SvedbergOpen
DISSEMINATIONOFKNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Resilient Event-Driven Architecture for Real-Time Digital Payment Processing Integrating Kafka-Based Microservices, Idempotent Saga Transactions, and Observability for Fault-Tolerant Bank Transfers

Sai Prasad Saladi[1], Sai Charan Obuladinne[2], Nikhil Singh[3]

[1] Senior Software Engineer, U.S. Bank, Gainesville, FL, USA. Email: saladi.dev@gmail.com
[2] Senior Software Engineer, Meta, Mountain View, CA, USA. Email: obuladinnesai@gmail.com
[3] Lead Automation Engineer, U.S. Bank, Atlanta, GA, USA. Email: Singh.nikhil254@gmail.com

Abstract
Real-time payment workflows must preserve transaction correctness while recovering from distributed failures. This study compared the Kafka-based Idempotent and Observable Saga Framework for Resilient Real-Time Payments (KIOS-RTP) with baseline asynchronous Kafka microservices across 120,000 matched executions. Each architecture processed 60,000 executions spanning normal-load, stress-load, and fault/recovery conditions. The evaluation covered wire and external transfers, eight fault classes, and incident detection using 160 injected episodes and 80 non-fault windows per configuration. Correct terminal states increased from 98.35% to 99.69%, duplicate processing decreased from 0.75% to 0.11%, and recovery success increased from 85.90% to 96.20%. Mean recovery time decreased from 298.5 to 171.6 s. Median latency increased from 142 to 151 ms, while throughput decreased from 2,310 to 2,245 transactions/s. Detection and diagnosis times decreased from 41.6 to 15.4 s and from 173.2 to 62.8 s. Descriptive ablations associated the control layers with different correctness and recovery outcomes. The contribution is an integrated assessment of reliability, performance, and observability. Incomplete implementation and statistical specifications, and an unresolved workload aggregation discrepancy, limit reproducibility and deployment conclusions.

Keywords: event-driven architecture; Kafka; microservices; Saga; idempotency; payment processing; observability.

I. INTRODUCTION

Asynchronous payment workflows distribute processing across operating services. A payment can pass through validation, persistence, event publication, transfer processing, settlement, and reconciliation before reaching its final state. The research problem concerns what happens when this sequence is interrupted: whether the transaction completes correctly, is compensated, or remains unresolved. Microservice data-management research identifies consistency across distributed service states as a central architectural difficulty [1]. Reviews of communication patterns and quality attributes similarly establish that architecture selection involves several interacting concerns rather than performance alone [2], [3].

The study evaluates KIOS-RTP as an integrated payment-processing architecture rather than a new messaging protocol. Its contribution is the joint assessment of event reliability, idempotency, Saga consistency, reconciliation, and transaction-correlated observability against payment correctness, processing cost, and fault recovery. The comparison includes matched executions, workload and transfer-type analyses, component ablations, and eight controlled fault classes.

Three integration questions motivate the evaluation. First, how does the complete control set affect duplicate processing and terminal-state consistency within a bank-transfer workflow? Second, what latency and throughput costs accompany the observed reliability changes? Third, how does transaction-correlated observability relate to incident detection, diagnosis, and recovery? These questions concern established mechanisms available in 2021–2022 [4], [5]; the contribution is their combined evaluation in the specified payment workflow.

The objectives and hypotheses define three linked comparisons. O1 and H1 concern integrated transaction correctness. O2 and O5, together with H3, concern the reliability–performance trade-off. O3 and H4 concern correlated incident detection and diagnosis. O4, H2, and H5 concern recovery under controlled faults. The primary comparison, incident analysis, and descriptive ablations provide complementary

evidence for these questions. Scholarly coverage is restricted to publications through 2022; that boundary does not establish experiment dates.

II. Related Work and Research Position

A. Distributed transactions and service consistency

Laigner et al. examine data-management practices and challenges in microservices [1], while the communication-pattern review [2] and quality-attribute review [3] provide architectural context. Daraghmi et al. investigate Saga-based transaction consistency [5]. Djerou and Tibermacine examine formal Saga verification [6], and Žežulka et al. integrate tracing with a transaction manager [7]. These studies address related coordination and visibility problems; they do not establish the payment outcomes measured here.

KIOS-RTP combines local transactions, Saga transitions, compensation, and terminal-state validation with duplicate detection and a processed-event registry. Reconciliation compares payment events and states and supports incomplete-transaction reprocessing. The evaluation examines the combined outcome of these mechanisms. It does not introduce a new Saga protocol, establish a new Kafka delivery guarantee, or compare every alternative distributed-transaction architecture.

B. Observability and incident reconstruction

Bento et al. discuss automated distributed-tracing analysis [8]; Usman et al. survey microservice observability [9]; and Li et al. examine industrial tracing practices [10]. Offline trace generation [11], fault localization [12], serverless troubleshooting [13], distributed/kernel tracing [14], bottleneck identification [15], and locally collected tracing data [16] address complementary aspects of execution visibility. KIOS-RTP connects transaction-correlated visibility with payment correctness and incident-management outcomes within the same reported evaluation.

The framework correlates Kafka, application, infrastructure, retry, Saga, and reconciliation information through a common transaction identifier. Its observability assessment includes detection accuracy and incident-management time. This choice links telemetry to operational outcomes rather than treating the presence of dashboards as evidence of resilience. Cyberattack detection through distributed tracing represents a related use of execution context [17]; however, the present experiment reports distributed failures and does not evaluate attack detection.

C. Fault tolerance and controlled evaluation

Service-level fault injection and observability-oriented chaos engineering provide methodological context for controlled resilience assessment [18], [19]. Architectural mapping and security reviews also show why resilience evidence should remain distinct from claims about broader architectural quality or security assurance [20]–[23]. KIOS-RTP therefore evaluates the declared failure scenarios without extending the findings to untested threats or deployment environments.

Research on application-level fault injection, reactive microservices, and fault taxonomies supports systematic examination of failure behavior [24]–[26]. Stream-processing studies address partial snapshots, checkpoint optimization, failure-aware recovery, fault-recovery performance, and single-node fault tolerance [27]–[32]. Work on clustered edge processing, database performance, and testing streaming programs broadens the evaluation context [33]–[35]. These references motivate measuring recovery alongside normal operation. Their algorithms, datasets, and numerical findings are not incorporated into the KIOS-RTP experiment. Complementary infrastructure research examines lightweight Linux systems for container workloads [36], cloud resource allocation and auto-scaling complexity [37], and SRv6-based latency and throughput optimization for private-cloud network functions [38]. These studies provide infrastructure context; they do not validate payment correctness or the KIOS-RTP results.

