МАТЕМАТИЧНЕ ТА КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ

MATHEMATICAL AND COMPUTER MODELING

UDC 004.42

EVALUATING FAULT RECOVERY IN DISTRIBUTED APPLICATIONS FOR STREAM PROCESSING APPLICATIONS: BUSINESS INSIGHTS BASED ON METRICS

Bashtovyi A. V. – Post-graduate student of the Department of Software, Lviv Polytechnic National University, Lviv, Ukraine.

Fechan A. V. – Dr. Sc., Professor of the Software Department, Lviv Polytechnic National University, Lviv, Ukraine.

ABSTRACT

Context. Stream processing frameworks are widely used across industries like finance, e-commerce, and IoT to process real-time data streams efficiently. However, most benchmarking methodologies fail to replicate production-like environments, resulting in an incomplete evaluation of fault recovery performance. The object of this study is to evaluate stream processing frameworks under realistic conditions, considering preloaded state stores and business-oriented metrics.

Objective. The aim of this study is to propose a novel benchmarking methodology that simulates production environments with varying disk load states and introduces SLO-based metrics to assess the fault recovery performance of stream processing frameworks.

Method. The methodology involves conducting a series of experiments. The experiments were conducted on synthetic data generated by application using Kafka Streams in a Docker-based virtualized environment. The experiments evaluate system performance under three disk load scenarios: 0%, 50%, and 80% disk utilization. Synthetic failures are introduced during runtime, and key metrics such as throughput, latency, and consumer lag are tracked using JMX, Prometheus, and Grafana. The Business Fault Tolerance Impact (BFTI) metric is introduced to aggregate technical indicators into a simplified value, reflecting the business impact of fault recovery.

Results. The developed indicators have been implemented in software and investigated for solving the problems of Fisher's Iris classification. The approach for evaluating fault tolerance in distributed stream processing systems has been implemented, additionally, the investigated effect on system performance under different disk utilization.

Conclusions. The findings underscore the importance of simulating real-world production environments in stream processing benchmarks. The experiments demonstrate that disk load significantly affects fault recovery performance. Systems with disk utilization exceeding 80% show increased recovery times by 2.7 times and latency degradation up to fivefold compared to 0% disk load. The introduction of SLO-based metrics highlights the connection between system performance and business outcomes, providing stakeholders with more intuitive insights into application resilience. The findings underscore the importance of simulating real-world production environments in stream processing benchmarks. The BFTI metric provides a novel approach to translating technical performance into business-relevant indicators. Future work should explore adaptive SLO-based metrics, framework comparisons, and long-term performance studies to further bridge the gap between technical benchmarks and business needs.

KEYWORDS: fault-tolerance, Kafka Streams, benchmarking, distributed systems, performance measurement, stream processing, SLO(Service level objectives).

ABBREVIATIONS

BFTI – Business Fault Tolerance Impact;
SLA – Service Level Agreement;
SLO – Service Level Objective;
IoT – Internet of Things;
JMX – Java Management Extensions;
CPU – Central Processing Unit;
RAM – Random Access Memory;
e2e – End-to-End;
API – Application Programming Interface;
VM – Virtual Machine;
I/O – Input/Output.

NOMENCLATURE

BFTI is a Business Fault Tolerance Impact, aggregated metric for evaluating the business impact of fault tolerance based on SLO indicators;

D is a disk usage for the state store in stream processing application.

$L(t)$ is a lag at a specific point of time t;

$L_{normalised}(t)$ is a normalised consumer lag at a specific t point;

$L_{\max}$ is a max allowed lag defined by stakeholders;

Latency (t) is a event processing delay at a specific t point;

DOI 10.15588/1607-3274-2025-3-2

$\Delta Latency$ is a latency difference;

$Latency_{actual}$ is an actual latency limit during faults;

$Latency_{SLO}$ is a maximum allowed latency as per the SLO;

n is a number of measurements of lag;

t_i is a time point during the total measured time;

T_{total} is the total time of an experiment;

$Throughput(t)$ is a number of events processed by the system at a specific t point;

$\Delta Throughput$ is a throughput difference;

$Throughput_{normal}$ is an average number of events/records processed per second when the system is operating normally;

$Throughput_{fault}$ is a number of events processed per second during the fault period;

$\Delta(t)$ is a vector of normalized deviations for metrics;

Δt is an interval between measurements;

S is a distributed processing system;

V_{lag} is a SLO based lag;

w_1 is a weight coefficient for V_{lag} in BFTI formula;

w_2 is a weight coefficient for calculated throughput in BFTI formula;

w_3 is a weight coefficient for latency in BFTI formula;

X_{target} is a vector target values for metrics;

$X(t)$ is a single value that represents how systems performs across different(latency, throughput, consumer lag) specific t point.

INTRODUCTION

In today's data-driven world, stream processing frameworks have become essential for handling real-time data streams across industries such as finance, e-commerce, and IoT. Fragkoulis et al.in their work [1] talk about how increasing volume and velocity of data have driven significant advancements in stream processing technologies, including the adoption of serverless computing, edge streaming, enhanced query capabilities, and hardware acceleration. While serverless architectures offer flexibility and cost efficiency, they also introduce challenges in state management and low-latency processing. Similarly, edge computing reduces latency by bringing processing closer to data sources but requires lightweight techniques, particularly for IoT environments [2]. Hardware acceleration using GPUs, FPGAs, and near-memory computing further enhances performance, making stream processing frameworks increasingly powerful and widely adopted. As stream processing applications gain traction, ensuring their performance, reliability, scalability, and fault tolerance becomes critical. They are even considered as a supportive tool in migration to distributed systems[3]. Metrics such as throughput, latency, and resource utilization are key indicators of system efficiency and stability, aiding in detecting and mitigating failures that could impact business operations. Benchmarking is a common approach used to evaluate these metrics, providing insights into system behavior under various conditions. Fault tolerance, in particular, is a crucial aspect of stream processing, enabling systems to recover from failures and maintain uninterrupted data processing [4]. Techniques such as checkpointing, state recovery, and replication are widely used to ensure data continuity and resilience [5][6]. This resilience is increasingly essential as applications expand in scale and complexity, underscoring the need for robust benchmarking to evaluate recovery times and fault-tolerance effectiveness [7]. However, despite advancements in benchmarking methodologies, many existing studies focus on clean-state conditions, often using synthetic data that does not accurately reflect real-world production environments. Current benchmarking approaches often fail to consider the impact of preloaded state stores on system performance, leading to an incomplete understanding of how state accumulation affects latency and recovery times. The approach[8] presented by Van Dongen G, et al., lacks representation of real-world scenarios where state stores are progressively populated, impacting latency and recovery time. Another research on stream-processing cost tracking [9] in the same manner does not consider the effect of compound time on the system performance.

