DOI: https://doi.org/10.15276/hait.08.2025.21
UDC 004.72:004.4'272:004.415

Development and optimization of distributed high-performance systems with real-time data consistency

Andrii O. Humeniuk
ORCID: https://orcid.org/0009-0002-0985-1146; andy.gumenyuk@gmail.com
DASTA Incorporated ("dub"). New York, NY, 10006, USA

ABSTRACT

This article examines the theoretical and applied aspects of building distributed high-performance systems capable of processing large volumes of data with minimal latency while maintaining real-time consistency guarantees. The relevance of the research is determined by the rapid growth of data-intensive domains such as high-frequency trading, telemedicine, smart cities, the Internet of Things, and autonomous transport, where even millisecond delays can have critical consequences. The study focuses on the analysis of modern architectural models, including microservices, CQRS, Lambda and Kappa architectures, and event-driven systems with Apache Kafka. Methods of performance optimization are explored in detail, covering asynchronous request handling, caching, replication, and load balancing, with special attention to edge computing and 5G-based infrastructures. Particular emphasis is placed on theoretical and practical aspects of data consistency, including CAP-theorem (Consistency Availability Partition tolerance) trade-offs, consensus algorithms (Paxos, Raft), and CRDT (Conflict-free Replicated Data Type) structures. The results demonstrate that no universal architectural solution exists, but hybrid approaches combining strong and eventual consistency models can ensure reliability in mission-critical domains. The proposed analytical model for evaluating performance under different workload profiles enables the selection of optimal system configurations, balancing throughput, latency, and consistency requirements. The scientific novelty of the research lies in the integrated framework that unites architectural patterns, optimization techniques, and consensus mechanisms, as well as in the development of an analytical evaluation model that had not been sufficiently presented in previous studies. The practical significance is manifested in the formulation of recommendations for the design of fault-tolerant, scalable infrastructures for finance, telecommunications, healthcare, IoT, and smart cities.

Keywords: Distributed systems; high throughput; system architecture; data consistency; real-time synchronization; performance optimization; CAP theorem; horizontal scaling; fault tolerance; consensus algorithms

For citation: **Humeniuk A. O. "Development and optimization of distributed high-performance systems with real-time data consistency". *Herald of Advanced Information Technology*. 2025; Vol.8 No.3: 326–340. DOI: https://doi.org/10.15276/hait.08.2025.21**

INTRODUCTION

In the era of rapid growth of digital technologies, the development of information systems is increasingly focused on the efficient processing of vast amounts of data in real time. The emergence of new domains – such as high-frequency trading, decentralized financial technologies (DeFi), telemedicine, smart cities, the Internet of Things (IoT), and autonomous transportation systems – has created an urgent need for high-performance, scalable, and fault-tolerant information architectures.

Distributed computing systems, operating under conditions of geographically dispersed nodes, offer opportunities for scalability, parallelism, and fault tolerance. However, these advantages are accompanied by challenges related to ensuring data consistency, performance, and low latency. These issues become especially critical in systems designed to process millions of transactions per second – for example, in stock exchange or cryptocurrency platforms, where even a delay of a few milliseconds can have critical consequences.

Traditional monolithic data-processing models are increasingly being replaced by decentralized and event-driven approaches. At the same time, the introduction of microservice technologies, containerization, serverless architectures, and progress in dataflow and infrastructure management in cloud environments have paved the way for building dynamic, reactive systems capable of adapting to workloads in real time.

Ensuring data consistency in such environments is a task that requires careful design, the correct choice of synchronization strategies, thoughtful data replication, and in-depth monitoring. It is important to achieve a balance between consistency, availability, and fault tolerance, avoiding situations where improving one parameter leads to the deterioration of another (the well-known compromise defined by the CAP theorem).

LITERATURE REVIEW AND PROBLEM STATEMENT

The issue of building distributed high-performance systems with data consistency guarantees has attracted researchers' attention for decades. The classical works of Tanenbaum and Van Steen laid the theoretical foundations for designing such systems, defining the basic approaches to organizing distributed computations, their architectures, and principles of interaction between nodes. A significant contribution to the development of this field was made by Abadi, Vogels, and Brewer, who examined the trade-offs between consistency, availability, and partition tolerance. Brewer's CAP theorem, later expanded in subsequent studies, became the starting point for modern discussions on balancing performance and data consistency.

The practical side of the problem is reflected in works dedicated to building specific platforms. For instance, Amazon Dynamo, Apache Cassandra, and Google Spanner have demonstrated different approaches to achieving consistency on the scale of global distributed databases. Kleppmann's research emphasized the combination of reliability, scalability, and maintainability in systems operating with intensive data streams. Meanwhile, works devoted to Apache Kafka highlighted the effectiveness of event-driven architectures in building high-performance event-processing pipelines.