III. Framework and Methodology

A. Study setting and experimental environment

The study compared baseline Kafka microservices with KIOS-RTP through transaction-event replay and controlled fault injection for wire and external transfers. The unit reported for payment outcomes was a transaction execution. Incident detection and throughput used separate measurement units. Experimental provenance and deployment details remain unverified, as specified in Section V.C; the literature cutoff of 2022 is not treated as evidence of experiment dates.

B. KIOS-RTP implementation

The specified payment path comprises request validation, transaction/event persistence, Kafka event exchange, transfer processing, Saga management, settlement/completion events, reconciliation, and final payment state. The five control layers are summarized in Table I. Event reliability covers durable topics, partition keys, ordering, acknowledgements, retries, and replay. Idempotency combines transaction/event identifiers, duplicate detection, and a processed-event registry. Saga consistency combines local transactions, state transitions, compensation, and validation. Reconciliation compares events with payment state and supports incomplete-transaction reprocessing. Observability correlates events, logs, metrics, retries, Saga state, and reconciliation through transaction identifiers.

TABLE I. KIOS-RTP CONTROL LAYERS

Layer	Specified mechanisms
Event reliability	Durable topics, partition keys, ordering, acknowledgements, retries, replay
Idempotent processing	Unique identifiers, duplicate detection, processed-event registry
Saga consistency	Local transactions, state transitions, compensation, validation
Reconciliation and recovery	Incomplete-payment detection, state comparison, reprocessing
Observability	Transaction correlation across events, logs, metrics and recovery

C. Baseline and controlled comparison

The baseline used standard asynchronous Kafka event exchange. KIOS-RTP added the integrated event-reliability, idempotency, Saga, reconciliation, recovery, and correlated-observability controls. The configurations received matched workloads and predefined fault scenarios. The evaluation therefore concerns the complete control set relative to this comparator. It does not isolate each mechanism in the primary comparison; component removal is examined separately in the ablations. Baseline safeguards and experimental-equivalence limitations are addressed in Section V.C.

D. Workload provenance and sample construction

The primary evaluation comprised 120,000 matched executions, with 60,000 per architecture. Within each architecture, 36,000 executions represented normal load, 12,000 stress load, and 12,000 fault/recovery conditions. Wire and external transfers contributed 36,000 and 24,000 executions, respectively. The ablation comparison used 10,000 matched trace executions for each configuration. Those executions are not added to the primary total because their overlap with the primary sample is unspecified.

Three measurement levels are distinguished: payment executions, throughput runs, and incidents/non-fault windows. Throughput summaries cover 30 runs per architecture; detection and diagnosis summaries cover 160 incidents. Low, medium, and high workload strata each contain 20,000 executions per architecture. Their cross-classification with normal, stress, and fault conditions is unavailable, so these allocations are not treated as interchangeable. Fault-class and detection-window composition is specified in Section III.F.

E. Outcome definitions and correctness verification

Table II links the measured outcome families to their analytical units. Latency is reported in milliseconds, throughput in transactions per second, and incident-management time in seconds. Correctness, duplicate execution, inconsistency, event loss, compensation, and reconciliation describe separate outcomes. They are not treated as mutually exclusive error categories because the record does not establish that relationship.

TABLE II. OUTCOME FAMILIES AND ANALYTICAL UNITS

Outcome family	Measures	Unit
Performance	Latency, p95/p99, lag, throughput	Execution; throughput run
Correctness	Terminal state, duplicates, inconsistency, event loss	Transaction execution
Recovery	Recovery, compensation, reconciliation, unresolved cases	Affected execution
Observability	Confusion matrix, detection, diagnosis, traceability	Incident or window

Reported correctness and error rates use the primary execution denominator of 60,000 per architecture. Fault-recovery and unreconciled-case rates use 12,000 affected executions per architecture. Incident-

classification measures use 160 fault episodes and 80 non-fault windows per configuration. Throughput summaries cover 30 runs, while detection and diagnosis summaries cover 160 incidents. These denominators establish the scope of the reported comparisons, not statistical independence. Compensation and reconciliation success retain their reported percentages because eligible denominators are unavailable. Duplicate delivery, repeated handler execution, and duplicate financial effect remain distinct concepts; the available outcome specification does not establish their equivalence.

H1 posits lower duplicate-processing and inconsistent-terminal-state rates under KIOS-RTP. H2 posits higher successful recovery under injected failures. H3 posits improved correctness with operationally acceptable latency and throughput. H4 posits shorter detection and diagnosis times under transaction-correlated observability. H5 posits fewer unreconciled and persistently inconsistent fault outcomes and shorter recovery times. The analyses report outcome-specific subtests. H3 is assessed in separate correctness and performance components; its operational-acceptability clause remains unevaluated because no verified acceptance threshold is documented.

F. Fault-injection and observability protocol

Fault validation covered microservice termination, consumer termination/restart, processing timeout, database unavailability, consumer delay, duplicate-event delivery, retry storms, and delayed/out-of-order events. Each class comprised 1,500 affected executions per architecture, totaling 12,000. Incident detection used 160 injected episodes and 80 non-fault windows per configuration. Incident count and affected-execution count describe different units. Their mapping is unavailable, and equal transaction counts do not establish equal incident allocation. The reported fault classes delimit the resilience assessment; unresolved injection and recovery procedures are described in Section V.C.

G. Statistical analysis and validation

The reported analyses used logistic regression for binary endpoints, paired t-tests for throughput and incident-management times, and non-parametric comparisons for latency and recovery measurements. Reported measures include coefficients, standard errors, odds ratios, relative risks, standardized paired differences, rank-based effects, confidence intervals, and absolute percentage-point changes. The exact primary latency test remains unresolved between Mann–Whitney and Wilcoxon, and the normalization behind r=0.16 is unspecified. These outputs are presented as reported estimates rather than independently reproduced analyses.

Validation covered normal operation, stress, fault injection, and recovery/replay. The reported fault-stratified analyses used Cochran–Mantel–Haenszel and Breslow–Day procedures. Model summaries included calibration, discrimination, and 2,000 bootstrap resamples. Principal inferential conclusions were reported as remaining significant after Holm correction. The fitted model specification, complete correction families, adjusted values, confidence-interval construction, bootstrap sampling unit, and treatment of dependence are not documented. Section V.C describes the resulting limits on inference.

