UDC 004.42
DOI: 10.15587/2706-5448.2025.325717

Artem Bashtovyi,
Andrii Fechan

DEVELOPMENT OF A STANDARDIZED APPROACH FOR EVALUATING BUSINESS INSIGHTS IN STREAM PROCESSING SYSTEMS BASED ON TECHNICAL METRICS

The object of research is the benchmarking process of stream processing frameworks, specifically evaluating the impact of Service Level Objectives (SLOs) in real-time data processing systems.

One of the most problematic aspects is the lack of standardization in SLO definitions, which leads to inconsistencies between technical performance indicators (latency, throughput) and business objectives. Additionally, existing benchmarking methodologies primarily assess technical metrics without considering their business relevance.

In the course of the study, experimental methods were used to analyze the relationship between latency and throughput under varying load conditions. A series of experiments were conducted with a Kafka Streams-based stream processing setup, modifying workload parameters and resource constraints.

The results obtained demonstrate the nonlinear relationship between latency and throughput. Increasing event rates can either enhance or degrade performance depending on resource constraints and Kafka Streams' commit interval settings. The findings demonstrate that under stable conditions, latency decreases from 21 s to 6.2 s while throughput increases from 0.6 ops/sec to 72 ops/sec. When computational bottlenecks are introduced, latency spikes to 349 s and throughput drops to 32 ops/sec, highlighting performance degradation. Conversely, distributed processing reduces latency to 11 s and increases throughput to 169.9 ops/sec. While higher loads generally improve throughput, excessive processing delays can unexpectedly reduce it due to resource contention.

These insights provide a foundation for dynamic SLO adjustments to optimize real-time data processing efficiency. The presented approach helps to avoid generalized and inefficient methods for measuring the performance of stream processing frameworks.

Keywords: *benchmarking, distributed systems, performance measurement, SLO (service level objectives), real-time processing.*

Received: 01.01.2025
Received in revised form: 04.03.2025
Accepted: 21.03.2025
Published: 31.03.2025

How to cite
Bashtovyi, A., Fechan, A. (2025). Development of a standardized approach for evaluating business insights in stream processing systems based on technical metrics. Technology Audit and Production Reserves, 2 (2 (82)), 15–20. https://doi.org/10.15587/2706-5448.2025.325717

1. Introduction

In recent years, the demand for real-time data processing has led to the rising popularity of stream processing frameworks, which provide the necessary infrastructure to handle continuous data streams for various applications Stream processing frameworks such as Apache Kafka, Apache Flink, and Apache Spark [1]. Streaming have rapidly gained popularity due to their capacity to handle real-time data, which is essential for applications in sectors like IoT, finance, and social media [2, 3]. These systems facilitate continuous data analysis, responding to the pressing demands for scalable and low-latency data handling in modern digital environments [4, 5]. The value of real-time processing is particularly highlighted in IoT ecosystems, where massive data streams from devices require immediate processing to ensure timely decision-making and reliability [6]. Stream processing allows for dynamic responses and insights in financial and social media analytics, where data arrives continuously from diverse sources. Most stream processing applications address business challenges, making performance metrics essential for evaluation. Comprehensive benchmarking provides insights into system reliability, yet assessing fault tolerance and scalability from a non-technical perspective remains challenging. While existing studies track metrics like recovery time and latency, they rarely connect them to business objectives or user experience. Service Level Objectives (SLOs) and Service Level Agreements (SLAs) define performance goals, typically based on latency and throughput, to ensure system efficiency. These metrics are crucial in stream processing as they ensure that the system can handle large, continuous data streams effectively. SLOs help stream processing systems maintain stable and reliable performance, which is important for real-time applications like traffic monitoring and financial analytics. These applications rely on fast data processing to function effectively. SLO metrics ensure that systems meet the required performance levels, helping to connect technical performance with business needs**.** For instance, an e-commerce platform might have an SLO to process customer orders within a second to enhance user experience and operational efficiency. Meeting these SLOs helps in achieving business goals such as customer satisfaction, operational efficiency, and competitive advantage [7]. Despite significant advancements in benchmarking methodologies, existing studies often fail to bridge the gap between technical performance indicators and business-driven service level objectives. A survey was conducted comparing existing high-quality benchmarking solutions for stream processing applications (Table 1) by assessing basic metrics and SLO-related features.

