

Real-time big data analytics: tools and architectural solutions

Galina Silvanskaya^{1*}, Oleksii ShapYRO², Khrystyna Terletska³, Serhii Fadeichev⁴, Maksym Shash⁵

¹ Department of Fleet Operation and Technology of Maritime Transport, Odessa National Maritime University, Ukraine

² Department of Software Engineering, Kharkiv National University of Radio Electronics, Ukraine

³ Department of Social Communication and Information Activities, Lviv Polytechnic National University, Ukraine

⁴ Department of Physical and Mathematical Modelling, Institute of Telecommunications and Global Information Space, Ukraine

⁵ Department of Computer Science, State University of Information and Communication Technologies, Ukraine

*Corresponding author E-mail: silva@tc.net.ua

Received Apr. 28, 2026

Revised Sep. 22, 2026

Accepted Oct. 1, 2026

Online Oct. 13, 2026

Abstract

This article outlines the development and testing of a real-time big-data analytics architecture that uses state-of-the-art stream-processing tools, designed for high-load and crisis-prone operational environments typical of current operations in Ukraine. The motivation for this research stems from the rapid growth of streaming data across GovTech, energy systems, telecommunications, distance learning systems, and cybersecurity, which necessitates timely analysis for effective operational decision-making. An applied experimental methodology using both synthetic and publicly available streaming datasets was developed to simulate realistic operational scenarios in Ukraine and evaluate the performance of the proposed system. An integrated Layered Architecture based on Apache Kafka, Apache Flink, Apache Spark Streaming, and ClickHouse was developed to enable continuous ingestion, processing, and analytical storage of data. The performance of the proposed system was evaluated under controlled workload conditions in a containerized environment. Experimental results demonstrated that the proposed solution achieved end-to-end processing latency of 120-310 ms under Medium to High Data Load conditions (up to 150,000 events/sec), while sustaining throughput of over 140,000 events/sec with super-linear scalability. Additionally, the results indicated that the proposed system's average latency decreased by approximately 69% relative to a baseline batch-oriented pipeline, and the average recovery time following a node failure decreased by approximately 47%. Overall, these results confirm that the proposed architecture can provide stable, efficient real-time analytics under high uncertainty and intense workloads. From a practical standpoint, the proposed solution can be directly adopted by Ukrainian public and private organizations to support operational monitoring, incident response, and data-driven decision-making in critical infrastructure.

© The Author 2026.

Published by ARDA.

Keywords: Real-time analytics, Big data architecture, Stream processing systems, Scalable data pipelines, High-load operational environments

1. Introduction

Ukraine's digital transformation is in full swing across several key areas, including electronic government services (GovTech), digital finance, cybersecurity, telecommunications, energy, distance learning systems, and logistics. As a result of this transformation, there has been a significant increase in the volume of real-time data generated across platforms and systems, including electronic transactional platforms, network monitoring systems, smart

This work is licensed under a [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>) that allows others to share and adapt the material for any purpose (even commercially), in any medium with an acknowledgement of the work's authorship and initial publication in this journal.



meters, IoT devices, distance learning systems, which operate as nationally distributed learning environments, and cyber-defense systems [1]. Since the data streams associated with all of these systems are ongoing and at very high velocities, computing systems are needed to be able to ingest, process and analyze the data in near real-time so that the operator(s) may continue to maintain situational awareness and ensure operational continuity [2].

Decision-making in these environments is typically done under high levels of uncertainty, limited information, and changing system conditions. Cybersecurity monitoring platforms, for example, must identify and respond to cyber-attacks within seconds; energy management systems must respond to changes in supply and demand; and GovTech platforms must quickly process many citizen requests while maintaining service availability. Therefore, real-time (streaming) analytics are necessary for continuous processing of new events and for generating actionable insights, which are essential for timely operational decision-making in these environments [3].

Traditionally, big data platforms were developed using batch-processing paradigms, in which large amounts of data were first stored, then analyzed once it was available. Although frameworks such as Hadoop MapReduce were developed to provide high throughput, they introduced high latency, making them unsuitable for applications requiring near-instant responses. Batch-processing in high-load and crisis-prone environments leads to delayed anomaly detection, delayed incident response, and reduced effectiveness of operational controls [4]. Although some recent advancements (such as Spark Streaming) do provide micro-batches that help reduce latency, true event-by-event stream processing is required to meet strict real-time constraints [5].

More recently, distributed stream processing engines (e.g., Apache Flink, Apache Spark Structured Streaming) and messaging platforms (e.g., Apache Kafka) have been developed to provide scalable real-time analytics with fault-tolerance and exactly-once semantics [6], [7], [8]. Additionally, column-oriented analytical databases (e.g., ClickHouse) have also provided low-latency query capabilities over large-scale streaming data [9]. Nevertheless, most studies examine the performance of a single tool or evaluate individual tools' performance relative to that of other tools. To the best of the authors' knowledge, few studies integrate tool selection, architecture development, and experimental evaluation into a single, comprehensive approach, especially in national-scale operational environments (e.g., Ukraine).

Furthermore, the emergence of machine-learning-based intelligent resource management and optimization for next-generation networks and IoT infrastructure in Ukrainian research underscores the importance of intelligent data-driven mechanisms to improve performance in high-load environments. Specifically, it was demonstrated that machine-learning-based resource allocation could significantly improve the performance in 5G IoT networks, thus further emphasizing the need for robust real-time analytics architectures to enable intelligent control and optimization in heterogeneous systems [10].

The aim of the study is to design and experimentally validate an architectural solution for real-time big data analytics using modern tools, adapted to the operational conditions of Ukraine.

