B, we present the design and implementation details of our proposed technique. Section III describes the testbed setup and presents the results of our experimental evaluation. We review related work in Section IV. Lastly, Section V summarizes the findings, concludes our work, and introduces future work.

II. Stream processing system at the Edge

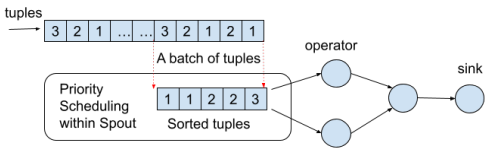
A. Stream Processing Engine

Edge Stream Processing Engines (ESPEs) are increasingly being deployed on IoT Gateways, which have more computing resources than wireless sensor networks but are still limited compared to the Cloud [9]. On these IoT Gateways, data streams produced by IoT devices are processed using data flow programming model, where each application is packaged as a directed acyclic graph (DAG) data structure, called a topology. Individual data points (tuples) flow through a topology from sources to sinks. Each inner node is an operation that performs arbitrary computation on the data, ranging from simple filtering to complex operations like Machine Learning (ML)-based classification algorithms. We assume that each application has an SLO target in terms of the 95th-percentile end-to-end latency of data tuples flowing through the topology.

Data stream processing systems can be broadly categorized into record-based (or continuous operator) streaming and micro-batching (or batched) streaming [10]. Record-based processing involves handling each record individually as soon as it arrives, allowing for real-time processing and lower latencies but sacrificing throughput. In contrast, micro-batching involves dividing the input stream into mini-batches and processing each batch one at a time. While this model results in higher throughput due to processing multiple records together, it typically has higher average latency. Apache Storm, Heron, and Flink support record-based streaming, while Spark Streaming and Storm Trident support micro-batching [11]. Combining these two methods may be a viable approach to optimize processing in an edge environment.



(a) Data stream processing.



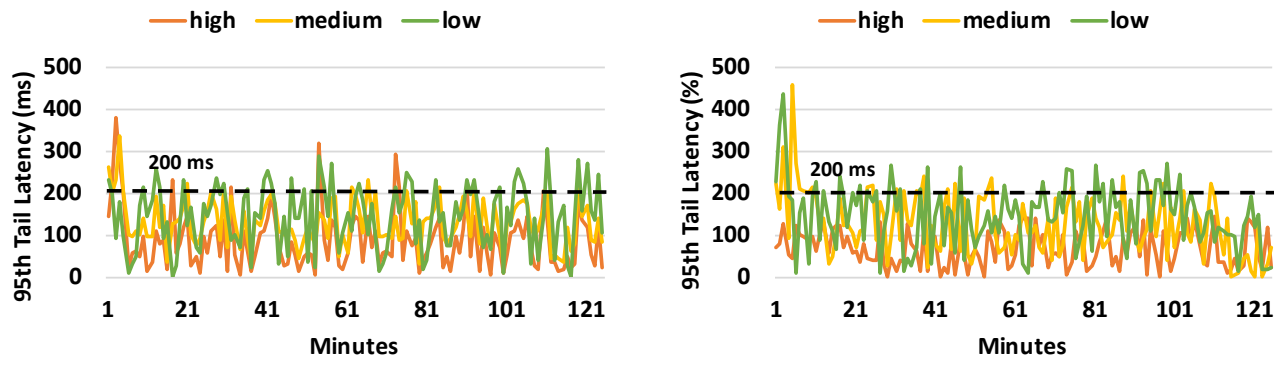
(b) Data stream processing with data-priority model. Priority number 1, 2, and 3 represent low, medium and high priorities respectively.

Fig. 2: Data stream processing model.

B. Priority-based Stream Processing

In this section, we describe our methodology to enable priority-based stream processing. Fig. 2a shows a traditional stream processing topology using Apache Storm, which consists of a DAG of spouts and bolts. A spout is a user-defined component that serves as data stream source. It is responsible to read data from an external source and emitting tuples (a basic unit of data) into the topology for processing. Whereas a bolt is a data processing unit. The system handles each record individually as soon as it arrives. However, this method does not take into account the varying degrees of urgency of different data items. In contrast, our priority-based stream processing system, as illustrated in Fig. 2b, is particularly useful in situations where some data items are more time-sensitive or critical than others, and require prompt processing, while others can tolerate longer processing times. Our priority scheduler is designed to support three priority levels 1, 2, and 3 which are referred to as low, medium, and high priority levels throughout the paper.