IV. Results and Findings

A. Evaluation sample and descriptive profile

The primary sample was balanced between architectures and included both transfer types. Table III reports its composition and continuous-outcome summaries. Latency distributions were right-skewed. Mean consumer lag decreased from 26.8 to 24.1 events and mean retries from 0.182 to 0.164 per execution. These operational summaries accompany the correctness, performance, and incident results below; the incident and ablation populations are reported separately from the primary execution total.

TABLE III. Evaluation composition and descriptive summaries

Measure	Baseline	KIOS-RTP
Executions	60,000	60,000
Wire / external	36,000 / 24,000	36,000 / 24,000
Normal / stress / fault	36,000 / 12,000 / 12,000	36,000 / 12,000 / 12,000
Fault classes / executions per class	8 / 1,500	8 / 1,500
Fault / non-fault windows	160 / 80	160 / 80
Latency mean ± SD, ms	158.6 ± 96.4	168.9 ± 101.7
Consumer lag mean ± SD	26.8 ± 41.5	24.1 ± 37.2
Retries mean ± SD	0.182 ± 0.61	0.164 ± 0.55
Recovery mean ± SD, s	298.5 ± 184.7	171.6 ± 108.2

Measure	Baseline	KIOS-RTP
Detection mean ± SD, s	41.6 ± 18.2	15.4 ± 7.1
Diagnosis mean ± SD, s	173.2 ± 78.6	62.8 ± 31.4

B. Primary correctness and performance

KIOS-RTP achieved higher transaction correctness than the baseline (Table IV). Correct terminal states increased from 59,010/60,000 (98.35%) to 59,814/60,000 (99.69%), an improvement of 1.34 percentage points and 804 executions. Duplicate processing decreased from 450 executions (0.75%) to 66 (0.11%), an 85.33% relative reduction. Inconsistent terminal states decreased from 630 (1.05%) to 132 (0.22%), while event loss decreased from 0.31% to 0.06%. Under controlled failures, successful recovery increased from 10,308/12,000 (85.90%) to 11,544/12,000 (96.20%). Unreconciled cases decreased from 648 (5.40%) to 132 (1.10%). The reported recovery odds ratio was 4.155, 95% CI 3.735–4.624, p<.001.

TABLE IV. PRIMARY PERFORMANCE AND RELIABILITY

Metric	Baseline	KIOS-RTP	Absolute change
Median latency	142 ms	151 ms	+9 ms
p95 latency	286 ms	302 ms	+16 ms
p99 latency	518 ms	540 ms	+22 ms
Mean throughput	2,310 tx/s	2,245 tx/s	−65 tx/s
Correct terminal state	59,010/60,000 = 98.35%	59,814/60,000 = 99.69%	+1.34 pp
Duplicate processing	450/60,000 = 0.75%	66/60,000 = 0.11%	−0.64 pp
Inconsistent terminal state	630/60,000 = 1.05%	132/60,000 = 0.22%	−0.83 pp
Event-loss rate	0.31%	0.06%	−0.25 pp
Unreconciled transactions under faults	648/12,000 = 5.40%	132/12,000 = 1.10%	−4.30 pp
Fault-recovery success	10,308/12,000 = 85.90%	11,544/12,000 = 96.20%	+10.30 pp
Mean recovery time	298.5 s	171.6 s	−126.9 s
Saga compensation success	88.4%	97.1%	+8.7 pp
Reconciliation success	91.6%	98.3%	+6.7 pp
Metric	**Relative change**		
Median latency	+6.34%		
p95 latency	+5.59%		
p99 latency	+4.25%		
Mean throughput	−2.81%		
Correct terminal state	+1.36%		

Duplicate processing	−85.33%
Inconsistent terminal state	−79.05%
Event-loss rate	−80.65%
Unreconciled transactions under faults	−79.63%
Fault-recovery success	+11.99%
Mean recovery time	−42.51%
Saga compensation success	+9.84%
Reconciliation success	+7.31%

These reliability improvements accompanied measurable performance costs (Figs. 1–2). Median latency increased from 142 to 151 ms, p95 latency from 286 to 302 ms, and p99 latency from 518 to 540 ms. Mean throughput decreased from 2,310 to 2,245 transactions/s, a 2.81% reduction. The findings quantify the observed reliability–performance trade-off. They do not establish operational acceptability without a verified service-level threshold.

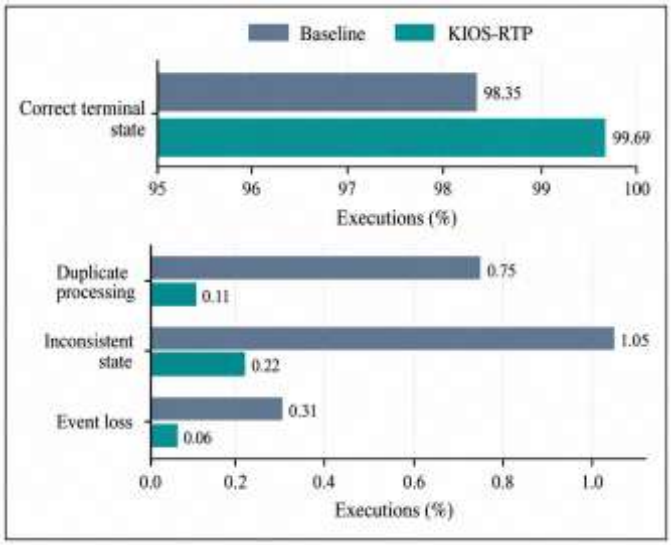

Fig. 1. Transaction correctness and residual errors. The correctness panel uses a restricted axis; error rates use a separate scale.

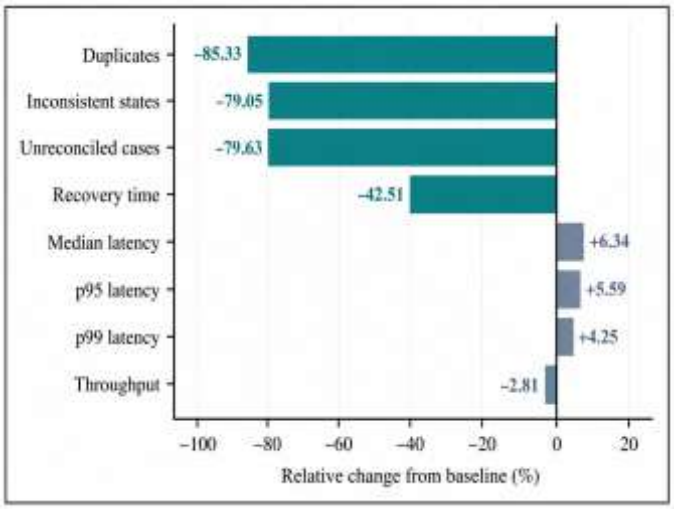

Fig. 2. Reported reliability and performance changes relative to baseline.