ISSN-L 2664-9969; E-ISSN 2706-5448

Table 1

Research on existing stream processing benchmarking papers

Article Title	Frameworks	Latency	E2E latency	Throughput	CPU utilization	Memory usage	SLO metrics
Stream bench: towards benchmarking distributed stream computing frameworks	Spark, Flink	✓	✓	✓	✓	–	–
ShuffleBench: a benchmark for large-scale data shuffling	Flink, Kafka	✓	–	✓	–	–	–
Open stream processing benchmark:	Kafka Streams	✓	✓	✓	✓	✓	–
Extensive analysis of distributed frameworks	Flink	–	–	–	–	–	–
Yahoo streaming benchmarks (YSB) [8]	Flink, Spark	✓	✓	✓	✓	–	–
Enterprise stream processing benchmark (ESPBench)	Flink, Spark	✓	–	✓	✓	✓	–
Theodolite: scalability benchmarking of distributed stream processing engines in microservice architectures	Kafka, Flink	✓	–	✓	✓	✓	✓

In the recent study [9], authors review existing stream processing benchmarks with an all-inclusive analysis of five characteristics: benchmark strategy, used workloads, data ingestion, supported systems under test (SUT), and set of metrics.

Another study [10] propose Stream Bench which aims to standardize the evaluation of stream computing systems by defining essential criteria for comparison. Their approach provides insights into the performance, fault tolerance, and scalability of distributed stream processing frameworks. One more research examine fault recovery mechanisms across different stream processing frameworks [11]. The authors introduce new fault-tolerance workloads in OSPBench, allowing for fine-grained analysis of metrics such as downtime, recovery speed, peak latency, and throughput degradation during various failure types, including master, worker, and driver failures. The study highlights how different frameworks perform at least once and exactly once semantics and suggests that future work could focus on adaptive checkpointing strategies and enhancing high-availability setups to further optimize recovery times. The authors introduced ShuffleBench, a novel benchmarking tool designed to evaluate the performance of stream processing frameworks like Flink, Kafka Streams, Hazelcast, and Spark in large-scale data shuffling scenarios [12]. The study focuses on key metrics like throughput, latency, and scalability, providing insights into the performance characteristics of these frameworks in cloud-native environments. It highlights that Flink achieves the highest throughput while Hazelcast demonstrates the lowest latency. However, the study points out that they did not use realistic production deployments. The paper explores a variety of fault-tolerance metrics in the context of worker node failures, recurrent failures, and recovery time under different failure scenarios. Specifically, it measures SLOs related to recovery time, assessing how latency and fault resilience affect system responsiveness, which in turn, represents how to minimize user-perceived latency disruptions, ensuring a resilient user experience. Future work is recommended to explore the relationship between throughput, latency, and fault tolerance, as well as to support non-uniform data distributions and extended industrial use cases. In [13] present ESPBench, a benchmarking framework for evaluating enterprise stream processing architectures by integrating streaming data with structured business data, a key gap in existing benchmarks. It assesses Apache Spark, Flink, and Hazelcast Jet using Apache Beam and covers data ingestion, query execution, validation, and performance evaluation. The benchmark automates execution and result validation, comparing latency, throughput, and system load. A major contribution is its ability to integrate real-time sensor data with business databases, making it highly relevant for enterprise and Industry 4.0 applications. The findings in [14] introduce custom metrics specific to fragmented state recovery, utilizing SLOs on latency and resilience per fragment to uphold business-critical response times. They gauge the resilience of Kafka and Flink applications by monitoring time-bound state recovery processes, ensuring rapid system reactivity under high data loads. Recovery times are measured to determine compliance with SLOs focused on maintaining user engagement levels and service continuity. Research conducted in [15, 16] emphasize SLA metrics related to micro-latency for fine-grained operations within Kafka and Flink frameworks. By measuring 99-th percentile latency, the study addresses how minute delays impact QoE (Quality of Experience), ensuring SLA compliance in high-priority applications. These micro-latency SLAs are critical for applications where even slight delays can impact user satisfaction or engagement. A previous study [17] measure SLO compliance for checkpoint latency, assessing Kafka and Flink frameworks on their ability to resume service after fault disruptions. By focusing on latency for system recovery, this study gauges how checkpoint efficiency contributes to SLO metrics critical for operational stability. SLOs in this context ensure the real-time continuity of applications with minimal disruptions. In another study [18] authors utilize the Theodolite benchmarking framework to assess the scalability of Apache Flink, Kafka Streams, and Hazelcast Jet. The study employs SLO-based metrics, such as the consumer lag trend, to evaluate message queuing within Kafka, setting a 1 % increase threshold as acceptable for system performance. Additionally, it introduces an SLO to ensure dropped messages remain below 1 % during high-load scenarios, particularly in short-window aggregations. As it is possible to see, majority of the studies do not consider tracking SLO at all.