Our priority scheduler is implemented within the spout. It scans the input stream for priority levels as soon as the data items enter the system. For each micro-batch of data tuples collected, it sorts them based on priority levels and streams the data to the rest of the topology. High-priority data items are processed first and given precedence for computing resources, while normal and then low-priority items are processed later. This ensures that high-priority items are given priority and their latency remains relatively stable even in cases where the input load suddenly increases. In this paper, we use a batch size of 10 and 100. A sensitivity analysis of batch size is presented in Section III-D. Currently, the algorithm supports three priority levels: low, medium, and high. High-priority data items are processed first and given priority for



(a) Without priority scheduler.

(b) With priority scheduler.

Fig. 3: 95th percentile latency for ETL topology with TAXI dataset under static workload. SLO target is 200 ms.

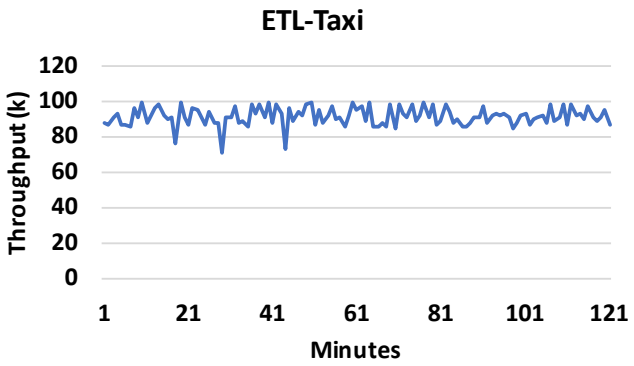


Fig. 4: Static workload. Workload is measured in tuples per minute.

computing resources, while medium and low-priority items are processed later. This ensures that high-priority items are prioritized and their latency remains relatively stable even when the input load suddenly increases. This process optimizes system performance and ensures that data from critical tasks are completed promptly. Furthermore, the algorithm can be extended to support a greater number of priority levels, resulting in more precise control over data item processing and resource allocation.

III. Evaluation

A. Experimental Testbed

1) Edge node configuration.: We setup a prototype testbed to represent an IoT Gateway by using a Ubuntu 16.04 machine equipped with 4 CPU cores and 8 GB RAM. We used Docker container engine (Version 18.06.2-ce) and Kubernetes (Version 1.18.2) container orchestration system to deploy an Apache Storm cluster composed of 12 containers. Two of the containers were used for Nimbus and Zookeeper, while the remaining containers were running as worker nodes. In our experiments, the default CPU limit of each container was set to 400 millicore (equal to 0.4 CPU core) and the requested CPU was 200 millicore.

2) Datasets.: We used two IoT datasets including Sense your City (CITY) [12] and New York city taxi trips (TAXI) [13]. The **CITY** dataset consists of real-time environmental data from crowd-sourced sensors deployed in seven cities across three continents, with approximately 12 sensors per city. The dataset reports six timestamped observations every minute, including temperature, humidity, ambient light, sound, dust, and air quality, along with sensor ID and geolocation metadata. The **TAXI** dataset comprises smart transportation data from 2 million taxi trips taken in 2013 in New York City, including pickup and drop-off dates, taxi and license details, start and end coordinates and timestamp, metered distance, taxes, and tolls paid.

3) Benchmarks.: We utilized the RIoTBench benchmark [8] suite to analyze our datasets, focusing on the ETL and PRED IoT applications. The suite includes four applications based on common IoT patterns for data pre-processing, statistical summarization, and predictive analytics. We did not use the STATS and TRAIN applications, which are integrated with public cloud services and unsuitable for low-latency Edge stream processing. To create a dynamic workload, we modified RIoTBench's input generator. It feeds multiple fixed-sized data batches (e.g., 10 tuples) into Storm's source task (Spout) every minute, with random delays following a Poisson distribution [14]. The workload intensity was adjusted by varying the total number of data batches per interval.