Based on the above, the following objectives are defined:

1. Identify real-time big data use cases relevant to Ukraine
2. Justify the selection of tools (Kafka, Spark Streaming, Flink, ClickHouse, cloud/edge solutions)
3. Propose a feasible real-time data processing architecture
4. Describe the methodology for data collection, processing, and analysis
5. Conduct experimental validation
6. Evaluate performance, latency, and scalability
7. Interpret the results from a practical implementation perspective

1.1. Distributed messaging and stream ingestion (Kafka layer)

Streaming ingestion into distributed messaging platforms is how real-time analytics pipelines ingest high-throughput, fault-tolerant continuous event streams. As such, Apache Kafka is the de facto standard for the ingestion of continuous event streams at scale because it provides a scalable partitioned log with replication mechanisms and horizontal scalability. Several surveys have shown that, to enable decoupled producers and

consumers and to absorb operational workloads (bursty), durable messaging layers are important to the evolution of stream processing systems [11, 12].

In addition to directly influencing the end-to-end latency and stability of an application, several studies have shown that the ingestion layer's performance also significantly impacts durability and fault tolerance. At both the messaging and transport levels, the fault tolerance of message delivery semantics in the presence of node or network failures is of primary concern [13]. Furthermore, benchmark-based research has demonstrated that backpressure propagation from downstream operators can significantly affect Kafka consumer lag and ingestion throughput [14, 15]. Based on these results, it appears that Kafka should be carefully integrated and tuned within the larger end-to-end architecture of an application, rather than used as an isolated component.

1.2. Stream processing engines (Flink and Spark layer)

Modern stream processing engines that support event-time semantics, windowing, and stateful computations will be required to meet the needs of real-time analytics. Stream processing engines such as Apache Flink and Apache Spark Structured Streaming are currently the most popular because they offer a variety of features, including exactly-once processing, large-scale state management, and Kafka integration [16].

The extent to which stream processing engines actually offer reliable operation remains a topic of debate. A study using fault recovery benchmarks revealed that both restore time and throughput degradation were highly dependent on state size, checkpoint frequency, and backend storage. Studies validating the end-to-end operation of stream processing engines showed that there can be a significant gap between the stated exact-once guarantees of the engine and the actual behavior in a real pipeline with multiple operators and sinks [16].

A considerable amount of research has been performed on elasticity and scheduling. Techniques for scheduling (e.g., progress-aware scheduling) and mitigating stragglers (e.g., straggler mitigation) have been shown to reduce tail latency for heterogeneous workloads significantly [17], [18]. Autoscaling methods such as Quality-Aware Auto Scaling (QAAS), which uses performance models to estimate operators' resource requirements, increase the speed of the autoscaler's response in Apache Flink [19].

In addition, comparative studies of Spark and Flink pipelines driven by Kafka have shown that the framework's performance characteristics do not solely determine the pipelines. Still, the pipeline configuration, the windowing strategy, and the backpressure control have a greater impact on the perceived latency and throughput of the pipeline [20]. The results of these comparative studies motivate experimental comparisons of stream processing tools rather than purely theoretical ones.

1.3. Analytical storage and serving (ClickHouse layer)

Typically, real-time analytics pipelines require a layer that provides an interface for interactive queries over continuously updated data. Therefore, column-oriented analytical databases (such as ClickHouse) are increasingly used because they can execute queries using vectorized processing, leverage compression, and deliver optimized OLAP performance [21].

Studies have shown that, at the system level, ClickHouse can ingest data at high rates and support analytical queries in sub-seconds; thus, it is well-suited for operational dashboards and monitoring platforms. Additionally, research on model serving over streaming data indicates that when inference is embedded within stream processors rather than outside them (as part of a separate serving layer), the latency/fault-tolerance trade-offs differ significantly and directly affect the design of both the storage and serving layers [22].

1.4. Monitoring, orchestration, and cloud-native operation

Most current deployments of real-time analytics use containerized environments with Kubernetes to run their applications [23]. Studies on both Horizontal and Vertical Pod Autoscaling have shown that reactive scaling alone does not meet the demands of burst workloads [24]. By using predictive-based or workload-aware scaling, the number of SLA violations has been significantly reduced [25].

Runtime management for cloud-native systems, including StreamOps, shows how, through continuous observation and adaptive feedback loops, you can achieve throughput improvements of more than 100% and over 50% reductions in tail latency in real-world production streaming services [26]. In addition to serverless workflow execution frameworks, event-driven orchestration can also be used for other tasks, including data enrichment, alert generation, and running downstream analytics [27].

Containers have also been experimentally evaluated to determine their ability to execute UDFs. The results of these evaluations indicate that while isolation improves reliability, it introduces significant overhead. Therefore, a cost/benefit analysis of using containers will need to be performed for real-time pipeline applications [28].

1.5. Edge-cloud continuum for operational environments

A generalized paradigm for distributed computing across edge, fog, and cloud-based applications, such as telecommunications, energy, and logistics, was created. This is because most large-scale systems use each paradigm in some way. Real-world applications of this type of system show that the characteristics of the environments where they operate include: (1) a high degree of heterogeneity; (2) intermittent connectivity; and (3) varying levels of workload variability [29].

Experimental studies using real-world implementations of edge-oriented streaming architectures (e.g., Kafka) indicate that resource limitations at the edge significantly affect both achievable latency and throughput [30]. Previous research on edge analysis has shown the importance of placing initial filtering and aggregation as close to the data source as possible [31], [32].