Saga compensation success increased from 88.4% to 97.1%, and reconciliation success increased from 91.6% to 98.3%. Their eligible denominators are not specified. These measures are retained as separate process outcomes rather than equated with aggregate correctness. Residual errors remained under KIOS-RTP, including 66 duplicate executions and 132 inconsistent terminal states; the evaluation therefore does not demonstrate error-free operation.

C. Observability and incident management

Baseline detection produced 130 true positives, 30 false negatives, 28 false positives, and 52 true negatives. KIOS-RTP produced 151 true positives, nine false negatives, 11 false positives, and 69 true negatives. Table V retains both the confusion counts and reported classification measures. The incident/window denominator is 240 per architecture, which is distinct from the transaction denominator used for payment correctness.

TABLE V. INCIDENT DETECTION AND OBSERVABILITY

Measure	Baseline	KIOS-RTP
TP / FN / FP / TN	130 / 30 / 28 / 52	151 / 9 / 11 / 69
Accuracy	75.83%	91.67%
Precision	82.28%	93.21%
Recall / sensitivity	81.25%	94.38%
Specificity	65.00%	86.25%
F1-score	81.76%	93.79%
False-positive rate	35.00%	13.75%
False-negative / miss rate	18.75%	5.62%
Mean time to detection	41.6 s	15.4 s
Mean diagnosis time	173.2 s	62.8 s
Mean recovery time	298.5 s	171.6 s
Transaction-level root cause traceable	72.5%	95.0%

Transaction-correlated observability shortened mean detection time from 41.6 to 15.4 s and mean diagnosis time from 173.2 to 62.8 s. Mean recovery time decreased from 298.5 to 171.6 s (Fig. 4). The corresponding relative reductions were 62.98%, 63.74%, and 42.51%. Root-cause traceability increased from 72.5% to 95.0%. These findings concern the integrated observability configuration; the analysis does not isolate the independent contribution of each telemetry source.

D. Hypothesis tests and model diagnostics

Table VI presents the reported hypothesis-specific estimates. H1a and H1b support lower duplicate-processing and inconsistency odds; H2 supports higher recovery odds. H3a supports improved correctness, while H3b and H3c quantify the throughput and latency penalties. These subtests do not establish H3's operational-acceptability component. H4a and H4b support shorter detection and diagnosis times. H5a–

H5c support lower unresolved-fault outcomes and shorter recovery. These are reported inferential decisions, subject to the unresolved test, model, and dependence specifications in Sections III.G and V.C.

TABLE VI. REPORTED HYPOTHESIS TESTS

Test	Effect; SE	Statistic	95% CI
H1a	β = −1.926; OR = 0.146; SE 0.132	z = −14.60	OR 0.113–0.189
H1b	β = −1.571; OR = 0.208; SE 0.096	z = −16.38	OR 0.172–0.251
H2	β = 1.424; OR = 4.155; SE 0.054	z = 26.15	OR 3.735–4.624
H3a	β = 1.685; OR = 5.395; SE 0.080	z = 21.04	OR 4.611–6.313
H3b	Δ −65 tx/s; dz = −0.93; SE 12.77	paired t = −5.09	−91.1 to −38.9 tx/s
H3c	Shift +9 ms; r = 0.16; SE 0.73-ms equivalent	z = 12.40	+7.6 to +10.4 ms
H4a	Δ −26.2 s; dz = −1.28; SE 1.62	paired t = −16.17	−29.4 to −23.0 s
H4b	Δ −110.4 s; dz = −1.70; SE 5.14	paired t = −21.48	−120.6 to −100.2 s
H5a	β = −1.636; OR = 0.195; SE 0.096	z = −16.97	OR 0.161–0.235
H5b	β = −1.723; OR = 0.178; SE 0.119	z = −14.43	OR 0.141–0.226
H5c	Reduction 111 s; rank-biserial r = 0.41; SE 8.5-s equivalent	z = −22.84	median reduction 96–126 s

Test	Decision
H1a	Supported
H1b	Supported
H2	Supported
H3a	Supported
H3b	Supported
H3c	Supported
H4a	Supported
H4b	Supported
H5a	Supported
H5b	Supported
H5c	Supported

All tests in Table VI have reported p<.001. H3 operational acceptance was not established. Latency-test selection and effect-size normalization remain unresolved.

The workload-by-architecture correctness interaction was significant, $\chi^2(2)=17.84$, adjusted p<.001, with reported Cramér's V=0.012. The transfer-type recovery interaction was not significant, z=0.03, p=.976. Reported model diagnostics were $\chi^2(8)=7.06$, p=.530, and ROC-AUC=0.812, 95% CI 0.803–0.821. The bootstrap summary reported 2,000 resamples, median recovery OR=4.14, and percentile CI 3.71–4.60. These outputs do not identify the fitted model, validation sample, or resampling unit. A non-significant calibration test does not establish perfect calibration.

E. Workload and transfer subgroups

Table VII shows that KIOS-RTP maintained higher terminal correctness across low, medium, and high workloads. At high load, correctness increased from 97.77% to 99.58%, while duplicate processing decreased from 1.08% to 0.16%. High-load throughput decreased from 2,970 to 2,872 transactions/s. Figure 6 displays latency and correctness in separate panels to avoid combining incompatible scales.

TABLE VII. PERFORMANCE BY WORKLOAD

Workload	Architecture	N	Median latency
Low	Baseline	20,000	110 ms
Low	KIOS-RTP	20,000	116 ms
Medium	Baseline	20,000	139 ms
Medium	KIOS-RTP	20,000	148 ms
High	Baseline	20,000	181 ms
High	KIOS-RTP	20,000	193 ms
Workload	**Architecture**	**p95 latency**	**Throughput**
Low	Baseline	218 ms	1,600 tx/s
Low	KIOS-RTP	230 ms	1,568 tx/s
Medium	Baseline	281 ms	2,360 tx/s
Medium	KIOS-RTP	296 ms	2,295 tx/s
High	Baseline	359 ms	2,970 tx/s
High	KIOS-RTP	380 ms	2,872 tx/s
Workload	**Architecture**	**Correct terminal state**	**Duplicate rate**
Low	Baseline	98.75%	0.45%
Low	KIOS-RTP	99.78%	0.07%
Medium	Baseline	98.43%	0.72%
Medium	KIOS-RTP	99.71%	0.10%
High	Baseline	97.77%	1.08%
High	KIOS-RTP	99.58%	0.16%

The baseline workload percentages imply 58,990 correct executions across the three equal strata, whereas the primary aggregate reports 59,010, a difference of 20 executions. This prevents exact aggregation of baseline workload correctness. The primary count and stratified percentages are retained separately because the authoritative execution records needed to resolve the difference were not supplied. No correction to either value is inferred from the reported percentages.