This aim of this research is to develop a standardized benchmarking methodology that integrates SLO-based performance evaluation into stream processing. By dynamically adjusting SLO thresholds based on system conditions, our approach ensures a balance between system efficiency and business priorities.

2. Materials and Methods

The object of research is the benchmarking process of stream processing frameworks, specifically evaluating the impact of SLOs in real-time data processing systems. An experimental research methodology was employed, systematically analyzing the correlation between latency and throughput under varying system loads. Three controlled experiments are conducted in a Kafka Streams environment: Stable Operation, Artificial bottleneck, Distributed processing. By comparing these results, let's provide a comprehensive evaluation of how different operational conditions impact stream processing efficiency and business-aligned performance measures. This approach ensures a structured and repeatable methodology for benchmarking, offering practical insights into optimizing real-time stream processing applications.

Benchmarking stream processing frameworks requires a comprehensive evaluation of how SLOs influence real-time data processing

systems. Understanding the relationship between technical performance metrics and business-driven goals is essential for ensuring that benchmarking methodologies provide meaningful insights. This study explores the alignment of key SLO metrics, such as latency and throughput, with both operational efficiency and business priorities.

SLOs set measurable expectations to ensure stream processing frameworks meet both operational and business goals. As is mentioned previously, despite their importance, most benchmarking studies do not explicitly define SLOs, even though they track key performance metrics. A well-defined SLO enhances technical resilience while aligning system behavior with user-centered outcomes like Quality of Experience (QoE). Without clear SLO definitions, linking benchmarking results to business impact remains difficult. The majority of benchmarking frameworks assess SLO compliance by tracking latency and throughput. Some studies have aimed to bridge the gap between performance metrics and effective SLO management. Findings from [19] reveal development of a real-time SLO monitoring framework using machine learning for anomaly detection and predictive analytics to prevent violations. Other researchers [20] explored resource allocation in multi-tenant environments with SLOs, emphasizing latency and a unique throughput metric called "juice". Their method balances minimum juice with maximum latency to optimize performance. Automated SLO enforcement through self-adaptive strategies is presented in [21], it highlights adjusting thresholds based on real-time metrics such as throughput, latency, and CPU usage to improve resource allocation. Other researchers [22] studied SLOs for optimizing DSP application placement, and minimizing end-to-end latency and power consumption in edge computing environments using empirical data and 3GPP guidelines. Despite progress, studies mainly emphasize technical metrics like latency and throughput, neglecting business-focused measures such as customer satisfaction and cost efficiency. This creates a gap between SLO monitoring and real-world business outcomes, as current studies often overlook agile business needs and varying metric priorities. Additionally, inconsistencies in SLO representation such as differences in units, scales, and measurement ranges – make benchmarking across stream processing systems difficult. Standardization is needed to improve comparability and usability for product owners and stakeholders. To address these gaps, our research focuses on the two most widely used SLO metrics – latency and throughput – to analyze their impact under varying conditions. Our experiments examine the correlation between these metrics to determine if SLO calculations reliably reflect system performance. By assessing these relationships in a controlled environment, let's provide insights into the validity of SLO measurements in real-world scenarios. Let's summarize the research of the existing papers in Fig. 1.