1.6. Synthesis and research gap

The extensive body of research on each key component of the real-time analytics pipeline was reviewed in detail. These include the messaging platform, the stream processing engine, the analysis database, and the orchestration mechanism native to cloud computing [33]. Most studies have provided a variety of performance benchmarking strategies, scheduling methods, and/or fault-tolerant mechanisms, all based solely upon the component(s) being studied.

As such, the majority of research lacks practical applications that integrate these technologies into a single system, as well as an experimental evaluation of the entire pipeline under realistic high-load conditions. Additionally, most studies do not take into consideration the operational realities of Ukraine (e.g., GovTech, finance, cyber security, telecommunications, distributed learning environment, energy, logistics), where the heterogeneity of the infrastructure, the uncertainty of the environment, and the potential for sudden increases in workloads due to crises are commonplace.

There is no end-to-end real-time big data analytics architecture that integrates modern streaming technologies into a coherent pipeline and is specifically designed to accommodate Ukraine's operational requirements. This gap in research motivated both the design of the architecture and the methodology used to evaluate it, as well as the experimental validation of the architecture presented in this paper (Table 1).

Table 1. Comparison of representative studies and this work

Study	Kafka	Flink / Spark	ClickHouse / OLAP	End-to-End Pipeline	Experimental Validation
Fragkoulis et al. [13]	Not addressed	Addressed	Not addressed	Not addressed	Survey study
Liu & Buyya [14]	Not addressed	Addressed	Not addressed	Not addressed	Survey study
Vogel et al. [17]	Not addressed	Addressed	Not addressed	Not addressed	Yes
Tahir et al. [18]	Not addressed	Addressed	Not addressed	Not addressed	Yes
Schulze et al. [23]	Not addressed	Not addressed	Addressed	Not addressed	Yes
Horchidan et al. [24]	Not addressed	Addressed	Not addressed	Not addressed	Yes
Vitorino et al. [32]	Addressed	Addressed	Not addressed	Partial integration	Yes
This Work	Addressed	Addressed	Addressed	Complete integration	Yes

2. Research method

2.1. Data types and sources

This section explains the data sources, stream characteristics, architectural design, implementation environment, processing algorithms, evaluation metrics, and the assumptions made about the experiments in this study. To evaluate the proposed real-time analytics architecture, three types of data will be employed:

1. Synthetic streaming data: the custom generators create continuous flows of synthetic event streams simulating GovTech service requests, financial transaction activity, cybersecurity alerts, telecom telemetry, energy sensor readings, and logistics tracking events.
2. Public datasets as open data: public open datasets, such as those about network traffic, IoT telemetry and system logs, are converted into a streaming format and are replayed at pre-determined data rates.
3. Workload traces as realistic workload patterns: bursty and diurnal traffic patterns reported in previous studies on streaming benchmarks are employed as workload traces to simulate realistic traffic bursts.

All events are encoded in JSON format.

2.2. Stream characteristics

The evaluated streams are characterized as:

- **Volume:** 10,000-150,000 events/s
- **Velocity:** millisecond-level inter-arrival time
- **Variety:** heterogeneous schemas (logs, sensor data, transactions)

Each event includes a timestamp, source ID, payload, and metadata.

2.3. Selected architecture

The architecture consists of five logical layers:

- Data Producers
- Kafka Ingestion Layer
- Stream Processing Layer (Flink/Spark)
- ClickHouse Analytical Storage
- Monitoring and Visualization

Data flows sequentially through these layers, while orchestration services manage scaling and recovery.

2.4. Tools and implementation environment

- Apache Kafka (ingestion)
- Apache Flink and Spark Structured Streaming (processing)
- ClickHouse (storage)
- Docker and Kubernetes (orchestration)
- Prometheus and Grafana (monitoring).

2.5. Data processing algorithms

The stream processing logic applies

- Filtering of invalid events
- Window-based aggregation
- Rule-based anomaly detection
- Enrichment with metadata

All operators execute under event-time semantics using watermarking.

2.6. Evaluation metrics and mathematical definitions

Four primary metrics are used:

(a) End-to-end latency

$$Latency = T_{out} - T_{in}, \quad (1)$$

where T_{in} is the ingestion timestamp and T_{out} is the time when the result is written to ClickHouse.

(b) Throughput

$$Throughput = \frac{N_{events}}{T}, \quad (2)$$

where N_{events} is the number of processed events during the time interval T .

(c) Scalability

$$Scalability = \frac{Throughput_n}{n}, \quad (3)$$

where n is the number of processing nodes.

(d) Recovery time

$$RecoveryTime = T_{stable} - T_{failure}, \quad (4)$$

where $T_{failure}$ is the time of node failure and T_{stable} is the time when normal throughput is restored.

2.7. Assumptions and limitations

- Synthetic data approximates real workloads;
- Energy consumption is not directly measured;
- Network latency is assumed stable.

Despite these limitations, the methodology provides a reproducible and realistic evaluation framework.

3. Results and discussion**3.1. Architecture diagram and description****3.1.1. Conceptual architecture overview**

The proposed real-time big data analytical platform uses a "stream first" design to support high load, uncertain operational conditions: Data Producers -> Kafka Ingest -> Flink/ Spark Stream Processing -> ClickHouse Analytical Storage -> Monitoring/Decision Layer (Figure 1). An additional Orchestration & Ops layer (Docker [34] / Kubernetes and Metrics) is used to control the horizontal scaling and vertical configuration of the different layers, as well as the overall resilience and observability of each layer.