Introducing a benchmark methodology that simulates production-like environments, including preloaded state stores, would allow for a more accurate assessment of system performance under realistic load conditions, which is crucial for decision-making regarding resource allocation and infrastructure needs [10]. Our study addresses this gap by introducing a benchmarking methodology that evaluates stream processing performance under varying state loads, simulating real-world conditions more accurately. Additionally, while most research primarily tracks technical metrics such as latency and throughput, there is a lack of business-oriented insights that connect system performance with user experience and operational efficiency. To resolve the existing gap, we integrate SLO-based metrics, providing a framework that translates technical performance indicators into business-relevant insights.

The object of study is the process of evaluation of fault recovery in distributed stream processing applications under realistic conditions, considering preloaded state stores and business-oriented metrics.

The subject of study is benchmarking methodologies for assessing fault tolerance in stream processing frameworks, focusing on the impact of state store accumulation, synthetic failures, and business-driven SLO metrics.

The purpose of the work is to develop and validate a benchmarking methodology that simulates production environments with varying state loads, integrates SLO-based metrics, and provides insights into the business impact of fault recovery in stream processing applications.

1 PROBLEM STATEMENT

Let's assume that S distributed stream processing system is provided, which constantly ingests and processes events in real time. The system maintains local state in an embedded state store that resides on disk. The system's performance is defined by the following primary metrics:

© Bashtovyi A. V., Fechan A. V., 2025
DOI 10.15588/1607-3274-2025-3-2

– $Latency\,(t) \in R_{\geq 0}$;
– $Throughput\,(t) \in R_{\geq 0}$;
– $L\,(t) \in R_{\geq 0}$.

Each of these metrics can degrade under fault conditions and contributes to the system's overall performance loss. However, monitoring them separately requires detailed technical analysis, making it difficult for non-technical stakeholders to evaluate the system's health quickly and effectively.

Let's assume that target values for these metrics must be:

$$X_{target} = [\, Latency_{SLO}\,, Throughput_{normal}, L_{\max}\,]. \qquad (1)$$

Let's define the vector of normalized deviations from expected behavior:

$$\Delta(t) = [\Delta Latency\,(t), \Delta Throughput(t), L_{normalised}\,(t)]. \qquad (2)$$

This vector represents the normalized deviations from the target values. By aggregating these deviations into a single value, we aim to simplify the monitoring process. It is required to build a mathematical model of a single normalised value $X(t) \in R_{\geq 0}$ that encapsulates the system's overall performance at time t and provides a single value which represents value of three metrics in the system at point t. Which must be adjustable from the priorities and requirements perspective.

The second objective is to analyze the behavior of S under varying levels of disk usage D in the embedded state store. The research's goal is to evaluate how changes in disk state store utilization D influence the system's real-time performance metrics and the resulting value of $X(t)$, providing insights for optimizing resilient stream processing in production-like environments.

2 REVIEW OF THE LITERATURE

SLO metrics are commonly referenced in fault tolerance research, they are rarely explicitly defined or structured to provide meaningful insights for stakeholders. Most studies focus on system stability from an engineering perspective, overlooking how technical failures translate into business impacts such as service availability and user satisfaction [11]. Existing benchmarking methodologies do not simulate production environments where state stores are preloaded and continuously evolving.

There are studies that have explored benchmarking methodologies for stream processing frameworks, with a strong emphasis on scalability and fault tolerance. The paper [12] offers provides insights into the scalability of frameworks like Apache Flink, Kafka Streams, and Hazelcast Jet within cloud-native microservice architectures. The study focuses on scaling challenges, particularly in dynamic resource allocation and fault recovery. However, it primarily addresses short-term scalability and does not extensively explore fault tolerance under long-term operational conditions, where large state accumulation and multi-fault scenarios may arise. This limitation highlights the need for research that considers fault recovery in systems with extensive state persistence. Another study [13] provides a comprehensive classification of fault tolerance techniques in stream processing systems, emphasizing their importance in preventing erroneous results and system unavailability. The authors highlight that failures in processing nodes or communication networks can lead to severe disruptions, impacting user experience and causing financial losses. The study introduces an evaluation framework for fault tolerance mechanisms in Apache Flink assessing efficiency in failure recovery. Key future research directions include adaptive checkpointing, integration with modern hardware, and parallel recovery mechanisms. Despite these advancements, the study does not account for real-world scenarios where applications run continuously, accumulating state over time.

3 MATERIALS AND METHODS

In this section, we propose our own method for benchmarking stream processing applications based on the basic metrics. In the end, evaluation and experiment details are presented. As we discussed previously, stakeholders may be interested in knowing how stream processing systems perform in general without technical details in debt. We concentrate on the business-related SLO metric, which is supposed to be straightforward and representative of business and engineering needs. Despite the fact, that the basic metrics like throughput, and latency are less conductive; they represent core system performance. In the following sections, we define core metrics of the system which are used for formulating our SLO metric.