Table VIII reports payment-type comparisons. Duplicates decreased in both categories. Recovery improved by 10.31 percentage points for wire transfers and 10.29 points for external transfers. Both subgroup medians increased by 9 ms. The interaction did not detect a recovery difference between transfer types.

TABLE VIII. TRANSFER-TYPE SUBGROUP COMPARISONS

Subgroup	Metric	Baseline	KIOS-RTP
Wire, N=72,000	Duplicate rate	288/36,000 = 0.80%	43/36,000 = 0.12%
External, N=48,000	Duplicate rate	162/24,000 = 0.68%	23/24,000 = 0.10%
Wire fault cases, N=14,400	Recovery success	85.69%	96.00%

External fault cases, N=9,600	Recovery success	86.21%	96.50%
Wire	Median latency	145 ms	154 ms
External	Median latency	138 ms	147 ms
Subgroup	**Metric**	**Difference**	**95% CI of difference**
Wire, N=72,000	Duplicate rate	−0.68 pp	−0.78 to −0.58 pp
External, N=48,000	Duplicate rate	−0.58 pp	−0.69 to −0.47 pp
Wire fault cases, N=14,400	Recovery success	+10.31 pp	+9.42 to +11.20 pp
External fault cases, N=9,600	Recovery success	+10.29 pp	+9.21 to +11.37 pp
Wire	Median latency	+9 ms	+7 to +11 ms
External	Median latency	+9 ms	+6 to +12 ms
Subgroup	**Metric**	**Statistical test**	**Effect**
Wire, N=72,000	Duplicate rate	z = −13.27, p<.001	RR = 0.149
External, N=48,000	Duplicate rate	z = −10.01, p<.001	RR = 0.142
Wire fault cases, N=14,400	Recovery success	z = 20.81, p<.001	RR = 1.120
External fault cases, N=9,600	Recovery success	z = 17.22, p<.001	RR = 1.119
Wire	Median latency	Mann–Whitney z=8.96, p<.001	r=0.033
External	Median latency	Mann–Whitney z=7.15, p<.001	r=0.033

F. Component ablations

The component ablations showed outcome-specific patterns (Table IX; Fig. 5). Removing idempotency increased duplicate processing from 0.11% to 0.58%. Removing Saga consistency increased inconsistent terminal states from 0.22% to 0.74% and reduced recovery success from 96.2% to 89.8%. Removing reconciliation increased unreconciled outcomes from 1.1% to 3.4%. Removing correlated observability increased detection time from 15.4 to 37.9 s and mean recovery time from 171.6 to 248.1 s. These descriptive comparisons associate different layers with duplicate suppression, state consistency, reconciliation, and incident management. They do not establish statistically isolated component effects because component-specific uncertainty estimates were not supplied.

TABLE IX. MATCHED-TRACE COMPONENT ABLATIONS

Configuration	Duplicate rate	Inconsistent-state rate	Fault recovery
Full KIOS-RTP	0.11%	0.22%	96.2%
Without event-reliability layer	0.24%	0.48%	92.1%
Without idempotency layer	0.58%	0.39%	94.5%
Without Saga consistency layer	0.13%	0.74%	89.8%
Without reconciliation/recovery layer	0.12%	0.53%	91.2%
Without correlated observability	0.12%	0.24%	93.7%
Baseline asynchronous Kafka	0.75%	1.05%	85.9%
Configuration	**Unreconciled rate**	**MTTD**	**Mean recovery time**
Full KIOS-RTP	1.1%	15.4 s	171.6 s
Without event-reliability layer	2.7%	18.6 s	209.4 s
Without idempotency layer	1.8%	16.8 s	184.5 s
Without Saga consistency layer	3.2%	18.1 s	234.8 s
Without reconciliation/recovery layer	3.4%	17.5 s	242.6 s
Without correlated observability	1.5%	37.9 s	248.1 s
Baseline asynchronous Kafka	5.4%	41.6 s	298.5 s
Configuration	**p95 latency**		
Full KIOS-RTP	302 ms		

Without event-reliability layer	294 ms
Without idempotency layer	296 ms
Without Saga consistency layer	297 ms
Without reconciliation/recovery layer	295 ms
Without correlated observability	300 ms
Baseline asynchronous Kafka	286 ms

G. Robustness across failures

Table X and Figure 3 show recovery improvements across all eight fault classes. The smallest reported gain was nine percentage points for microservice termination and duplicate delivery. The largest was 11.9 points for delayed/out-of-order events. Database unavailability retained the lowest KIOS-RTP recovery percentage, 93.5%, and the longest reported recovery time, 240 s, among the listed classes.

TABLE X. RECOVERY ACROSS CONTROLLED FAULT CLASSES

Failure condition	**Baseline recovery**	**KIOS-RTP recovery**	**Improvement**
Microservice termination	88.0%	97.0%	+9.0 pp
Consumer termination/ restart	86.5%	96.0%	+9.5 pp
Processing timeout	83.0%	94.5%	+11.5 pp
Database unavailability	82.5%	93.5%	+11.0 pp
Kafka consumer delay	87.5%	97.0%	+9.5 pp
Duplicate-event delivery	89.0%	98.0%	+9.0 pp
Retry storm	84.0%	95.0%	+11.0 pp
Delayed/out-of-order events	86.7%	98.6%	+11.9 pp
Overall	85.9%	96.2%	+10.3 pp
Failure condition	**Baseline recovery time**	**KIOS-RTP recovery time**	
Microservice termination	240 s	135 s	

Consumer termination/ restart	275 s	160 s
Processing timeout	360 s	205 s
Database unavailabilit y	410 s	240 s
Kafka consumer delay	255 s	145 s
Duplicate-event delivery	190 s	105 s
Retry storm	330 s	190 s
Delayed/out-of-order events	328 s	193 s
Overall	298.5 s	171.6 s