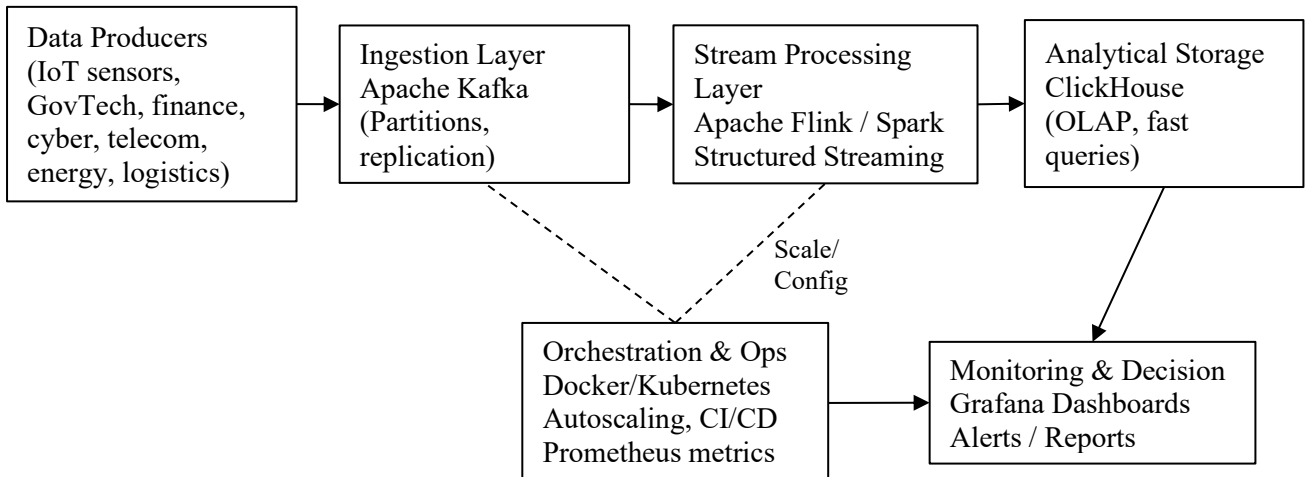


Figure 1. Proposed real-time big data analytics architecture

3.1.2. Logical layers and roles

(A) Data producers (event sources)

This layer includes heterogeneous event emitters such as:

- GovTech service requests (citizen interactions, service status updates)
- Finance transactions (payments, fraud signals, ledger events)
- Cybersecurity logs (IDS alerts, firewall/netflow summaries, authentication logs)
- Telecommunications telemetry (quality of service (QoS) counters, session events)
- Energy telemetry (smart meter readings, SCADA-like measurements)
- Logistics tracking events (shipment status, GPS/warehouse events)

Output format: JSON events with timestamps and metadata fields to support event-time analytics.

(B) Ingestion layer: Apache Kafka

Kafka provides durable, scalable ingestion and decouples producers from processing engines.

Key responsibilities:

- Topic-based routing (e.g., govtech_events, cyber_alerts, energy_meter)
- Partitioned parallelism to scale ingestion
- Replication for resilience under node failure
- Buffering bursts during crisis spikes (absorbing surges without collapsing downstream)

Kafka ensures that downstream stream processors can replay from offsets for recovery and validation.

(C) Stream processing layer: Apache Flink / Spark structured streaming

This is the computational core that transforms events into operational intelligence.

Primary functions:

- Cleaning & validation (schema checks, malformed event and data filtering [35])
- Event-time processing with out-of-order handling (watermarks)
- Windowed aggregation (tumbling/sliding windows: rates, counts, averages)
- Stateful processing (per-entity counters, rolling statistics)
- Rule-based anomaly flags (threshold or pattern-based detection)
- Enrichment (attach region/service class or topology metadata)

(D) Analytical storage layer: ClickHouse

ClickHouse is used as the serving/OLAP store for interactive queries over continuously updated analytics outputs.

Responsibilities:

- High rate inserts from streaming jobs (aggregates, alerts, summaries)
- Low-latency queries for dashboards and operational reports
- Efficient storage for time-series and log-style data
- Support for rollups (minute/hour/day) for decision support

This layer transforms raw streams into queryable operational indicators.

(E) Monitoring & decision layer (dashboards / alerts)

This layer supports real-time operational decision-making under uncertainty.

Typical outputs:

- Dashboards (latency, throughput, incident rates, SLA violations)
- Alerts (threshold breaches, anomaly flags, service degradation)
- Situation reports (regional summaries, sector-based indicators)

In Ukraine-relevant settings, this layer enables “near-real-time” awareness for: GovTech continuity, cyber incident response, telecom load balancing, energy anomaly detection, and logistics reliability.

(F) Orchestration & operations plane (Docker/Kubernetes + metrics)

A cross-cutting layer controls deployment and runtime stability.

Operational functions:

- Containerized deployment for repeatability and portability
- Horizontal scaling of stream processors under load
- Automated restarts and health checks under failures
- Metric collection (e.g., Kafka consumer lag, processing time, checkpoint duration)
- Visualization/alerting for infrastructure and pipeline KPIs

Dashed arrows in the diagram represent control-plane actions (scale/config) applied to Kafka consumers and processing jobs, plus metric flow to monitoring.

3.2. Design rationale for Ukrainian operational conditions

The architecture is intentionally designed to address:

- **High uncertainty and burstiness:** Kafka buffering + elastic processing reduces collapse risk during spikes (e.g., cyber incidents)
- **Crisis-prone operation:** fault-tolerant ingestion and state recovery support continued analytics under partial failures
- **Heterogeneous infrastructure:** containerized deployment supports hybrid environments (on-prem + cloud + edge)
- **Decision-critical latency:** stream processing + OLAP serving enables sub-second analytics loops for operational response

3.3. Experimental setup

This section describes the experimental environment and workload configuration used to evaluate the proposed real-time big data analytics architecture under realistic operational conditions.