The throughput metric represents the number of events per certain time mark. The metric is considered to be a status quo for most event-based applications. We define throughput as a number of processed events per second on the instance in general. For our experiment, we had to understand the change in throughput under different states of the application. The change is calculated based on the difference between throughput under normal conditions and throughput when some parts of the system are under a fault. In this way, we can define how faults affect the throughput. The change is stated as throughput difference which is declared by the formula(3):

$$\Delta Throughput = Throughput_{normal} - Throughput_{fault}. \qquad (3)$$

The latency metric represents the time delay from when an event is generated or received to when it is fully processed and produces a result. In event-based applications, latency is a critical measure of system responsiveness. Increased latency impacts customer experience and can lead to missed business opportunities in real-time applications. We describe latency as the average time taken to process each event from the moment it enters the system to when it completes processing on the instance. As with throughput, we calculate $\Delta Latency$ as the difference in latency observed under normal conditions versus

© Bashtovyi A. V., Fechan A. V., 2025
DOI 10.15588/1607-3274-2025-3-2

when fault conditions are introduced into the system by the formula(4):

$$\Delta Latency = Latency_{actual} - Latency_{SLO}. \tag{4}$$

This comparison will allow us to quantify the influence of faults on latency. In addition to $\Delta Latency$, we defined $Latency_{SLO}$, which describes a certain latency threshold acceptable by business requirements. Some applications, like the critical financial sector strictly require minimum latency for the operations in a system. This value is quite subjective and depends on the business needs. The best way to establish Latency SLO latency is by defining limitations for a certain business process by stakeholders based on the monitoring and recording of an average Latency for a specific time frame. Total latency is calculated by the formula(5):

$$Latency = \frac{\Delta Latency}{Latency_{SLO}}. \tag{5}$$

Consumer lag refers to the difference between the last message produced to a Kafka topic and the last message consumed by a downstream consumer. It indicates how far behind the consumer is in processing the data, which can occur due to factors such as high data production rates, network bottlenecks, slow processing by the application, failures, network delays, or other reasons. The actual consumer lag $L(t)$ at time t is normalized and defined by formula(6):

$$L_{\text{normalaised}}(t) = \min\left(\frac{L(t)}{L_{\max}}, 1\right). \tag{6}$$

Basically, $L_{normalised}(t) = 0$ when there is no lag and $L_{normalised}(t) = 1$ when the lag reaches or exceeds $L_{\max}$.

The formula is V_{lag} designed to quantify the proportion of time during which a Kafka Streams application experiences consumer lag violations relative to a defined SLO threshold, specifically , which is defined by the stakeholders based on both business expectations and tracking average or common lag in the production system. The actual formula(7):

$$V_{lag} = \frac{1}{T_{total}}\left(\sum_{i=1}^{n} L_{normalized}(t_i) \times \Delta t\right). \tag{7}$$

This metric allows for precise monitoring of application performance by assessing how often and for how long the system fails to meet the lag criteria, which is critical for maintaining real-time processing guarantees. The use of time-weighted integration ensures that the metric reflects the severity and duration of SLO violations, enabling more accurate diagnosis and optimization of stream processing topologies. This makes it an essential tool for evaluating fault tolerance and ensuring that the application meets business-critical requirements. Since the formula uses time-weighted experiments measurements must be conducted at least for two time points with the specified time differences. By normalizing lag values to the range [0, 1] and aggregating them over time, the formula provides a clear, comparable, and actionable measure of performance degradation under varying workloads or fault scenarios. The resulting value 0 means there is no lag on the consumer while 1 means lag is severe respectively to the defined lag threshold L_{max}. Based on the previously mentioned formulas, we defined Business Fault Tolerance Impact (BFTI) metric which is designed to quantify the overall business impact of a fault in Kafka Streams by considering the direct effects on SLOs, recovery time, throughput reduction, and their subsequent impact on operational costs, which is described by the formula(8):

$$BFTI = w_1 \times V_{lag} + w_2 \times \left(\frac{Throughput}{Throughput_{normal}}\right) + w_3 \times (Latency). \tag{8}$$

This metric provides a view of how system performance during failures translates to operational impact on the business. The results of the formula are represented in a value that is in the range of [0,1]. Lower BFTI values indicate high fault tolerance, meaning the application recovers quickly from failures with minimal impact on throughput, latency, or SLO violations. Higher BFTI values suggest poor fault tolerance, indicating significant performance degradation during failures, such as prolonged recovery times, high lag, or unacceptably high latency. In the Table 1 we have defined a reference table to simplify the interpretation of results.

Table 1 – BFTI formula results interpretation

BFTI output	Explanation	Description
(0 – 0.3]	Excellent fault tolerance	The system performs reliably under failure conditions. No immediate action is required.
(0.3–0.6]	Good fault tolerance	Minor impact on performance during failures. Monitor specific bottlenecks (e.g., lag or latency).
(0.6–0.8]	Moderate fault tolerance	Noticeable performance degradation. Review system capacity and failure recovery mechanisms.
(0.8–1.0]	Poor fault tolerance	Critical issues with fault handling. Immediate optimization or adding new instance is required.

Additionally, the formula introduces weighted prioritization based on business-critical metrics. Weights related to 3 main measurements: throughput, latency, and V_{lag}. A larger weight coefficient means a larger impact and priority for the respective measurement. If throughput is more critical than latency and V_{lag} than $w_2 > w_3$ and $w_2 > w_1$.

4 EXPERIMENTS

In this section we describe our technical setup used for experiments, actual experiment methodology and architecture. The technologies were selected based on our experience, usage in production, popularity, and expertise. Moreover, the selected tools mimic real-world production deployments. Our experiment replicated real-world stream processing environments using a Docker-based virtual machine setup managed via Docker Compose (v2.32.4). Containers were configured with 8 GB RAM, 40 GB disk space, and an 8-core CPU. Apache Kafka (vcp-kafka:7.1.0-1-ubi8) served as the message broker, while Kafka Streams (v3.8.1) with Spring Boot (v3.4.1) handled stream processing. Metrics were tracked using JMX, with Prometheus(v2.54.1) for data collection and Grafana for real-time visualization. The experiments ran on two Kafka client instances to evaluate distributed processing performance, ensuring a high-throughput and reproducible benchmarking environment.

For the experiment, we created a sample infrastructure based on the state-of-the-art technologies discussed above. Fig 1. shows the architectural solution. Kafka Streams was chosen as the core client library for our stream processing experiments due to our experience with it, its seamless integration with Kafka-based ecosystems, and its suitability for projects requiring rapid deployment [14]. As a lightweight, client-side library, it simplifies real-time data processing without the need for additional infrastructure, unlike other solutions that may require dedicated servers. Kafka Streams also supports essential features such as fault tolerance, stateful processing, and windowing, making it a practical and efficient choice for high-velocity data streams in agile business environments. We defined producer application and consumer application, message broker, metrics aggregator, and visualization of the metrics. To collect data and state we set JMX exporters that publish metrics from consumer applications and message brokers to monitor the metrics defined above.