4) Evaluation method.: In this work, we use Apache Storm as a representative SPE, which is a distributed real-time computation system for processing unbounded streams of data. Storm topologies consist of spouts and bolts, where spouts serve as sources of data streams, and bolts are the data processing units. We evaluate four applications (ETL-TAXI, ETL-CITY, PRED-CITY, PRED-TAXI) from the RIoTBench benchmark suite. Each application uses a different combination of topology and dataset. The input data is allocated randomly to one of three different priority levels: low, medium, and high. This ensures an equal distribution of

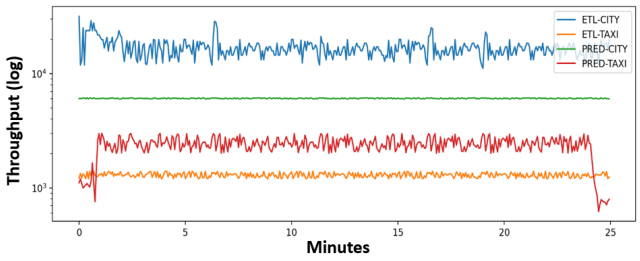


Fig. 5: Mix of static and dynamic workloads. Workload is measured in tuples per minute and shown in log scale.

input data across all priority levels. For each application, we set the SLO target of 200 ms for the 95^{th} percentile end-to-end latency of data tuples flowing through the topology.

B. 95^{th} Percentile Latency of an Application Running Alone Under Static Workload

First, we evaluate the impact of priority scheduling on the 95^{th} percentile end-to-end latency of a single application facing a static workload of 90K tuples per minute as shown in Fig. 4. For CPU resource allocation, we use the $\mathcal{BO}$ technique, presented in VRebalance [4], with the aim to meet the SLO target of 200 ms. The effectiveness of the priority scheduler in preventing SLO violations for high-priority data collected at two-hour intervals is demonstrated in Fig.3. Fig.3a reveals that without the priority scheduler, high-priority data suffered from SLO violations due to fluctuation in the 95th percentile latency. In contrast, Fig. 3b shows that the priority scheduler maintained the 95th percentile latency around 100 ms, which is well within the SLO threshold. These findings suggest that a priority-based data scheduler can effectively ensure stable performance for higher-priority data and prevent SLO violations.

C. Performance Comparison with Multiple Applications Running Concurrently under Dynamic Workload

We evaluate the impact of priority scheduling on SLO violations of multiple applications running concurrently under dynamic workload conditions. As shown in Fig. 5, ETL-CITY and PRED-TAXI face dynamic workloads, while ETL-TAXI and PRED-CITY operate under static workloads. The entire workload lasts for 24 hours and includes repeating patterns. We conduct a comparative analysis with existing dynamic resource allocation techniques, including VRebalance [4] $\mathcal{BO}$, and Hill Climbing with two different step sizes (50 millicore and 200 millicore), and a static resource allocation method with two different resource allocation levels (400 millicore and 4000 millicore, representing partial and full resource utilization, respectively).

In Fig. 6 and Fig. 7, we compare the SLO violation rates of different resource allocation methods without and with priority scheduling respectively. Fig. 6 illustrates that, in the absence of priority scheduling, all resource allocation methods lead to SLO violations across all categories of data and applications. In particular, ETL-CITY suffers up to 30% SLO violations even for high-priority data. In contrast, Fig. 7 shows that our priority scheduler is able to reduce the SLO violation rate of high-priority data to 0% for all applications and resource allocation methods. $\mathcal{BO}$ with priority scheduling achieves the best overall performance with only up to 8.5% SLO violations for low-priority data and up to 3% SLO violations for medium-priority data.