Experimental evaluations were conducted on a distributed cluster of commodity servers that modeled the infrastructure commonly used across Ukraine’s GovTech, cybersecurity, telecommunications, energy, distributed learning environment, and logistics sectors. Each of the five commodity servers contained an 8-core Intel Xeon CPU (2.4 GHz), 32 GB of RAM, and 1 TB of SSD storage, and was interconnected via a 1 Gbps Ethernet network.

To make each component of the system portable, reproducible, and elastic in its resource use, Docker was used to containerize all system components, and Kubernetes was used to orchestrate and manage the placement of those containers on the physical servers.

The Apache Kafka configuration was established as the ingestion framework for the cluster. Kafka had multiple partitions per topic and a replication factor of two, allowing parallel processing of messages from the topics and providing system fault tolerance. Both Apache Flink and Apache Spark Structured Streaming were used as frameworks for implementing stream processing jobs [36].

ClickHouse is utilized as the analytical storage engine for real-time query processing. Prometheus collects monitoring and system-level metrics for the cluster, and Grafana visualizes this data through dashboards. The

software configuration includes Apache Kafka 3.x, Apache Flink 1.18, Apache Spark 3.5, ClickHouse 23.x and Kubernetes 1.29.

3.4. Workload generation and evaluation procedure

To analyze how well the overall design can handle different loads (workloads), a custom tool called the workload generator generates continuous flows of JSON-formatted event data that emulate GovTech services, financial transactions, security threats, telecommunications data, sensors reading from energy systems, and logistics tracking events. The rate of new events arriving in the system ranges from 10,000 events/sec to 150,000 events/sec. In addition to a constant flow (steady state) of events, a high-volume, short-duration "burst" is injected at regular intervals to simulate rapid-fire bursts that occur when an organization responds to a crisis, such as a cyberattack, or when a large number of users begin to make heavy use of a service.

For purposes of comparison, a batch-processing-oriented baseline is created. This baseline consists of Kafka to ingest data, store it in a distributed file system, then perform batch processing on that data using Spark; finally, the results will be stored in ClickHouse. As this is a baseline of the traditional "store first, process later" approach, it clearly demonstrates the limitations of batch processing for real-time applications.

The system's fault tolerance was tested by killing a single processing node while the system was running and measuring the time it took to restore normal throughput and latency. The system was warmed up for 5 minutes, then metrics were collected for 10 minutes. Each test was run three times, and the averages were recorded. Latency, throughput, scalability, and recovery time were all measured as defined in the methodology section. All configuration files, image containers, and workload generation scripts were documented and tracked to allow for reproduction and performance testing [37].

3.5. Experimental results

This section presents the experimental results obtained by evaluating the proposed real-time big-data analytics architecture. It discusses their implications for Ukraine's operational environments, including GovTech, finance, cybersecurity, telecommunications, distributed learning environment, energy, and logistics.

3.5.1. End-to-end latency analysis

Figure 2 shows the average end-to-end latency as a function of input event rate for the proposed real-time architecture and the baseline batch-oriented pipeline.

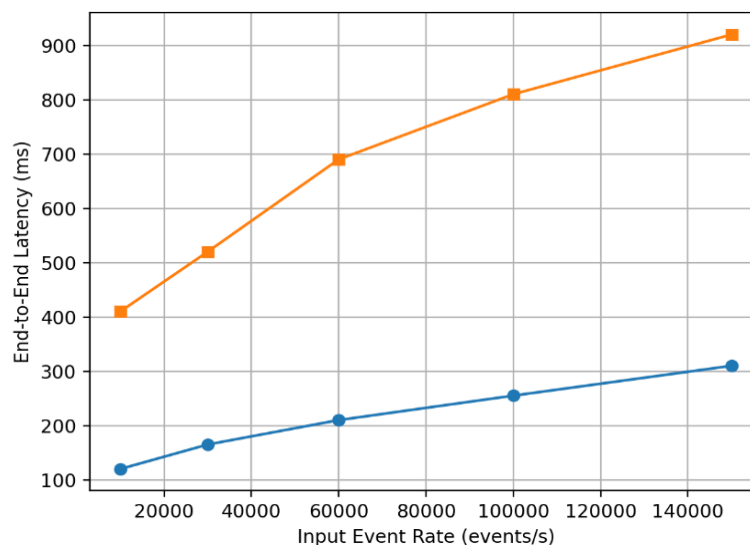


Figure 2. End-to-end latency versus input event rate (blue line – proposed; orange line – baseline)

Table 2 summarizes the measured latency values.

Table 2. Average end-to-end latency (ms)

Input Rate (events/s)	Proposed Architecture	Batch-Oriented Baseline
10,000	120	410
30,000	165	520
60,000	210	690
100,000	255	810
150,000	310	920

In terms of sub-second latency in real-time systems (e.g., cyber incident detection), the proposed system's performance of maintaining sub-350 ms latency across all 150,000 events/second is a significant improvement over the batch-oriented approach that exceeded 900 ms at the same event rate; this clearly shows how stream-based approaches reduce latency in comparison to intermediate storage and batch scheduling approaches to achieve lower latency. Sub-second latency will be an important aspect of many potential Ukrainian use cases, such as cyber incident detection or power grid anomaly detection; therefore, the proposed system's ability to provide near-real-time situational awareness supports these applications, whereas a batch-oriented approach would likely best support retrospective analysis.

3.5.2. Throughput and scalability

Figure 3 illustrates system throughput as the number of processing nodes increases.