For the producer, we created a data-generator app, that operated continuously throughout the experiment, providing a consistent workload that enabled detailed tracking of individual metrics across iterations. The data generator generates synthetic data at a rate of 700 events per second for two specific topics. This rate was chosen to ensure optimal resource utilization and maintain clarity in experimental conditions. In our experiment, these are tutor-in and lesson-in topics, ids for the models are generated iteratively, beginning from 0. We took a synthetic example from the educational domain, tutors can have multiple lessons attached to them.

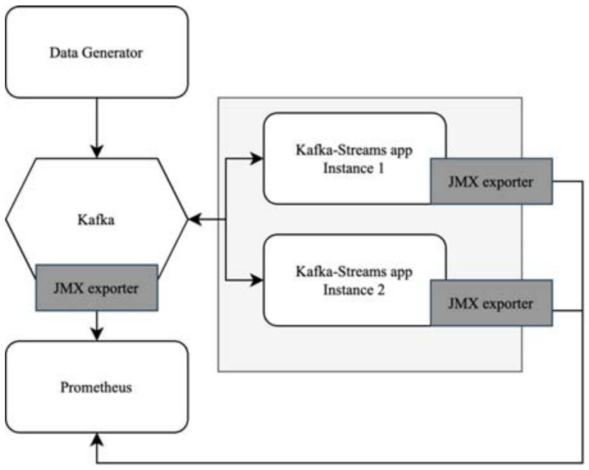

Figure 1 – The architecture of the experimental application

Tutor topic events are not necessarily related to unique tutors, so multiple events can be connected to a single business entity. The app allows a flexible configuration change for the rate of the events and other parameters. Fig. 2 shows the Kafka Stream topology used by consumers in our application. The topology consists of two source processors, multiple intermediary processors, and one sink processor where eventual results are recorded. The application consumes input events from two different topic which are populated by the producer app. There are two live instances of consumer applications. The metrics we measure are calculated based on the values from two instances, hence the moments one instance is not working metrics should reflect that accordingly. Consumer applications expose all existing build-in metrics via JMX to Prometheus.

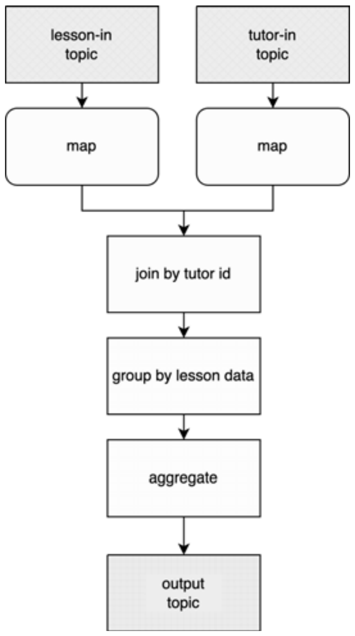

Figure 2 – Synthetic topology architecture

DOI 10.15588/1607-3274-2025-3-2

In order to measure the effect of storage capacity on the processing throughput our topology contains stateful operation – aggregation. Stateful processing helps us understand the relationships between events and leverage these relationships for more advanced stream processing use cases. To support stateful operations, we need a way of storing and retrieving the remembered data, or state, required by each stateful operator in our application (e.g., count, aggregate, join, etc.). The storage abstraction that addresses these needs in Kafka Streams is called a state store, and since a single Kafka Streams application can leverage many stateful operators, a single application may contain several state stores. There are two options for how Kafka Streams can handle stateful operations, in memory and disk storage state. The memory state is efficient in terms of processing and fast operations. On the other hand disk storage has significant benefits in comparison to the in-memory: A state can exceed the size of available memory. In the event of failure, persistent stores can be restored quicker than in-memory stores. Since the application state is persisted to disk, Kafka Streams does not need to replay the entire topic to rebuild the state store whenever the state is lost (e.g., due to system failure, instance migration, etc.). It just needs to replay whatever data is missing between the time the application went down and when it came back up.

For our application, every individual Kafka Streams instance preserves its own disk-based state store. When an individual application fails on the restart it checks out local state stores and restores all of the relevant values. If an application cannot be started on the same instance then a new instance has to be started. In this scenario, the state store will not be replicated on the new instance automatically and the application has to restore a state from scratch. Since every state store is backed by Kafka Topics it is quite a simple task to do. The application has to replay all of the messages available in backed topics and restore the state. In order to speed up instance state store recovery there are multiple techniques that can be used. One of them is to have disk memory saved somewhere in the durable storage so after instance replacement there is a way to simply use the existing disk instead of replaying Kafka messages. Since the focus of the experiment is to provide insights on the efficient and rapid state recovery we based our experiment on the persistent disk state store for the applications. We conducted a detailed series of experiments over a total execution time of more than 24 hours to evaluate system performance and fault tolerance under varying conditions. The experiment consisted of six iterations, with each iteration running for 2 hours to gather metrics systematically. Within each iteration, we assessed fault tolerance by introducing synthetic failures, randomly terminating one instance multiple times during the execution period to simulate real-world disruption scenarios. We determined the instance exactly after 30 minutes of beginning an interaction. To ensure robust and unbiased results, we repeated the experiments under three different disk load conditions, with two iterations per load state. Eventually, we took average results per two interactions. For the first set of iterations, the disk state was empty, representing 0% capacity utilization. The subsequent set of iterations simulated a slightly(0%), partially(about 50%), and heavily(more than 80%) loaded state. This approach allowed us to observe the system's behavior across varying levels of resource utilization, ensuring the reliability and accuracy of the evaluation outcomes. In order to load a disk for our experiment we decided to generate data synthetically, thereby populating individual instance stores with relevant events which are aggregated on the instances. This mimics the real-world scenarios when an application works for days and stores a lot of data in state storage. To monitor disk load during the experiments, we utilized the Prometheus metrics disk_free_bytes and disk_total_bytes. These metrics allowed us to calculate the percentage of disk capacity utilized at any given time, providing a clear representation of the disk load for each test scenario. This approach ensured precise tracking of disk usage, which was critical for evaluating the impact of varying disk load states on system performance and fault tolerance.