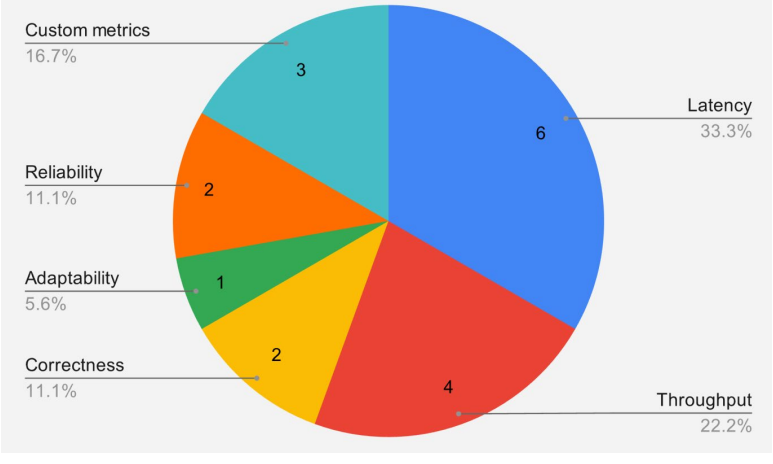

Fig. 1. Metrics used for SLO calculations across the papers

2.1. Setup

To evaluate the impact of different operational conditions on stream processing performance, let's design three experiments simulating stable operation, computational bottlenecks, and distributed processing. Our experimental setup replicated realistic virtualized environments typical for stream processing deployments. Using Docker Compose, it is possible to orchestrate containers within a Docker-based virtual machine, allocating 8 GB RAM, 40 GB disk space, and an 8-core CPU (4 performance and 4 efficiency cores). For data processing, let's employ Apache Kafka as the message broker and Apache Kafka Streams as the client library, integrated with Spring Boot. JMX technology was used to monitor key metrics, while Prometheus and Grafana handled metric collection and visualization. For the simulation of an event processing application, it is possible to create a simple Kafka Streams topology containing mapping and joining events from different topics and grouping records. Let's use our custom synthetic data generator with varying rates of generating events. To evaluate the impact of different operational conditions on stream processing performance, let's design three experiments simulating stable operation, computational bottlenecks, and distributed processing. In the first experiment, a regular and stable event processing rate was simulated to represent optimal operating conditions for a stream processing application. The second experiment introduced random delays ranging from 1 to 5 seconds during the mapping stage, designed to emulate scenarios where event-producing applications perform complex computations. Finally, the third experiment involved the deployment of a second processing instance with an identical topology, simulating a distributed processing environment to evaluate its impact on system performance.

2.2. Throughput and latency definitions

The throughput, metric represents the number of events processed within a given time frame and serves as a standard performance indicator for most event-driven applications. It is possible to define throughput as the number of processed events per second per instance, reflecting the system's ability to handle continuous data streams efficiently. Higher throughput typically indicates better system scalability, but it must be balanced with other factors such as latency and resource utilization. Latency measures the time delay between when an event is generated or received and when it is fully processed to produce a result. It is a crucial metric for assessing system responsiveness in event-based applications. Increased latency can negatively impact user experience, delay decision-making processes, and lead to missed business opportunities in real-time applications such as financial trading, IoT monitoring, or customer-facing services. It is possible to define latency (Formula (1)) as the average time taken to process each event from its entry into the system until it completes processing on an instance. In high-throughput environments, reducing latency while maintaining system stability is a key challenge, making it essential to optimize processing pipelines, resource allocation, and failure recovery mechanisms.