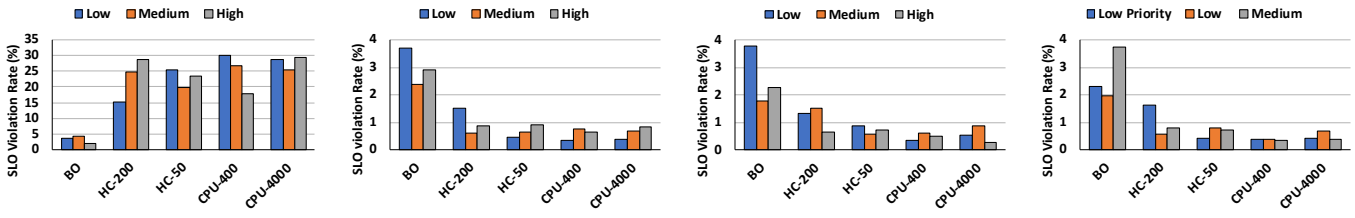
D. Analysis of batch size sensitivity

The choice of batch size is a critical factor that can significantly impact the performance and efficiency of a stream processing system. Fig. 8 compares the SLO violation rates for $\mathcal{BO}$ with priority scheduling using different batch sizes. We observe that a batch size of 10 achieves lower SLO violation rates than a batch size of 100, for all applications. A batch size of 10 reduces the SLO violation rate by up to 55% and 30% in the case of low-priority and medium-priority data respectively. For high-priority data, the SLO violation rate is 0% for both batch sizes.

In general, smaller batch sizes allow for more frequent updates to the system, resulting in faster processing times for individual data items. This can be particularly important for real-time processing, where timely updates are crucial. However, if the batch size is too small, priority scheduling may not be very effective. In the extreme case, a batch size of one will be equivalent to disabling priority scheduling. On the other hand, larger batch sizes can introduce more latency into the system as data items must wait to be processed until the entire batch is complete. In this paper, we use a batch size of 10 since it achieves a good balance between real-time responsiveness and service differentiation between data items based on their priority.

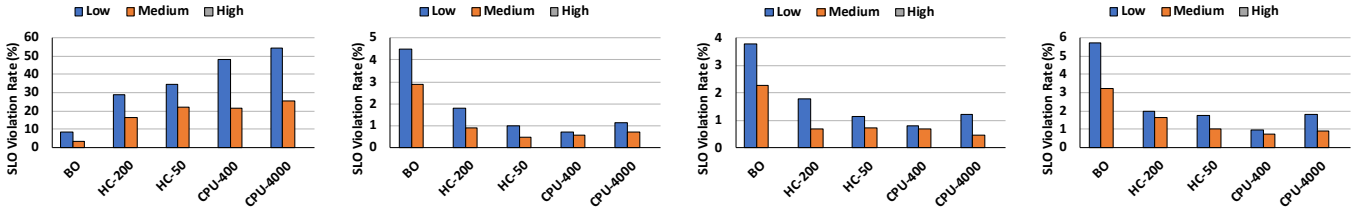
IV. Related Work

Resource management is a critical aspect of Edge computing. Xu et al. [15] proposed an auction-based mechanism for resource contract establishment and a latency-aware scheduling technique to optimize the utility for Edge computing infrastructures and service providers. Araldo et al. [16] implemented a polynomial-time resource allocation algorithm that enables Edge network operators to maximize their utility, which can improve inter-domain traffic savings, QoE for users, or other metrics of interest. Wang et al. [17] investigated workload reduction and resource allocation for cloudlet applications to manage application quality of service in the presence of contention for cloudlet resources. In the context of stream processing, several works have focused on auto-scaling distributed systems by monitoring the performance model of streaming dataflows [18], [19]. Kalavri et al. [20] proposed an automatic scaling controller based on a general performance model of streaming dataflows and lightweight instrumentation to estimate the processing and output rates of individual dataflow operators. These approaches differ from our work, which focuses on the study of priority-based stream processing in conjunction with various resource



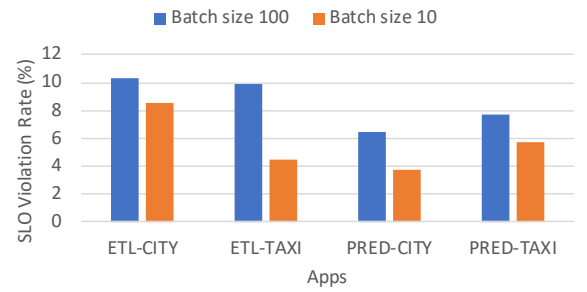
(a) ETL-CITY. (b) ETL-TAXI. (c) PRED-CITY. (d) PRED-TAXI.