Throughput generally refers to the rate at which a system processes data. In Kafka consumers, we measured it as the number of records consumed per second from our source topics. Meaning that if records were consumed successfully then the records will be processed by the next nodes in the topology. For the throughput, we decided to measure throughput for individual instances and sum this value up. We selected kafka_consumer_records_consumed_total, which is a counter metric that increments whenever a Kafka consumer successfully processes a record. It directly reflects the number of records consumed over time, making it a reliable proxy for measuring throughput. Since the metric is cumulative we decided to take a rate of this metric and measure the number of records consumed per 1 minute. End-to-end (e2e) latency measures the total time taken for a record to traverse from the source topic to the sink topic within a Kafka Streams topology. This metric captures the processing delays introduced at each node in the topology, including computation, state store interactions, and any intermediate transformations. Measuring e2e latency helps us understand the overall performance of the data pipeline and identify potential bottlenecks. For this measurement, we chose the kafka_stream_processor_node_record_e2e_latency_avg metric, which represents the average latency for records processed by each processor node in the topol-

DOI 10.15588/1607-3274-2025-3-2

ogy. This metric provides granularity at the processor node level, allowing us to analyze and aggregate the latency for the entire topology. To calculate the e2e latency for individual instances and across the application, we averaged the metric across all processor nodes and instances. Since latency is not cumulative, we used an average calculation instead of a rate: avg(kafka_stream_processor_node_record_e2e_latency _avg). This setup allows us to evaluate the latency at each processor node. To calculate the overall e2e latency for the entire topology, we aggregated the metric across all nodes and instances: This approach ensures that we capture the average e2e latency for processing records, giving us insights into the system's overall responsiveness and helping to pinpoint areas for optimization.

The V_{lag} metric uses the kafka_consumer_fetch_manager_records_lag_avg metric, which represents the average consumer lag in Kafka consumers. This metric is aggregated and normalized over time to reflect the proportion of lag violations across the system. The base metric kafka_consumer_fetch_manager_records_lag_avg is collected for each consumer group and topic. We applied the metric for our two input topics. Since the metric is tracked across two running instances it is aggregated by topic only and normalized for value to be in [0,1] range.

Implementing the BFTI formula within Prometheus involved monitoring and calculating essential metrics directly from the Kafka Streams application. It is designed to evaluate the system's fault tolerance based on three critical components: SLO-based lag, throughput degradation, and latency increase. These components are derived from the metrics above and provide a quantitative measure of the system's performance under fault conditions. Each component incorporates arguments that must be defined collaboratively by stakeholders and the engineering team. These arguments are based on system behaviour during normal operation and the tolerable thresholds established for each metric. For instance, tolerable limits for latency or throughput degradation may reflect SLO agreements or operational baselines. The table 2, shows values we have set for these arguments, ensuring alignment with real-world operational expectations and providing a framework for accurately assessing the system's resilience. This structured approach allows for reproducible evaluation and fosters a deeper understanding of the system's fault tolerance characteristics.

Table 2 – Input arguments for the experiment

Argument	Value	Description
w_1, w_2, w_3	1	Determine the relative importance of each factor lag violation, throughput degradation, and latency increase.
L_{max}	1000 messages	Lag SLO Threshold. The maximum acceptable consumer lag beyond which SLO is considered violated.
$Latency_{SLO}$	35sec	The end-to-end latency threshold that defines acceptable performance.
$Throughput_{normal}$	500 messages/sec	The normal or expected throughput of the Kafka Streams application under fault-free conditions

5 RESULTS

This section presents the findings from our experiments. Our analysis showed a positive correlation between system failures and changes in key performance metrics, including those measured by the SLO metric. In the first iteration, no synthetic data was preloaded onto the hard drive, resulting in a 0% load from experimental data. However, due to Kafka's internal metadata storage, the actual disk utilization was approximately 24%.

Metrics gathered under these baseline conditions served as benchmarks for defining SLO constraints during subsequent experiments. As illustrated on Fig. 3a, e2e latency significantly increased during an instance shutdown, doubling its normal value during the restoration period of the second instance. This spike is consistent with expectations, as live instances require additional time to rebalance and synchronize with the data generation rate during failures. Once the disrupted instance was restored, latency returned to baseline levels, reflecting the system's recovery capabilities under fault conditions. Initially, the system maintained a stable load, with throughput exhibiting relatively consistent, non-volatile values. Additionally, we can see(Fig. 3b), throughput was not significantly impacted by instance failures. In fact, brief increases in throughput were observed, indicating that the active instances temporarily accelerated processing to catch up on event backlogs. Figure 4 illustrates the effect of instance failures on SLO-based lag. The experiment reveals a dramatic spike in consumer lag, with values increasing from a regular average of 112 messages to over 15,000 messages in a short period. This behaviour aligns with expectations, as the number of incoming events remained constant while the number of active instances available to process these events was temporarily reduced. Consequently, the accumulation of unprocessed messages caused the lag to escalate rapidly during the failure period.