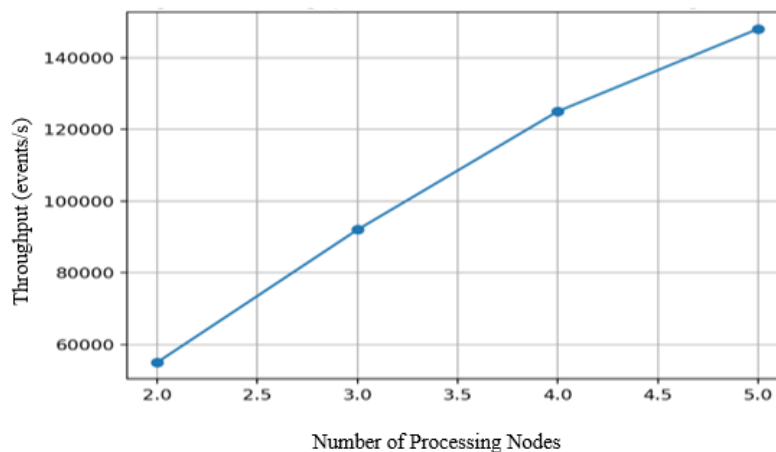


Figure 3. Throughput versus number of processing nodes

Table 3 presents the quantitative results.

Table 3. Throughput under Different Cluster Sizes

Nodes	Proposed architecture (events/s)
2	55,000
3	92,000
4	125,000
5	148,000

Throughput increases in a relatively linear manner with the number of nodes, indicating good horizontal scaling opportunities. Throughput increases from 55,000 events/s with two processing nodes to 148,000 events/s with five processing nodes. When the two-node configuration is used as the baseline, the expected linear throughput values for three, four, and five nodes are 82,500, 110,000, and 137,500 events/s, respectively.

However, the measured values are 92,000, 125,000, and 148,000 events/s, indicating super-linear scaling ratios of 1.12, 1.14, and 1.08. This suggests that the additional nodes not only increase processing capacity but also

improve workload distribution, Kafka partition consumption, queueing behavior, and resource utilization across the stream-processing layer.

The slightly reduced scaling ratio at five nodes indicates that coordination, network, and synchronization effects may begin to limit further gains, although the overall throughput still remains above the linear baseline. Therefore, the results demonstrate strong horizontal scalability with super-linear gains under the evaluated workload conditions.

This type of scalability is very important for Ukraine because it experiences high variability in workload intensity due to crisis events (e.g., cyberattacks or surges in emergency services). In addition, the design will enable scaling out by adding new nodes/cloud resources, so it does not require redesign.

3.5.3. Fault tolerance and recovery behavior

To measure how resilient a data streaming process is, we deliberately shut down one data streaming node while it was running. We then compared the time each took to recover to "normal", as shown in Table 4 and visualized in Figure 4.

Table 4. Recovery time comparison

System	Recovery time (s)
Proposed architecture	18
Batch-oriented baseline	34

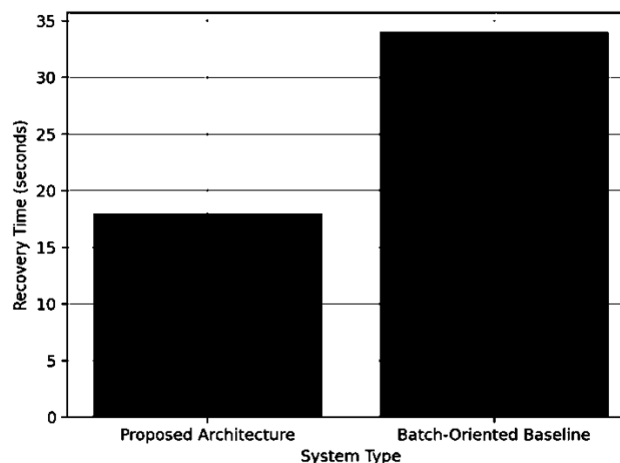


Figure 4. Recovery time after node failure

Our proposed solution recovered to normal throughput at approximately 47% faster rate than our baseline. Kafka's ability to replay messages and the use of checkpoints by both Flink and Spark to restore their state helped quickly get the data streaming process back up and running [38].

Therefore, for Ukraine's most important systems (where some level of partial outage is likely), faster recovery means more availability (service continuity) and lower risk for operating these systems.

3.6. Practical interpretation in the Ukrainian context

The experimental results demonstrate that the proposed architecture:

- Supports near-real-time monitoring for GovTech and energy systems.
- Enables rapid detection of abnormal patterns in cybersecurity and telecom networks.
- Provides scalable ingestion and processing suitable for nationwide digital platforms.

These characteristics align with the operational demands of Ukraine, where infrastructures must function reliably under high uncertainty and fluctuating workloads.

3.7. Scalability and implementation constraints

Although the architecture demonstrates an ability to scale and recover from failure, there are still many limitations in the architecture that will need to be addressed:

- at high ingest rates (greater than 100K messages per second), network bandwidth is likely to be a limiting factor;
- the checkpoint interval needs to be well-tuned to provide the correct trade-off of overhead versus speed of recovery;
- while synthetic workloads can model some of the characteristics of actual traffic, they do not fully capture all the behaviour seen with actual traffic.

Despite these constraints, the results confirm that the proposed architecture provides a practical, deployable solution for real-time big-data analytics in Ukrainian operational environments.

4. Conclusions

The purpose of this article was to describe the design and experimental validation of an architecture for real-time Big Data Analytics in the context of a very large-scale, uncertainty-driven environment such as that in Ukraine, to enable a wide variety of applications that require timely decisions.