In order to define latency and throughput from a business perspective it is possible to take into account a normal scenario which defines the referential values for both metrics. These values are considered to be default or regular and defined as normal in our formulas. While raw latency and throughput values provide insight into system performance, they do not account for variations in normal operating conditions. To address this, let's introduce SLO Latency (Formula (2)) and SLO. Throughput (Formula (3)) as relative metrics that compare current performance to a predefined baseline. SLO Latency normalizes the observed latency against a reference scenario, providing a scalable measure of processing delays under different conditions. Similarly,

ISSN-L 2664-9969; E-ISSN 2706-5448

SLO Throughput expresses the system's ability to maintain or improve event processing efficiency compared to a standard operating load:

$$\text{Latency} = T_{end} - T_{start}, \quad (1)$$

where T_{start} – time when an event is consumed by an application; T_{end} – time when the event is fully processed by a streaming application.

Definition of SLO latency:

$$SLO_{latency} = \left(\frac{Latency}{Latency_{normal}} \right), \quad (2)$$

where *Latency* – measured latency; $Latency_{normal}$ – latency in a normal scenario, when 1 event per second is published.

Definition of SLO throughput:

$$SLO_{Throughput} = \left(\frac{Throughput}{Throughput_{normal}} \right), \quad (3)$$

where *Throughput* – measured throughput; $Throughput_{normal}$ – throughput in a normal scenario, when 1 event per second is published.

2.3. Evaluation

For our experiment, let's design a Kafka Streams-based infrastructure to evaluate stream processing performance. Kafka Streams was selected as the core client library due to its seamless integration with Kafka ecosystems, lightweight architecture, support for fault tolerance, stateful processing, and windowing, making it well-suited for real-time data processing in agile business environments. The experimental setup consisted of a producer application, consumer application, message broker, metrics aggregator, and visualization tools. Metrics were collected using JMX exporters, which exposed relevant data from consumer applications and message brokers to Prometheus for monitoring. The producer, a synthetic data generator, published events at a configurable rate of 1,200,500 events for two input topics. The consumer application implemented a Kafka Streams topology with two source processors, that handle input topics, multiple intermediary processors, including the joining of two topics by shared identifier, and a final sink processor where results were recorded. In the experiment where two instances were used, let's deploy the same application with the same instance id, thereby instances were within one consumer group and the load was distributed across both of them. Prometheus was used to track metrics of stream processing applications and eventually. Results were aggregated to Grafana.

3. Results and Discussion

It is possible to document the experiment results in Table 2.

Table 2

SLO-latency and SLO-throughput grouped by experiment

Experiment	Events	Latency, sec	Throughput, ops/sec	SLO latency	SLO throughput
Normal condition	1	21	0.6	1	1
Normal condition	200	16	43	0.76	71.67
Normal condition	500	6.2	72	0.3	120
Bottleneck on mapping	1	25	0.8	1.19	1.33
Bottleneck on mapping	200	182	60.7	8.67	101.17
Bottleneck on mapping	500	349	32	16.62	53.33
Multiple instances	1	22	1.6	1.05	2.67
Multiple instances	200	21.4	102.5	1.02	170.83
Multiple instances	500	11	169.9	0.52	283.17