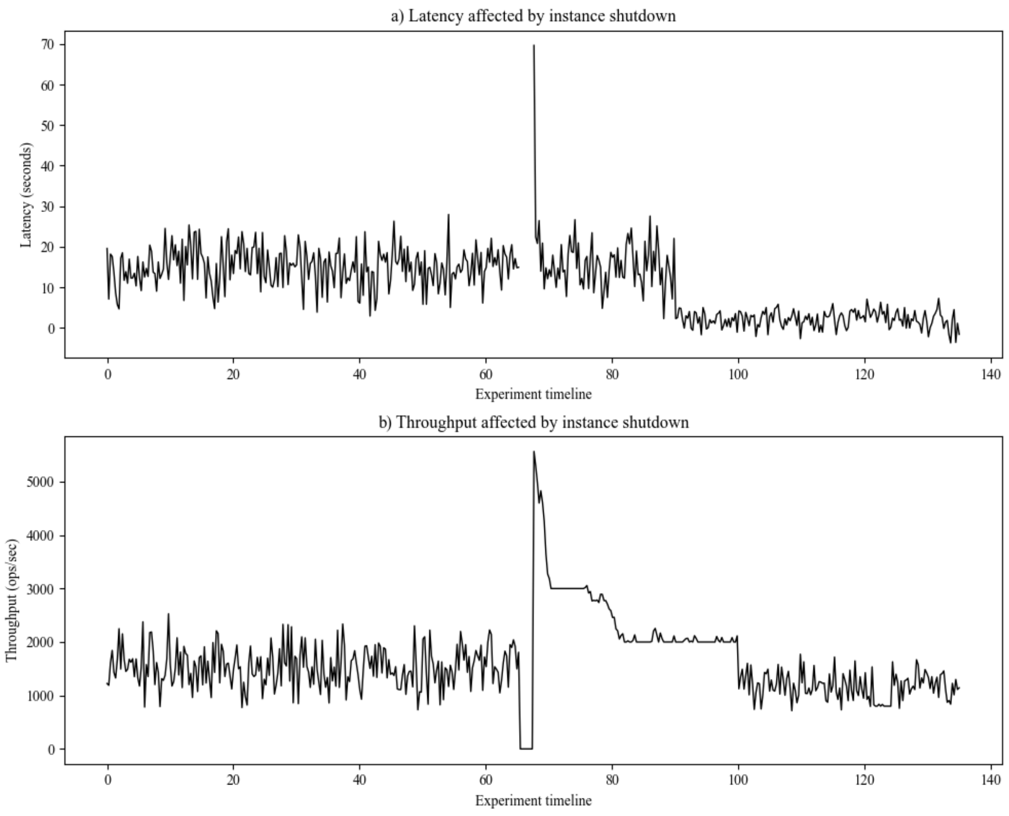

Figure 3 – Latency and throughput affected by instance shutdown at 66 minute of experiment

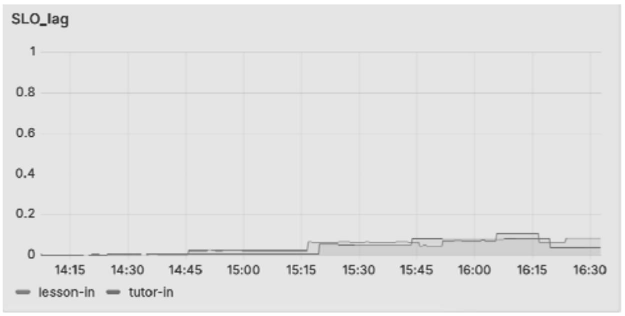

Figure 4 – Failures effect on V_{lag} latency

In the next phase of the experiment, we introduced varying levels of disk load and conducted additional iterations using the same methodology described earlier. The results, summarized in Table 3, reveal the impact of disk load on throughput, latency, and overall application performance. Notably, the system maintained stable performance when disk utilization remained below 80%, with state restoration occurring relatively quickly and without significant degradation in the basic operational characteristics of the instances. However, when disk utilization exceeded 80%, the system exhibited marked performance degradation. Latency following a fault increased nearly fivefold compared to regular latency under minimal disk load conditions. Additionally, recovery time was prolonged, taking approximately 2.7 times longer than under a 0% disk load. Throughput also declined significantly, dropping to less than half of the normal rate observed under lighter disk load conditions.

Table 3 – Experiment results for the defined metrics

Metric	Disk load		
	0%	<50%	>80%
Latency, sec	87	121	445
Throughput, ops/sec	912	819	331
V_{lag}	0.21	0.34	0.54
BFTI	0.31	0.41	1

6 DISCUSSION

The findings in the section above highlight the critical role of disk utilization in maintaining the performance and fault recovery capabilities of the system. While author[15] extensively evaluated stream processing frameworks under failure scenarios, their benchmark primarily focused on stateless or short-lived workloads with empty state stores. In contrast, our approach simulates production-grade conditions with preloaded persistent states and long-running streams, uncovering fault recovery delays that were not visible in prior benchmarks. This highlights both a methodological extension and deeper insight into the behavior of stateful applications under disk-intensive loads. Additionally, no authors provide a standardized methods for stream processing app tracking. Based on the experiment results represented in Table 3 BFTI formula provides stakeholders with a comprehensive measure of the overall fault tolerance and performance stability based

DOI 10.15588/1607-3274-2025-3-2

on the single output. We can see the BFTI definition correlates with increased latency, lower throughput, and increased V_{lag}. The formula simply represents an aggregation of all of the results in a simple manner. It simplifies decisions and saves time for interested people mainly because they can take a look at individual value instead of monitoring the whole system for different metrics. If the following and deeper analysis is required then engineering teams can elaborate on discovery and investigate various metrics of the system. However, it is the next step after monitoring BFTI results. The impact of a failure on one instance is straightforward: the application processes fewer events at a slower pace. But the long-term effects of such failures are more critical. Our analysis shows that latency and throughput remain degraded for a considerable period even after the affected instance has restarted. This occurs because the data generator continues to produce events during the failure, leading to a backlog of unprocessed events. As a result, some events remain delayed while the failed instance recovers and the system rebalances. For stakeholders, this means the system handles business tasks at a reduced efficiency, potentially affecting operational timelines and customer satisfaction. The metrics reveal that it took over an hour for the application to recover to its initial performance levels, indicating that fault recovery is not instantaneous and can disrupt normal operations for an extended period. This recovery lag underscores the importance of considering fault-tolerance strategies to minimize downtime and ensure smoother operations. One potential mitigation strategy is scaling the application dynamically in response to failures. For example, when an instance fails, a new instance could be launched alongside restarting the failed one. However, this approach has limitations it requires the number of Kafka partitions to be equal to or greater than the total number of instances, and scaling down later can introduce additional overhead due to rebalancing. Moreover, this solution may be resource-intensive and could temporarily impact throughput and overall performance. These findings highlight the need for careful planning of fault-tolerance mechanisms that balance recovery speed, resource allocation, and system performance, ensuring minimal disruption to both operations and stakeholder expectations.

CONCLUSIONS

The study addressed critical gaps in benchmarking methodologies for stream processing frameworks by simulating production-like environments and introducing SLO-based metrics to evaluate fault-tolerance performance. Key findings demonstrate that while systems maintain stable performance under moderate disk loads, performance degrades significantly when disk utilization exceeds 80%. The increased latency, throughput reduction, and prolonged recovery times observed under heavy disk loads underline the importance of robust fault-tolerance mechanisms. Furthermore, the incorporation of SLO-based metrics provided meaningful insights into how technical disruptions affect business outcomes, emphasizing the value of bridging the gap between engineering metrics and stakeholder objectives.