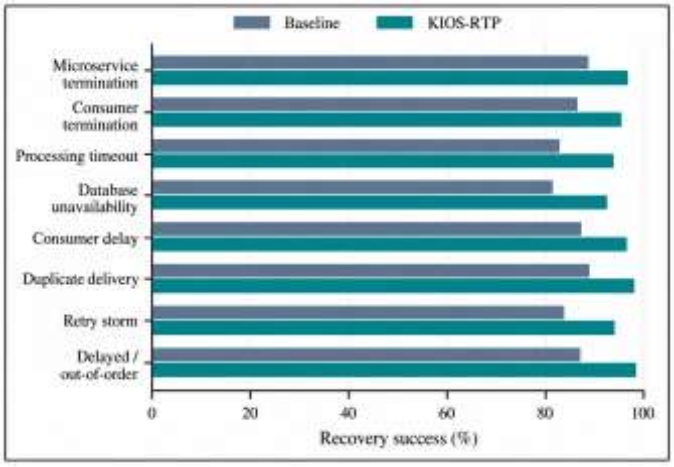

Fig. 3. Recovery success across eight controlled fault classes.

Recovery improved across all eight reported fault classes. The fault-stratified common OR was 4.08, 95% CI 3.63–4.59, $\chi^2(1)=639.4$, p<.001. The Breslow–Day test did not detect heterogeneity, $\chi^2(7)=8.41$, p=.298. This result does not establish identical effects across fault classes. Interpretation of inferential precision depends on how executions sharing fault incidents were treated.

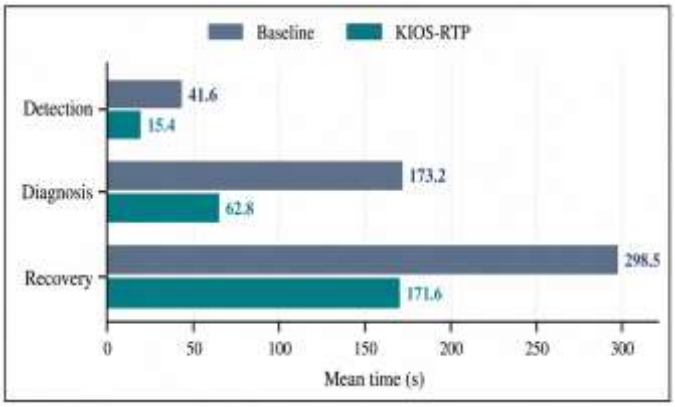

Fig. 4. Mean incident detection, diagnosis, and recovery times.

V. Discussion

A. Integrated reliability and performance

The contribution is the joint evaluation of transaction correctness, processing cost, and recovery within one payment workflow. Correctness and error outcomes address O1 and H1; latency and throughput

address the measurable part of H3; incident-management results address O3 and H4; and fault recovery and unresolved outcomes address O4, H2, and H5. Together these comparisons address O2 and O5 without establishing operational acceptance. This integrated assessment is consistent with architectural concerns in prior data-management, communication, and quality-attribute research [1]–[5]. The evidence applies to the evaluated KIOS-RTP configuration and comparator.

The absolute and relative changes describe different aspects of the trade-off. Duplicate processing decreased by 0.64 percentage points, corresponding to the reported 85.33% relative reduction; terminal correctness increased by 1.34 percentage points. KIOS-RTP nevertheless retained 66 duplicate executions and 132 inconsistent terminal states. Recovery reached 96.20%, with database unavailability remaining the weakest reported class at 93.5% and 240 s mean recovery time. Median latency increased by 9 ms and throughput decreased by 65 transactions/s. These findings quantify observed benefits and residual failures rather than demonstrating error-free or universally acceptable operation.

B. Observability as part of recovery

Transaction correlation connects a payment's events with its processing and recovery history. The observed detection and diagnosis improvements support assessing visibility alongside consistency controls [8]–[16]. The observability ablation showed comparatively small changes in duplicate and inconsistent-state rates alongside longer detection and recovery times. This descriptive pattern separates transaction-protection outcomes from incident-management outcomes without assigning a causal effect to each telemetry source.

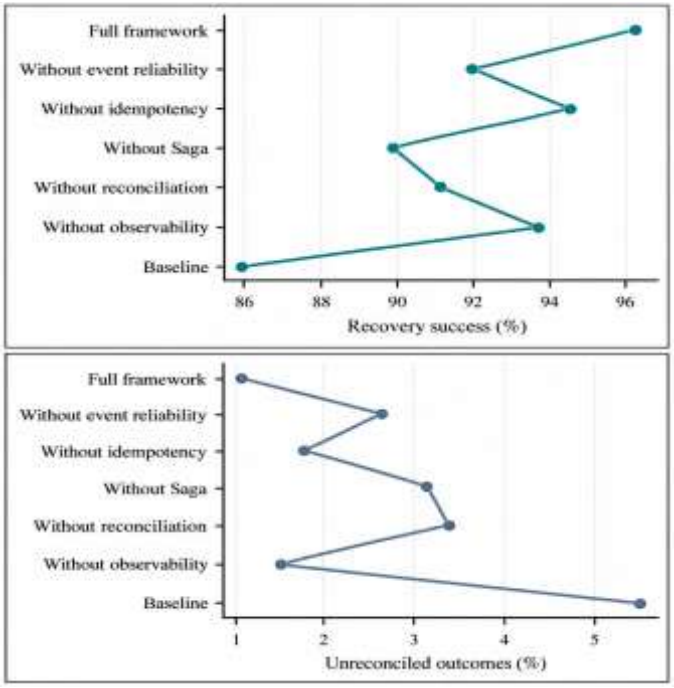

Fig. 5. Recovery success and unreconciled outcomes in matched-trace ablations.

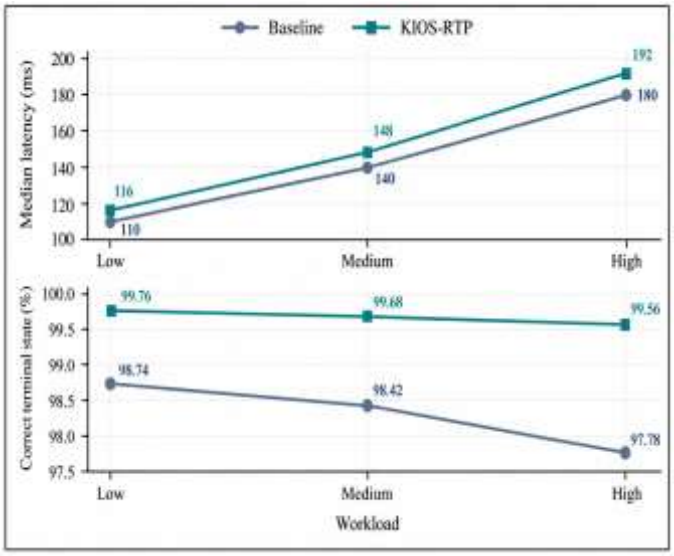

Fig. 6. Median latency and terminal correctness across workload strata.