The first experiment evaluated event processing rates under optimal conditions, where the system operated without external disruptions, providing a baseline for understanding performance scaling. It literally simulates normal system behavior under moderate and stable load, when business operates as usual. When processing a single event, the system showed a latency of 21 seconds and a throughput of 0.6 ops/sec. As the event volume increased to 500, latency significantly decreased to 6.2 seconds, while throughput improved to 72 ops/sec, indicating that the system effectively introduces optimizations under high-load scenarios. These results align with existing literature on stream processing, which highlights the role of commit intervals in balancing fault tolerance and performance efficiency. In our research, Kafka's commit interval played a crucial role, as shorter intervals guarantee data consistency but introduce additional processing overhead, while longer intervals enhance throughput at the cost of potential data loss during faults. Our findings suggest that while increased event loads typically improve throughput, the performance gains are not strictly linear, revealing inherent system constraints and diminishing returns at higher event rates. This observation indicates that stream processing applications must fine-tune commit strategies based on workload intensity to maintain an optimal balance between data consistency and processing efficiency. To better understand system behavior under more complex processing conditions, the second experiment introduced artificial delays ranging from 1 to 5 seconds in the event mapping process, simulating real-world computational loads. This modification led to an increase in latency to 25 seconds for a single event, while throughput improved slightly to 0.8 ops/sec. However, as the event volume reached 500, latency surged dramatically to 349 seconds, while throughput dropped by nearly 50 % compared to 200 events. The results indicate that while moderate loads initially improve throughput due to system optimizations, excessive processing delays cause severe performance degradation. This effect is similar in fault-tolerant stream processing, where frequent commit intervals ensure state consistency but introduce significant computational overhead. Our findings align with previous research suggesting that while stateful stream processing can enhance resilience, it also imposes constraints on scalability, particularly in latency-sensitive applications. The experiment defines the necessity of carefully balancing commit intervals, computational complexity, and event load to prevent bottlenecks that could impact system reliability and responsiveness. In the third experiment, a second processing instance was introduced to simulate a distributed setup, significantly improving system performance. With two instances handling the workload, latency for 500 events decreased to 11 seconds, while throughput increased to 169.9 ops/sec. This improvement confirms that distributed processing effectively mitigates bottlenecks and enhances scalability. However, our results also reveal that the relationship between latency and throughput remains non-linear even in distributed environments. While additional instances help alleviate individual node constraints, they also introduce system-wide complexities such as state synchronization and rebalancing, which can impact overall efficiency. Furthermore, our findings highlight the challenges of defining standardized SLO metrics, as system variations and workload differences make direct comparisons across diverse applications difficult. Traditional performance benchmarks, which primarily rely on raw latency and throughput measurements, may fail to capture their relative significance in different execution contexts. By normalizing performance metrics and incorporating business-specific priorities, SLO evaluations can better represent real-world service expectations and application behavior. These insights have significant practical implications for organizations deploying stream processing frameworks, as they can guide resource allocation strategies, fault-tolerance mechanisms, and performance optimizations tailored to specific business needs.

Ultimately, it is possible to conclude the necessity of advancing benchmarking methodologies beyond conventional metrics is necessary. Specifically, it is possible to advocate for adaptive, context-aware approaches that reflect industry demands and operational realities. The solution would be presenting a flexible SLO formula which would allow businesses to specify details and configuration coefficients to adapt assessment based on the system business requirements. Moreover, future work must focus on presenting a generic, normalized, and standardized approach for SLO estimation within the stream processing application.

Practical Significance: The results of this research can be applied in multiple practical contexts, including real-time data processing, financial transaction systems, and cloud-based stream processing services. Companies can use the proposed methodology to better align technical performance with business expectations by dynamically adjusting SLO thresholds based on system conditions. Additionally, companies that rely on distributed stream processing frameworks can optimize resource allocation because they are able to monitor extra metrics based on the stakeholder's expectations.

Research Limitations: This research was conducted in a controlled environment with predefined configurations, meaning the results may not fully capture variations in real-world deployments. Factors such as network latency, heterogeneous infrastructure, and dynamic workload fluctuations were not extensively modeled. Additionally, the research primarily focused on Kafka Streams, and while the findings are applicable to other stream processing frameworks, further validation is needed for broader generalization. Future work should focus on developing a universal, adaptable SLO calculation formula to ensure consistency in evaluating stream processing applications across diverse environments, SLO parameters beyond latency and throughput.

Impact of Martial Law in Ukraine: The ongoing martial law in Ukraine has not influenced the research process significantly. Power outages and unstable internet connectivity created minor challenges in conducting continuous benchmarking tests, occasionally disrupting data collection. Nevertheless, the major time spend on the research was not disturbed by other factors.