The scientific novelty of the obtained results lies in the proposed methodology for evaluating fault recovery in stream processing applications using preloaded state stores and business-driven performance indicators for the first time. Unlike existing approaches, this method integrates SLO-based metrics to quantify the business impact of failures, providing a novel perspective on fault tolerance assessment. Additionally, our work introduces a new benchmarking framework that considers varying state loads, which enables a more realistic evaluation of stream processing resilience in production environments.

The practical significance of the results is that the developed benchmarking methodology and BFTI metric allow practitioners to assess the reliability of stream processing applications with greater precision. The methodology was validated through experiments, demonstrating its applicability for real-world deployments. The proposed approach can be directly integrated into performance monitoring systems, aiding decision-makers in optimizing resource allocation, failure recovery strategies, and system resilience.

Prospects for further research include refining the proposed benchmarking methodology to accommodate different types of state store implementations, such as distributed file systems or cloud-based storage solutions. Future work should also explore the development of adaptive SLO-based metrics that dynamically adjust based on workload variations and user-defined business priorities. Additionally, extending the study to compare fault recovery across multiple stream processing frameworks, such as Apache Flink and Spark Streaming, would provide deeper insights into optimizing real-time data processing for various industry applications.

REFERENCES

1. Fragkoulis M., Carbone P., Kalavri V. et al. A survey on the evolution of stream processing systems, *The VLDB Journal*, 2024, Vol. 33, № 2, pp. 507–541. DOI: 10.1007/s00778-023-00819-8
2. Sasaki Y. A survey on IoT big data analytic systems: Current and future, *IEEE Internet of Things Journal*, 2022, Vol. 9, № 2, pp. 1024–1036. DOI: 10.1109/JIOT.2021.3131724
3. Bashtovyi A., Fechan A. Change data capture for migration to event-driven microservices: Case study, *Proc. of the IEEE Int. Conf. on Computer Science and Information Technologies (CSIT),* 2023, pp. 1–4. DOI: 10.1109/CSIT61576.2023.10324262
4. Vogel A., Henning S., Perez-Wohlfeil E. et al. A comprehensive benchmarking analysis of fault recovery in stream processing frameworks, *Proc. of the 18th ACM Int. Conf. on Distributed and Event-Based Systems,* 2024, pp. 171–182. DOI: 10.48550/arXiv.2404.06203
5. Marcotte P., Grégoire F., Petrillo F. Multiple fault-tolerance mechanisms in cloud systems: A systematic review, *2019 IEEE Int. Conf. on Software Quality, Reliability and Security Companion (QRS-C),* 2019, pp. 337–344. DOI: 10.1109/ISSREW.2019.00104

6. Friedman E., Tzoumas K. Introduction to Apache Flink: Stream Processing for Real Time and Beyond. Sebastopol, O'Reilly Media, 2016, 322 p.
7. Wu H., Shang Z., Peng G., Wolter K. A reactive batching strategy of Apache Kafka for reliable stream processing in real-time, *2020 IEEE 31st Int. Symp. on Software Reliability Engineering (ISSRE)*, 2020, pp. 252–261. DOI: 10.1109/ISSRE5003.2020.00028
8. Van Dongen G., Van den Poel D. Evaluation of stream processing frameworks for fault tolerance and performance metrics, *IEEE Access*, 2021, Vol. 9, pp. 102349–102365. DOI: 10.1109/TPDS.2020.2978480
9. Venkataraman S., Yang Z., Parashar M. et al. Cost of fault-tolerance on data stream processing, *Proc. of the VLDB Endowment,* 2017, Vol. 10, № 11, pp. 1478–1491. DOI: 10.1007/978-3-030-10549-5_2
10. Grambow M. Benchmarking Microservice Platforms and Applications in the Cloud. Berlin, TU Berlin, 2024. [in press].
11. Henning S., Hasselbring W. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud, *Journal of Systems and Software,* 2024, Vol. 208, pp. 111879. – DOI: 10.1016/j.jss.2023.111879
12. Wang X., Zhang C., Fang J. et al. A comprehensive study on fault tolerance in stream processing systems, *Frontiers of Computer Science*, 2022, Vol. 16, P. 162603. DOI: 10.1007/s11704-020-0248-x
13. Hoseiny Farahabady M. R., Taheri J., Zomaya A. Y. et al. A dynamic resource controller for resolving quality of service issues in modern streaming processing engines, *2020 IEEE 19th Int. Symp. on Network Computing and Applications (NCA)*, 2020, pp. 1–8. DOI: 10.1109/NCA51143.2020.9306697
14. Van Dongen G., Van den Poel D. A performance analysis of fault recovery in stream processing frameworks, *IEEE Access,* 2021, Vol. 9, pp. 93745–93763. DOI: 10.1109/ACCESS.2021.3093208
15. Van Dongen G. Open stream processing benchmark: an extensive analysis of distributed stream processing frameworks : Master's thesis. Ghent, Ghent University, Faculty of Economics and Business Administration, 2021, 112 p.

Accepted 18.03.2025.
Received 11.06.2025.

УДК 004. 42

ОЦІНКА ВІДНОВЛЕННЯ РОЗПОДІЛЕНИХ СИСТЕМ ПІСЛЯ ЗБОЇВ У ДОДАТКАХ ПОТОКОВОЇ ОБРОБКИ ДАНИХ: РОЗУМІННЯ МЕТРИК З ТОЧКИ ЗОРУ БІЗНЕСУ

Баштовий А. В. – аспірант кафедри програмного забезпечення національного університету "Львівська політехніка", Львів, Україна.

Фечан А. В. – д-р техн. наук, професор кафедри програмного забезпечення, Національний університет "Львівська політехніка", Львів, Україна.

АНОТАЦІЯ

Актуальність. Фреймворки потокової обробки даних широко використовуються в галузях фінансів, електронної комерції та IoT для ефективної обробки потоків даних у реальному часі. Проте більшість методологій тестування не відтворюють умови реальної роботи після впровадження, що призводить до неповної оцінки продуктивності відновлення після збоїв. Об'єктом дослідження є оцінка фреймворків потокової обробки у реалістичних умовах з урахуванням попередньо завантажених сховищ даних та бізнес-орієнтованих метрик.