C. Scope and threats to validity

Implementation reproducibility is limited by unavailable deployment and configuration records. Missing details include experiment dates and setting; hardware, operating system, network arrangement, service counts and resource limits; software versions; broker, partition and replication settings; acknowledgement, timeout, retry and replay policies; event schemas and transition rules; atomic duplicate-registry and financial updates; concurrent retry handling and identifier retention; Saga coordination and failed compensation; and reconciliation timing and resolution rules. The architecture can therefore be interpreted at the mechanism level, but its execution cannot be independently reconstructed. Related research supplies methodological context [18], [19], [24]–[35], rather than study-specific settings.

Internal-validity and inferential limitations concern workload provenance, inclusion/exclusion rules, missingness, pairing keys, execution order, warm-up, resets, run duration, cache effects, baseline safeguards, and infrastructure equivalence. Original traces, repeated executions, workload strata, fault incidents, and ablation samples are not fully linked. Fault-injection tooling, timing, duration, intensity, restoration procedures, observation windows, alert rules, and traceability adjudication are unspecified. Shared traces, runs, services, and incidents may induce dependence; the record does not establish dependence-aware variance or resampling. Exact latency-test selection, effect-size normalization, model predictors and coding, diagnostic validation sample, multiplicity families, and confidence-interval methods remain unresolved.

Measurement limitations include the unreported correctness oracle, complete financial invariants, assessment deadlines, latency timestamp boundaries, clock synchronization, throughput, interval, compensation/reconciliation denominators, and handling of missing or unsuccessful observations. The baseline workload total differs from the primary aggregate by 20 correct executions and requires record-level reconciliation. Access permissions and data/code availability were not established. The bibliography through 2022 does not certify experiment dates. Monetary loss, customer outcomes, regulatory compliance, security assurance, and live settlement integrity were not evaluated. Conclusions remain confined to the supplied execution, incident, and matched-trace outcomes.

VI. Conclusion

KIOS-RTP combined event reliability, idempotency, Saga coordination, reconciliation, recovery, and transaction-correlated observability in the evaluated Kafka payment workflow. The reported comparison showed higher terminal correctness, fewer duplicates and inconsistent states, improved fault recovery, and shorter detection and diagnosis times, accompanied by higher latency and lower throughput. Descriptive ablations linked component removal with different outcome patterns. The findings support integrated evaluation of correctness, recoverability, and processing cost against the specified baseline. Operational acceptance and independently reproducible inference remain unestablished until implementation records, the workload count discrepancy, and statistical specifications are resolved.

References

[1] R. Laigner, Y. Zhou, M. A. V. Salles, et al., “Data management in microservices: state of the practice, challenges, and research directions,” Proceedings of the VLDB Endowment, vol. 14, no. 13, pp. 3348-3361, 2021, doi: 10.14778/3484224.3484232.

[2] I. Karabey Aksakalli, T. Çelik, A. B. Can, et al., “Deployment and communication patterns in microservice architectures: A systematic literature review,” Journal of Systems and Software, vol. 180, Art. no. 111014, 2021, doi: 10.1016/j.jss.2021.111014.

[3] S. Li, H. Zhang, Z. Jia, et al., “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review,” Information and Software Technology, vol. 131, Art. no. 106449, 2021, doi: 10.1016/j.infsof.2020.106449.

[4] M. Söylemez, B. Tekinerdogan, A. Kolukısa Tarhan, “Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review,” Applied Sciences, vol. 12, no. 11, Art. no. 5507, 2022, doi: 10.3390/app12115507.

[5] E. Daraghmi, C. Zhang, S. Yuan, “Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture,” Applied Sciences, vol. 12, no. 12, Art. no. 6242, 2022, doi: 10.3390/app12126242.

[6] M. Djerou, O. Tibermacine, “SAGA Distributed Transactions Verification Using Maude,” 2022 International Symposium on iNnovative Informatics of Biskra (ISNIB), pp. 1-6, 2022, doi: 10.1109/isnib57382.2022.10076050.

[7] M. Žežulka, O. Chaloupka, B. Rossi, “Integrating Distributed Tracing into the Narayana Transaction Manager,” Proceedings of the 6th International Conference on Complexity, Future Information Systems and Risk, pp. 55-62, 2021, doi: 10.5220/0010448200550062.

[8] A. Bento, J. Correia, R. Filipe, et al., “Automated Analysis of Distributed Tracing: Challenges and Research Directions,” Journal of Grid Computing, vol. 19, no. 1, Art. no. 9, 2021, doi: 10.1007/s10723-021-09551-5.