4. Conclusion

This research provides new insights into benchmarking stream processing frameworks, particularly focusing on the impact of SLO on real-time data systems. By conducting multiple experimental scenarios, the non-linear relationship between latency and throughput is identified, highlighting that increasing event rates can either improve or degrade system performance depending on resource constraints. These findings demonstrate that fixed SLO definitions may not accurately reflect real-world conditions, necessitating dynamic SLO adjustments based on system state and workload intensity. The research has shown that under stable conditions, latency can be reduced from *21 seconds to 6.2 seconds* while throughput increases from *0.6 ops/sec to 72 ops/sec* as the event volume grows. However, introducing processing bottlenecks results in significant performance degradation, with latency reaching *349 seconds* and throughput dropping to *32 ops/sec*. In contrast, distributed processing mitigates these bottlenecks, reducing latency to *11 seconds* and increasing throughput to *169.9 ops/sec* under similar load conditions. These quantitative findings confirm that system performance is highly dependent on resource allocation strategies, commit intervals, and load balancing mechanisms. From a practical perspective, these findings can help organizations refine their approach to stream processing by dynamically adjusting SLOs based on real-time conditions. This is particularly relevant for industries relying on real-time analytics, IoT data streams, and financial transactions, where consistent performance is critical. By moving away from rigid benchmarks and embracing adaptive SLO strategies, businesses can achieve better resource management and fault tolerance.

While this research provides a strong foundation, it is primarily focused on Kafka Streams, meaning further studies should extend these findings to other frameworks like Apache Flink and Spark Streaming. Future work should also explore how machine learning techniques can be leveraged to predict system behavior and dynamically adjust SLOs, ensuring more accurate and standardized performance assessments across different real-time processing environments.

Conflict of interest

The authors declares that they have no conflict of interest in relation to this study, including financial, personal, authorship or other, which could affect the study and its results presented in this article.

Financing

The study was performed without financial support.

Data availability

Data will be made available on reasonable request.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

References

1. Tantalaki, N., Souravlas, S., Roumeliotis, M. (2019). A review on big data real-time stream processing and its scheduling techniques. *International Journal of Parallel, Emergent and Distributed Systems, 35 (5),* 571–601. https://doi.org/10.1080/17445760.2019.1585848
2. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I. (2013). Discretized streams: fault-tolerant streaming computation at scale. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 423–438. https://doi.org/10.1145/2517349.2522737
3. Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., Campbell, R. H. (2017). Samza: Stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment, 10 (12),* 1634–1645. https://doi.org/10.14778/3137765.3137770
4. Saxena, S., Gupta, S. (2017). *Practical real-time data processing and analytics: distributed computing and event processing using Apache Spark, Flink, Storm, and Kafka.* Packt Publishing Ltd., 360.
5. Raptis, T. P., Passarella, A. (2023). A Survey on Networked Data Streaming With Apache Kafka. *IEEE Access, 11,* 85333–85350. https://doi.org/10.1109/access.2023.3303810
6. Dias de Assunção, M., da Silva Veith, A., Buyya, R. (2018). Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications, 103,* 1–17. https://doi.org/10.1016/j.jnca.2017.12.001
7. Kalim, F. (2020). *Satisfying service level objectives in stream processing systems.* [Doctoral dissertation; University of Illinois at Urbana-Champaign]. Available at: https://www.ideals.illinois.edu/items/116227
8. *Benchmarking Streaming Computation Engines at Yahoo!* Yahoo. Available at: http://yahooeng.tumblr.com/post/135321837876/benchmarkingstreaming-computation-engines-at
9. Wang, Y., Boissier, M., Rabl, T. (2024). A survey of stream processing system benchmarks. *Proceedings of the 16th TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2024).* Guangzhou.
10. Lu, R., Wu, G., Xie, B., Hu, J. (2014). Stream Bench: Towards Benchmarking Modern Distributed Stream Computing Frameworks. *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing,* 69–78. https://doi.org/10.1109/ucc.2014.15
11. van Dongen, G., Poel, D. V. D. (2021). A Performance Analysis of Fault Recovery in Stream Processing Frameworks. *IEEE Access, 9,* 93745–93763. https://doi.org/10.1109/access.2021.3093208
12. Henning, S., Vogel, A., Leichtfried, M., Ertl, O., Rabiser, R. (2024). ShuffleBench: A Benchmark for Large-Scale Data Shuffling Operations with Distributed Stream Processing Frameworks. *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, 2–13. https://doi.org/10.1145/3629526.3645036