Fig. 6: SLO violation without priority scheduler.

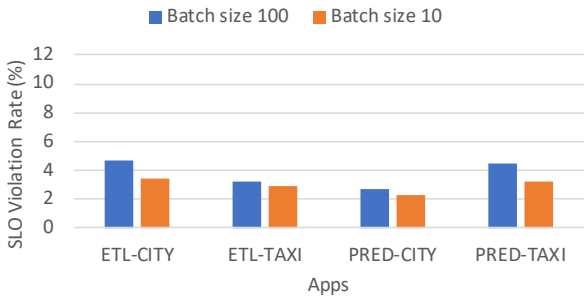


(a) ETL-CITY. (b) ETL-TAXI. (c) PRED-CITY. (d) PRED-TAXI.

Fig. 7: SLO violation with priority scheduler. SLO violation rate for high-priority data is 0%.



(a) SLO violation rate for low-priority data.



(b) SLO violation rate for medium-priority data.

Fig. 8: Comparison of SLO violations with different batch sizes. For high-priority data, SLO violation rate is 0% for both batch sizes 10 and 100.

allocation techniques in a resource-constrained Edge node. In particular, we highlight how data prioritization along with $\mathcal{BO}$-based resource allocation technique can achieve superior SLO guarantees in terms of the 95^{th} percentile latency.

Load shedding is an important technique used in edge computing to prevent system overload and ensure timely processing of high-priority tasks while maintaining a given latency bound. Some recent works [21], [22], have proposed load-shedding frameworks that utilize a probabilistic model to learn the significance of events in a window based on their position and type. They provide algorithms for determining when and how many events to drop and predicting a utility threshold to maintain the latency bound. While input-based load shedding is a common technique for CEP queries, the utility of individual events can vary significantly depending on the presence of partial matches, leading to potential quality issues. To address this, state-based techniques have been developed to selectively discard partial matches and maintain high result quality under limited resources [23]. Load shedding is complementary to our work. We plan to investigate the integration of load shedding and priority scheduling in our future work.

Approximation techniques have been used in hybrid edge-cloud streaming systems to minimize traffic over the WAN while achieving acceptable downstream query accuracy. One such technique involves sampling at the network edge, which has proven effective in reducing transmissions without introducing significant errors in downstream queries. Additionally, this approach enables error bounds to be set for different types of queries, which is essential in ensuring optimal performance in production deployments. Recently, Wolfrath [24] discovered that data streams across devices located in the same geographical area often exhibit correlations. This insight led to the development of a hybrid edge-cloud system that balances the tradeoff between sampling at the edge and estimation of missing values in the cloud, ultimately reducing WAN traffic. In contrast, our approach is focused on real-time stream processing at the Edge, with the primary goal of

meeting strict SLO guarantees. We prioritize processing data at the Edge as it is generated, rather than waiting for it to be transmitted to the cloud for processing.

V. Conclusions & Future work

In this paper, we investigate how data prioritization impacts the performance and stability of critical applications in resource-constrained stream processing at the Edge. Our findings reveal that organizing data based on priority levels and processing it accordingly can significantly improve the system's efficiency and stability. In particular, our priority scheduler achieved 0% SLO violation rate for high-priority data in concurrently running RIoTBench applications in terms of their 95^{th} percentile latency.

In the future, we will explore the integration of priority scheduling and load shedding to address congestion and tackle more complex scenarios. Load shedding can be used to selectively drop incoming data during periods of high congestion, thus reducing the load on the system and preventing it from becoming overwhelmed. However, it is important to carefully evaluate the trade-offs between processing efficiency and data loss when implementing load shedding, as dropping important data can have significant consequences in certain applications. Overall, the integration of priority scheduling and load shedding could offer valuable insights into how to better manage congestion and optimize the performance of stream processing systems. As such, we plan to explore these methods in future work, with a focus on evaluating their effectiveness, identifying potential trade-offs, and identifying optimal configurations for specific use cases. We also intend to assess how storage and processing power affect SLO performance.