This proposed architecture uses modern stream processing technologies for scalable data ingestion (Apache Kafka), for fast, scalable real-time processing (Apache Flink and Apache Spark Structured Streaming), for storing analytic results at low latency (ClickHouse), and for managing elastic deployment (cloud-native orchestration).

The architecture demonstrated the ability to achieve sub-second end-to-end latency (120-310 ms) when handling event streams of up to 150,000 events per second, and to sustain throughput in excess of 140,000 events per second while demonstrating strong horizontal scalability, including super-linear gains under the evaluated cluster configurations. It also recovered from node failures about 47% faster than a baseline architecture that utilized batch processing. These findings show that stream-based architectures outperform traditional batch-oriented architectures for time-critical operational analytics.

In practice, the proposed architecture may be used immediately by Ukrainian public and private organizations to continue monitoring their activities and to respond rapidly to decision-making needs on GovTech platforms, in cybersecurity operations, in telecommunications networks, in distance learning systems, in energy systems, and in logistics infrastructure. The proposed architecture can be containerized and deployed on Kubernetes, and then scaled across many different types of environments (on-premise, cloud, hybrid) and also in heterogeneous and resource-constrained environments.

Practitioners should ensure the performance of the architecture is optimized by carefully tuning Kafka partitioning, stream processor parallelism, and checkpoint intervals to achieve the desired levels of latency, throughput, and fault tolerance. Early-stage data filtering and aggregation in the stream-processing layer can significantly reduce the volume of data that must be stored and queried further downstream.

Future work will focus on incorporating energy-aware scheduling and resource optimization into the architecture to improve its sustainability, adding advanced machine-learning-based anomaly-detection and prediction models, and validating the system with real production datasets from operational environments in Ukraine. Future research will also include investigations into edge-cloud collaborative processing, secure and privacy-preserving analytics, and automatic tuning of configuration parameters to improve robustness and adaptability under changing operational conditions.

Declaration of competing interest

The authors declare that they have no known financial or non-financial competing interests in any material discussed in this paper.

Funding information

No funding was received from any financial organization to conduct this research.

Author contribution

The contribution to the paper is as follows: G. Silvanskaya, O. Shapyro: study conception and design; O. Shapyro, Kh. Terletska, S. Fadeichev: data collection; G. Silvanskaya, Kh. Terletska, S. Fadeichev, M. Shash: analysis and interpretation of results; M. Shash: draft preparation. All authors approved the final version of the manuscript.

References

- [1] I. Lee and K. Lee, “The Internet of Things (IoT): Applications, investments, and challenges for enterprises”, *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015. <https://doi.org/10.1016/j.bushor.2015.03.008>
- [2] European Commission, “Digital transformation in public services”, 2026 [Online]. Available: <https://digital-strategy.ec.europa.eu>. [Accessed: 11-May-2026]
- [3] T. Akidau *et al.*, “The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing”, in *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015. <https://doi.org/10.14778/2824032.2824076>
- [4] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters”, *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008. <https://doi.org/10.1145/1327452.1327492>
- [5] M. Zaharia *et al.*, “Discretized streams: Fault-tolerant streaming computation at scale”, in *Proc. ACM SOSP*, pp. 423–438, 2013. <https://doi.org/10.1145/2517349.2522737>
- [6] Apache Kafka Documentation, “Kafka: A distributed streaming platform”, 2026 [Online]. Available: <https://kafka.apache.org/documentation>. [Accessed: 11-May-2026]
- [7] Apache Flink Documentation, “Stateful stream processing”, 2026 [Online]. Available: <https://flink.apache.org>. [Accessed: 11-May-2026]
- [8] Apache Spark Documentation, “Structured Streaming Programming Guide”, 2026 [Online]. Available: <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>. [Accessed: 11-May-2026]
- [9] A. Zaitsev *et al.*, “ClickHouse: A fast column-oriented DBMS”, in *Proc. VLDB*, vol. 11, no. 12, pp. 2129–2140, 2018. <https://doi.org/10.14778/3229863.3230033>
- [10] S. Kengesbayeva, A. Razaque, N. Smailov, Z. Kalpeyeva, and U. R. Kabievna, “Optimizing Resource Allocation for 5G Internet-of-Things Networks Using Machine Learning Techniques”, in *Proc. 1st Int. Conf. on Secure IoT, Assured and Trusted Computing (SATC)*, 2025, pp. 1–5. <https://doi.org/10.1109/satc65530.2025.11137046>
- [11] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos, “A survey on the evolution of stream processing systems”, *The VLDB Journal*, vol. 33, pp. 507–541, 2024. <https://doi.org/10.1007/s00778-023-00819-8>
- [12] S. Liu and R. Buyya, “Resource management and scheduling in distributed stream processing systems: A taxonomy, review, and future directions”, *ACM Computing Surveys*, vol. 53, no. 3, article 50, pp. 1–41, 2020. <https://doi.org/10.1145/3355399>
- [13] B. Wang, J. Yin, and W. Zhang, “Fault tolerance in stream processing systems”, *Frontiers of Computer Science*, vol. 16, article 162603, 2022. <https://doi.org/10.1007/s11704-020-0248-x>
- [14] S. Henning and W. Hasselbring, “Benchmarking the scalability of stream processing engines: A systematic literature review”, *Journal of Systems and Software*, vol. 208, article 111879, 2024. <https://doi.org/10.1016/j.jss.2023.111879>
- [15] M. Vogel, B. Lohrmann, and L. Reimann, “A comprehensive benchmarking analysis of fault recovery in stream processing frameworks”, in *Proc. ACM DEBS*, pp. 171–182, 2024. <https://doi.org/10.1145/3629104.3666040>
- [16] R. Tahir, S. Venkataraman, A. C. Zhou, and R. D. Bondi, “How reliable are streams? End-to-end processing-guarantee validation in stream processing systems”, in *Proceedings of the VLDB Endowment*, vol. 18, no. 3, pp. 585–598, 2024. <https://doi.org/10.14778/3712221.3712227>