[9] M. Usman, S. Ferlin, A. Brunstrom, et al., "A Survey on Observability of Distributed Edge & Container-Based Microservices," IEEE Access, vol. 10, pp. 86904-86919, 2022,
doi: 10.1109/access.2022.3193102.
[10] B. Li, X. Peng, Q. Xiang, et al., "Enjoy your observability: an industrial survey of microservice tracing and analysis," Empirical Software Engineering, vol. 27, no. 1, Art. no. 25, 2022,
doi: 10.1007/s10664-021-10063-9.
[11] D. Ernst, S. Tai, "Offline Trace Generation for Microservice Observability," 2021 IEEE 25th International Enterprise Distributed Object Computing Workshop (EDOCW), pp. 308-317, 2021, doi: 10.1109/edocw52865.2021.00062.
[12] J. Rios, S. Jha, L. Shwartz, "Localizing and Explaining Faults in Microservices Using Distributed Tracing," 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), pp. 489-499, 2022, doi: 10.1109/cloud55607.2022.00072.
[13] M. C. Borges, S. Werner, A. Kilic, "FaaSter Troubleshooting - Evaluating Distributed Tracing Approaches for Serverless Applications," 2021 IEEE International Conference on Cloud Engineering (IC2E), pp. 83-90, 2021, doi: 10.1109/ic2e52221.2021.00022.
[14] L. Gelle, N. Ezzati-Jivan, M. R. Dagenais, "Combining Distributed and Kernel Tracing for Performance Analysis of Cloud Applications," Electronics, vol. 10, no. 21, Art. no. 2610, 2021, doi: 10.3390/electronics10212610.
[15] C. Casse, P. Berthou, P. Owezarski, et al., "A Tracing Based Model to Identify Bottlenecks in Physically Distributed Applications," 2022 International Conference on Information Networking (ICOIN), pp. 226-231, 2022, doi: 10.1109/icoin53446.2022.9687217.
[16] J. Ruuskanen, A. Cervin, "Distributed online extraction of a fluid model for microservice applications using local tracing data," 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), pp. 179-190, 2022, doi: 10.1109/cloud55607.2022.00037.
[17] S. Jacob, Y. Qiao, B. Lee, "Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing," Proceedings of the 7th International Conference on Information Systems Security and Privacy, pp. 588-595, 2021, doi: 10.5220/0010308905880595.
[18] C. S. Meiklejohn, A. Estrada, Y. Song, et al., "Service-Level Fault Injection Testing," Proceedings of the ACM Symposium on Cloud Computing, pp. 388-402, 2021, doi: 10.1145/3472883.3487005.
[19] J. Simonsson, L. Zhang, B. Morin, et al., "Observability and chaos engineering on system calls for containerized applications in Docker," Future Generation Computer Systems, vol. 122, pp. 117-129, 2021, doi: 10.1016/j.future.2021.04.001.
[20] V. Bushong, A. S. Abdelfattah, A. A. Maruf, et al., "On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study," Applied Sciences, vol. 11, no. 17, Art. no. 7856, 2021, doi: 10.3390/app11177856.
[21] M. G. de Almeida, E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," Applied Sciences, vol. 12, no. 6, Art. no. 3023, 2022, doi: 10.3390/app12063023.
[22] D. Berardi, S. Giallorenzo, J. Mauro, et al., "Microservice security: a systematic literature review," PeerJ Computer Science, vol. 7, Art. no. e779, 2022, doi: 10.7717/peerj-cs.779.
[23] A. Pereira-Vale, E. B. Fernandez, R. Monge, et al., "Security in microservice-based systems: A Multivocal literature review," Computers & Security, vol. 103, Art. no. 102200, 2021, doi: 10.1016/j.cose.2021.102200.
[24] A. Al-Said Ahmad, P. Andras, "Scalability resilience framework using application-level fault injection for cloud-based software services," Journal of Cloud Computing, vol. 11, no. 1, Art. no. 1, 2022, doi: 10.1186/s13677-021-00277-z.
[25] J. A. Rasheedh, S. Saradha, "Reactive Microservices Architecture Using a Framework of Fault Tolerance Mechanisms," 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC), pp. 146-150, 2021, doi: 10.1109/icesc51422.2021.9532893.
[26] F. Silva, V. Lelli, I. Santos, et al., "Towards a Fault Taxonomy for Microservices-Based Applications," Proceedings of the XXXVI Brazilian Symposium on Software Engineering, pp. 247-256, 2022, doi: 10.1145/3555228.3555245.
[27] Takdir, H. Kitagawa, T. Amagasa, "Region-based Sub-Snapshot (RegSnap): Enhanced Fault Tolerance in Distributed Stream Processing with Partial Snapshot," 2022 IEEE International Conference on Big Data (Big Data), pp. 3374-3382, 2022,
doi: 10.1109/bigdata55660.2022.10020607.
[28] X. Wang, C. Zhang, J. Fang, et al., "A comprehensive study on fault tolerance in stream processing systems," Frontiers of Computer Science, vol. 16, no. 2, Art. no. 162603, 2022,
doi: 10.1007/s11704-020-0248-x.
[29] S. Jayasekara, S. Karunasekera, A. Harwood, "Optimizing checkpoint-based fault-tolerance in distributed stream processing systems: Theory to practice," Software: Practice and Experience, vol. 52, no. 1, pp. 296-315, 2022, doi: 10.1002/spe.3021.

[30] S. M. A. Akber, H. Chen, H. Jin, "FATM: A failure-aware adaptive fault tolerance model for distributed stream processing systems," Concurrency and Computation: Practice and Experience, vol. 33, no. 10, Art. no. e6167, 2021, doi: 10.1002/cpe.6167.
[31] G. van Dongen, D. V. D. Poel, "A Performance Analysis of Fault Recovery in Stream Processing Frameworks," IEEE Access, vol. 9, pp. 93745-93763, 2021, doi: 10.1109/access.2021.3093208.
[32] G. Theodorakis, F. Kounelis, P. Pietzuch, et al., "Scabbard: single-node fault-tolerant stream processing," Proceedings of the VLDB Endowment, vol. 15, no. 2, pp. 361-374, 2021, doi: 10.14778/3489496.3489515.
[33] R. Dautov, S. Distefano, "Stream Processing on Clustered Edge Devices," IEEE Transactions on Cloud Computing, vol. 10, no. 2, pp. 885-898, 2022, doi: 10.1109/tcc.2020.2983402.
[34] M. Weißbach, T. Springer, "Performance of Databases Used in Data Stream Processing Environments," Proceedings of the 12th International Conference on Cloud Computing and Services Science, pp. 15-26, 2022, doi: 10.5220/0011018300003200.
[35] Q. Ye, M. Lu, "SPOT: Testing Stream Processing Programs with Symbolic Execution and Stream Synthesizing," Applied Sciences, vol. 11, no. 17, Art. no. 8057, 2021, doi: 10.3390/app11178057.
[36] Gokul, "Lightweight Linux-powered edge systems: Facilitating secure and efficient container workloads at the edge," 2022, doi: 10.52783/jes.8583.
[37] G. C. P. Reddy, "Asymptotic analysis in cloud resource allocation: Scalability of virtual machines, scheduling complexity, and auto-scaling mechanisms," Journal of Computational Analysis and Applications, vol. 27, no. 7, pp. 23–43, 2019. [Online]. Available: https://eudoxuspress.com/index.php/pub/article/view/1951
[38] R. S. Kadali and G. C. P. Reddy, "Reducing latency and improving throughput for NFV workloads using SRv6 in private cloud networks," International Journal of Communication Networks and Information Security, vol. 11, no. 3, pp. 432–447, 2019. [Online]. Available: https://www.ijcnis.org/index.php/ijcnis/article/view/8048