Acknowledgments

The research is supported by NSF CNS 1911012 grant and NSF CNS 2124908 grant.

References

[1] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, pp. 64–71, 01 2017.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] C. Qin, H. Eichelberger, and K. Schmid, "Enactment of adaptation in data stream processing with latency implications—a systematic literature review," *Information and Software Technology*, vol. 111, pp. 1 – 21, 2019.

[4] P. Kang, P. Lama, and S. U. Khan, "Slo-aware virtual rebalancing for edge stream processing," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*, 2021, pp. 126–135.

[5] C.-C. Hung, G. Ananthanarayanan, P. Bodík, L. Golubchik, M. Yu, V. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *ACM/IEEE Symposium on Edge Computing (SEC)*, October 2018.

[6] P. Bellavista, A. Corradi, A. Reale, and N. Ticca, "Priority-based resource scheduling in distributed stream processing systems for big data applications," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 363–370.

[7] T. Ajila and S. Majumdar, "Data driven priority scheduling on spark based stream processing," in *2018 IEEE/ACM 5th International Conference on Big Data Computing Applications and Technologies (BDCAT)*, 2018, pp. 208–210.

[8] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: An iot benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017, e4257 cpe.4257.

[9] A. d. S. Veith, M. Dias de Assunção, and L. Lefèvre, "Latency-aware strategies for deploying data stream processing applications on large cloud-edge infrastructure," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 445–456, 2023.

[10] M. Dayarathna and S. Perera, "Recent advancements in event processing," *ACM Computing Surveys (CSUR)*, vol. 51, pp. 1 – 36, 2018.

[11] H. Herodotou, L. Odysseos, Y. Chen, and J. Lu, "Automatic performance tuning for distributed data stream processing systems," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 3194–3197.

[12] *Data Canvas. Sense your city: Data art challenge.* [Online]. Available: http://datacanvas.org/sense-your-city/

[13] B. Donovan and D. Work, *New York City Taxi Trip Data (2010-2013)*, 2016. [Online]. Available: https://doi.org/10.13012/J8PN93H8

[14] A. C. Cameron and P. K. Trivedi, *Regression Analysis of Count Data*, 2nd ed., ser. Econometric Society Monographs. Cambridge University Press, 2013.

[15] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *2017 IEEE International Conference on Edge Computing (EDGE)*, 2017, pp. 47–54.

[16] A. Araldo, A. D. Stefano, and A. D. Stefano, "Resource allocation for edge computing with multiple tenant configurations," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1190–1199.

[17] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, and M. Satyanarayanan, "Towards scalable edge-native applications," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, ser. SEC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 152–165.

[18] J. S. Van Der Veen, B. Van Der Waaij, E. Lazovik, W. Wijbrandi, and R. J. Meijer, "Dynamically scaling apache storm for the analysis of streaming data," in *2015 IEEE First International Conference on Big Data Computing Service and Applications*, 2015, pp. 154–161.

[19] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy, "Dhalion: Self-regulating stream processing in heron," *Proc. VLDB Endow.*, vol. 10, no. 12, p. 1825–1836, Aug. 2017.

[20] V. Kalavri, J. Liagouris, M. Hoffmann, D. Dimitrova, M. Forshaw, and T. Roscoe, "Three steps is all you need: Fast, accurate, automatic scaling decisions for distributed streaming dataflows," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 783–798.

[21] A. Slo, S. Bhowmik, and K. Rothermel, "Espice: Probabilistic load shedding from input event streams in complex event processing," in *Proceedings of the 20th International Middleware Conference*, ser. Middleware '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 215–227.

[22] ——, "State-aware load shedding from input event streams in complex event processing," *IEEE Transactions on Big Data*, vol. 8, no. 5, pp. 1340–1357, 2022.

[23] B. Zhao, N. Q. Viet Hung, and M. Weidlich, "Load shedding for complex event processing: Input-based and state-based techniques," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1093–1104.

[24] J. Wolfrath and A. Chandra, "Efficient transmission and reconstruction of dependent data streams via edge sampling," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2022, pp. 47–57. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IC2E55432.2022.00013