- [17] A. Farhat, N. Raza, P. K. Ghosh, and T. F. Abdelzaher, "Klink: Progress-aware scheduling for data stream systems", in *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pp. 485-498, 2021. <https://doi.org/10.1145/3448016.3452794>
- [18] M. Wu, D. Sun, S. Gao, and R. Buyya, "Straggler mitigation via hierarchical scheduling in elastic stream computing systems", *Future Generation Computer Systems*, vol. 166, article 107673, 2025. <https://doi.org/10.1016/j.future.2024.107673>
- [19] S. Liu, Y. Li, and H. Yang, "QAAS: Quick accurate auto-scaling for streaming processing", *Frontiers of Computer Science*, vol. 18, article 181201, 2024. <https://doi.org/10.1007/s11704-022-1706-4>
- [20] P. Matteussi, M. Donato, and S. Bortolotti, "A comparative study of backpressure mechanisms for Apache Spark Streaming", *Sensors*, vol. 22, no. 13, article 4756, 2022. <https://doi.org/10.3390/s22134756>
- [21] M. Schulze *et al.*, "ClickHouse: A fast analytical database for data streams", in *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 3731-3744, 2024. <https://doi.org/10.14778/3685800.3685802>
- [22] S. Horchidan, E. Kritharakis, V. Kalavri, and P. Carbone, "Evaluating model serving strategies over streaming data", in *Proc. ACM Middleware*, 2022. <https://doi.org/10.1145/3533028.3533308>
- [23] T. T. Nguyen *et al.*, "Horizontal pod autoscaling in Kubernetes for elastic container orchestration", *Sensors*, vol. 20, no. 16, article 4621, 2020. <https://doi.org/10.3390/s20164621>
- [24] H. Yuan *et al.*, "A time series-based approach to elastic Kubernetes scaling", *Electronics*, vol. 13, no. 2, article 285, 2024. <https://doi.org/10.3390/electronics13020285>
- [25] M. Niazi *et al.*, "Vertical pod autoscaling in Kubernetes for elastic container collaborative framework", *Computers, Materials & Continua*, vol. 74, no. 1, pp. 591-606, 2023. <https://doi.org/10.32604/cmc.2023.032474>
- [26] Y. Mao *et al.*, "StreamOps: Cloud-native runtime management for streaming services in ByteDance", *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 3501-3514, 2023. <https://doi.org/10.14778/3611540.3611543>
- [27] S. Burckhardt *et al.*, "Netherite: Efficient execution of serverless workflows", in *Proceedings of the VLDB Endowment*, vol. 15, no. 8, 2022. <https://doi.org/10.14778/3529337.3529344>
- [28] K. Saur *et al.*, "Containerized execution of UDFs: An experimental evaluation", in *Proceedings of the VLDB Endowment*, vol. 15, 2022. <https://doi.org/10.14778/3551793.3551860>
- [29] X. Masip-Bruin *et al.*, "Managing the cloud continuum: Lessons learnt from a real fog-to-cloud deployment", *Sensors*, vol. 21, no. 9, article 2794, 2021. <https://doi.org/10.3390/s21092974>
- [30] J. P. Vitorino *et al.*, "IRONEDGE: Stream processing architecture for edge applications", *Algorithms*, vol. 16, no. 2, Article 123, 2023. <https://doi.org/10.3390/a16020123>
- [31] S. N. Srirama, "Distributed edge analytics in edge-fog-cloud continuum", *Internet Technology Letters*, vol. 8, no. 3, article e562, 2024. <https://doi.org/10.1002/itl2.562>
- [32] L. Kong *et al.*, "Edge-computing-driven Internet of Things: A survey", *ACM Computing Surveys*, vol. 55, no. 8, 2023. <https://doi.org/10.1145/3555308>
- [33] A. Al-Dulaimy *et al.*, "The computing continuum: From IoT to the cloud", *Internet of Things*, vol. 27, article 101272, 2024. <https://doi.org/10.1016/j.iot.2024.101272>
- [34] E. Hadzhyiev *et al.*, "Application of Docker Compose for Constructing the Infocommunication System for Online Processing of Astronomical Images", In *Proceedings of the IEEE International Conference on Advanced Computer Information Technologies*, pp. 629-633, 2025. <https://doi.org/10.1109/acit65614.2025.11185586>
- [35] V. Vlasenko *et al.*, "Development of a procedure for fragmenting astronomical frames to accelerate high frequency filtering", *Eastern-European Journal of Enterprise Technologies*, vol. 3, no. 9 (129), pp. 70-77, 2024. <https://doi.org/10.15587/1729-4061.2024.306227>
- [36] Y. Mei *et al.*, "Disaggregated state management in Apache Flink 2.0", in *Proceedings of the VLDB Endowment*, vol. 18, no. 12, 2025. <https://doi.org/10.14778/3750601.3750609>
- [37] S. Khlamov *et al.*, "Comparative analysis of JMeter and Postman for API-based performance testing", *CEUR Workshop Proceedings*, vol. 4048, pp. 426-440, 2025. <https://ceur-ws.org/Vol-4048/paper34.pdf>
- [38] R. Gracia-Tinedo *et al.*, "Practical storage-compute elasticity for stream data processing", in *Proc. of the 24th International Middleware Conference*, pp. 1-7, 2023. <https://doi.org/10.1145/3626562.3626828>