ISSN-L 2664-9969; E-ISSN 2706-5448

13. Hesse, G., Matthies, C., Perscheid, M., Uflacker, M., Plattner, H. (2021). ESPBench: The Enterprise Stream Processing Benchmark. *Proceedings of the ACM/SPEC International Conference on Performance Engineering,* 201–212. https://doi.org/10.1145/3427921.3450242

14. Xu, H., Liu, P., Ahmed, S. T., Da Silva, D., Hu, L. (2023). Adaptive Fragment-Based Parallel State Recovery for Stream Processing Systems. *IEEE Transactions on Parallel and Distributed Systems, 34 (8),* 2464–2478. https://doi.org/10.1109/tpds.2023.3251997

15. Dongen, G. V. (2021). *Open stream processing benchmark: An extensive analysis of distributed stream processing frameworks.* [Doctoral dissertation; Ghent University].

16. van Dongen, G., Steurtewagen, B., Van den Poel, D. (2018). Latency Measurement of Fine-Grained Operations in Benchmarking Distributed Stream Processing Frameworks. *2018 IEEE International Congress on Big Data (BigData Congress).* San Francisco, 247–250. https://doi.org/10.1109/bigdatacongress.2018.00043

17. Jayasekara, S., Karunasekera, S., Harwood, A. (2021). Optimizing checkpoint-based fault-tolerance in distributed stream processing systems: Theory to practice. *Software: Practice and Experience, 52 (1),* 296–315. https://doi.org/10.1002/spe.3021

18. Henning, S., Hasselbring, W. (2021). Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures. *Big Data Research, 25,* 100209. https://doi.org/10.1016/j.bdr.2021.100209

19. Kavuri, S., Narne, S. (2020). Implementing Effective SLO Monitoring in High-Volume Data Processing Systems. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 5 (6),* 558–578. https://doi.org/10.32628/cseit206479

20. Kalim, F., Xu, L., Bathey, S., Meherwal, R., Gupta, I. (2018). Henge: Intent-driven multi-tenant stream processing. *Proceedings of the ACM Symposium on Cloud Computing,* 249–262. https://doi.org/10.1145/3267809.3267832

21. Griebler, D., Vogel, A., De Sensi, D., Danelutto, M., Fernandes, L. G. (2020). Simplifying and implementing service level objectives for stream parallelism. *The Journal of Supercomputing, 76 (6),* 4603–4628. https://doi.org/10.1007/s11227-019-02914-6

22. Kayser, C., Dias de Assunção, M., Ferreto, T. (2024). Lapse: Latency & Power-Aware Placement of Data Stream Applications on Edge Computing. *Proceedings of the 14th International Conference on Cloud Computing and Services Science.* SciTePress, 358–366. https://doi.org/10.5220/0012737400003711

✉***Artem Bashtovyi,*** *PhD Student, Assistant, Department of Software, Lviv Polytechnic National University, Lviv, Ukraine, e-mail: artem.v.bashtovyi@lpnu.ua, ORCID: https://orcid.org/0000-0003-4304-8605*

Andrii Fechan, *Doctor of Technical Sciences, Professor, Department of Software, Lviv Polytechnic National University, Lviv, Ukraine, ORCID: https://orcid.org/0000-0001-9970-5497*

✉*Corresponding author*