Мета роботи. Розробка нової методології оцінювання продуктивності відновлення після збоїв у фреймворках потокової обробки, яка імітує виробничі умови з різними рівнями завантаження диска та вводить SLO-орієнтовані метрики для оцінки.

Метод. Методологія передбачає серію експериментів із використанням Kafka Streams у віртуалізованому середовищі на базі Docker. Експерименти оцінюють продуктивність системи при трьох рівнях завантаження диска: 0%, 50% та 80%. Під час роботи вводяться синтетичні збої, а ключові метрики, такі як пропускна здатність, затримка та відставання споживачів, відстежуються за допомогою JMX, Prometheus та Grafana. Запропонована метрика Впливу Бізнесу на Толерантність до Збоїв (BFTI) агрегує технічні показники у спрощене значення, що відображає бізнес-ефекти відновлення після збоїв.

Результати. Експерименти показують, що рівень завантаження диска суттєво впливає на продуктивність відновлення. При завантаженні диска понад 80% час відновлення збільшується у 2,7 рази, а затримка зростає до п'яти разів у порівнянні з 0% завантаження. Введення SLO-орієнтованих метрик підкреслює зв'язок між продуктивністю системи та бізнес-результатами, надаючи зацікавленим сторонам більш інтуїтивну оцінку стійкості програми.

Висновки. Отримані результати підкреслюють важливість моделювання реальних виробничих умов у тестуванні фреймворків потокової обробки. Метрика BFTI пропонує новий підхід до перетворення технічних показників у бізнес-орієнтовані індикатори. Подальші дослідження повинні включати адаптивні SLO-метрики, порівняння фреймворків та дослідження продуктивності на довготривалих інтервалах для подальшого усунення розриву між технічними показниками та бізнес-потребами.

КЛЮЧОВІ СЛОВА: потокова обробка даних, відмовостійкість, Kafka Streams, зняття метрик, розподілені системи, цілі рівня обслуговування (SLO), вимірювання продуктивності.

ЛІТЕРАТУРА

1. A survey on the evolution of stream processing systems / [M. Fragkoulis, P. Carbone, V. Kalavri et al.] // The VLDB Journal. – 2024. – Vol. 33, № 2. – P. 507–541. DOI: 10.1007/s00778-023-00819-8
2. Sasaki Y. A survey on IoT big data analytic systems: Current and future / Y. Sasaki // IEEE Internet of Things Journal. – 2022. – Vol. 9, № 2. – P. 1024–1036. DOI: 10.1109/JIOT.2021.3131724
3. Bashtovyi A. Change data capture for migration to event-driven microservices: Case study / A. Bashtovyi, A. Fechan // Proc. of the IEEE Int. Conf. on Computer Science and Information Technologies (CSIT). – 2023. – P. 1–4. DOI: 10.1109/CSIT61576.2023.10324262
4. A comprehensive benchmarking analysis of fault recovery in stream processing frameworks / [A. Vogel, S. Henning, E. Perez-Wohlfeil et al.] // Proc. of the 18th ACM Int. Conf. on Distributed and Event-Based Systems. – 2024. – P. 171–182. DOI: 10.48550/arXiv.2404.06203
5. Marcotte P. Multiple fault-tolerance mechanisms in cloud systems: A systematic review / P. Marcotte, F. Grégoire, F. Petrillo // 2019 IEEE Int. Conf. on Software Quality, Reliability and Security Companion (QRS-C). – 2019. – P. 337–344. DOI: 10.1109/ISSREW.2019.00104
6. Friedman E. Introduction to Apache Flink: Stream Processing for Real Time and Beyond / E. Friedman, K. Tzoumas. – Sebastopol : O'Reilly Media, 2016. – 322 p.
7. Wu H. A reactive batching strategy of Apache Kafka for reliable stream processing in real-time / H. Wu, Z. Shang, G. Peng, K. Wolter // 2020 IEEE 31st Int. Symp. on Software Reliability Engineering (ISSRE). – 2020. – P. 252–261. – DOI: 10.1109/ISSRE5003.2020.00028
8. Van Dongen G. Evaluation of stream processing frameworks for fault tolerance and performance metrics / G. Van Dongen, D. Van den Poel // IEEE Access. – 2021. – Vol. 9. – P. 102349–102365. DOI: 10.1109/TPDS.2020.2978480
9. Venkataraman S. Cost of fault-tolerance on data stream processing / [S. Venkataraman, Z. Yang, M. Parashar et al.] // Proc. of the VLDB Endowment. – 2017. – Vol. 10, № 11. – P. 1478–1491. DOI: 10.1007/978-3-030-10549-5_2
10. Grambow M. Benchmarking Microservice Platforms and Applications in the Cloud / M. Grambow. – Berlin : TU Berlin, 2024. – [in press].
11. Henning S. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud / S. Henning, W. Hasselbring // Journal of Systems and Software. – 2024. – Vol. 208. – P. 111879. DOI: 10.1016/j.jss.2023.111879
12. A comprehensive study on fault tolerance in stream processing systems / [X. Wang, C. Zhang, J. Fang et al.] // Frontiers of Computer Science. – 2022. – Vol. 16. – P. 162603. DOI: 10.1007/s11704-020-0248-x
13. A dynamic resource controller for resolving quality of service issues in modern streaming processing engines / [M. R. Hoseiny Farahabady, J. Taheri, A. Y. Zomaya et al.] // 2020 IEEE 19th Int. Symp. on Network Computing and Applications (NCA). – 2020. – P. 1–8. DOI: 10.1109/NCA51143.2020.9306697
14. Van Dongen G. A performance analysis of fault recovery in stream processing frameworks / G. Van Dongen, D. Van den Poel // IEEE Access. – 2021. – Vol. 9. – P. 93745–93763. DOI: 10.1109/ACCESS.2021.3093208
15. Van Dongen G. Open stream processing benchmark: an extensive analysis of distributed stream processing frameworks : Master's thesis / G. Van Dongen. – Ghent : Ghent University, Faculty of Economics and Business Administration, 2021. – 112 p.

DOI 10.15588/1607-3274-2025-3-2