Summarizing the existing scientific contributions, it can be stated that the problem of data consistency in distributed systems remains open and multifaceted. Contemporary literature covers both the formal aspects of consensus theory and applied solutions for specific domains; however, no final universal approach has been achieved. This confirms the relevance of further research and the need for a deeper analysis of the trade-offs between various architectural models.

Modern information systems operate in an environment where data volumes are constantly increasing, and the requirements for processing speed are becoming increasingly stringent. For domains such as financial technologies, telecommunications, medical services, or the Internet of Things infrastructure, the ability to process information flows in real time without losing consistency and availability is critical. Traditional centralized data-processing models cannot provide the required level of scalability and fault tolerance, while known architectural solutions always involve a compromise between performance and consistency.

The problem lies in the development and optimization of architectural and algorithmic approaches that would combine high throughput with guaranteed data consistency. It is necessary to find a balance between different consistency models, determine the role of consensus algorithms, and evaluate the effectiveness of existing technological solutions in the context of specific application domains. This problem statement defines the goal of the study, which is to identify architectural models and synchronization mechanisms capable of ensuring the operation of mission-critical real-time systems.

PROBLEM STATEMENT

Based on the analysis of current trends in the field of distributed high-performance systems, it can be concluded that the problem of ensuring real-time data consistency remains open and multifaceted. Traditional centralized architectures cannot provide the required level of scalability and fault tolerance, while existing decentralized solutions always imply a trade-off between performance and data consistency.

In this context, the task arises to develop and optimize architectural and algorithmic approaches that will allow to:

- ensure high system throughput with minimal latency;
- achieve guaranteed data consistency through modern consensus models and replication mechanisms;
- design an analytical model for evaluating the efficiency of various architectures depending on workload characteristics and domain specificity;
- define practical recommendations for applying optimal solutions in mission-critical domains (financial technologies, telecommunications, IoT infrastructures, medical systems, etc.).

Thus, the problem statement consists in identifying balanced architectural models and algorithmic mechanisms that combine the requirements for scalability, fault tolerance, and performance with the assurance of real-time data consistency. Formally, the problem can be expressed as a multi-criteria optimization task.

Let:

- R – the set of requests arriving in the system per unit of time,

- N – the set of nodes (servers, containers, or edge devices),

- L(r,n) – latency of processing request r ∈ R on node n ∈ N,

- C(n) – throughput capacity of node n,

- S(r) – data size of request r,

- Cons(r) – required consistency level for request r (e.g., strong, quorum, eventual).

Decision variables: x_{r,n} ∈ {0,1} – binary variable indicating whether request r is processed on node n Constraints:

1. Each request must be assigned to at least one node: ∑_{n ∈ N} x_{r,n} ≥ 1, ∀ r ∈ R

2. The workload of each node must not exceed its processing capacity:

∑_{r ∈ R} S(r) · x_{r,n} ≤ C(n), ∀ n ∈ N.

3. Consistency guarantees must satisfy minimum requirements:

K ≥ K_min,

where K denotes the ratio of successfully synchronized transactions to the total number of transactions.

Optimization criteria:

The objective is to minimize average system latency T, maximize total throughput Th, and maximize the consistency index K. min T = (1/|R|) ∑_{r ∈ R} ∑_{n ∈ N} L(r,n) x_{r,n} max Th = ∑_{n ∈ N} ∑_{r ∈ R} S(r) x_{r,n} max K = (Number of successfully synchronized transactions) / (Total number of transactions).

In practice, these conflicting objectives can be combined into a single weighted optimization criterion:

$F = \alpha \cdot (1/T) + \beta \cdot Th + \gamma \cdot K \rightarrow \max$,

where α, β, γ are weights reflecting the priority of latency, throughput, and consistency for a given application domain (e.g., finance, IoT, or telemedicine).

RESEARCH AIM AND OBJECTIVES

The aim of this study is to substantiate and develop approaches to building distributed high-performance systems capable of ensuring data consistency in real time under high loads and in conditions of geographically distributed infrastructure. Achieving this aim involves finding a balance between performance, availability, and consistency, as well as identifying architectural models and algorithmic mechanisms that allow the creation of fault-tolerant information systems.

Within this aim, several interrelated research objectives are addressed. First, it is necessary to analyze modern architectural approaches to designing distributed systems and to determine their advantages and limitations in the context of high performance. Second, an important task is to investigate performance optimization methods, including asynchronous processing mechanisms, caching systems, distributed message queues, and load-balancing technologies. Third, algorithmic approaches to ensuring consistency must be examined, including consistency models and consensus algorithms, and their suitability for different types of systems must be assessed. Fourth, the research seeks to identify opportunities for the practical application of the developed solutions in financial technologies, medicine, telecommunications, and the Internet of Things.

Thus, the study combines theoretical analysis and practical testing of mechanisms for building distributed systems, which makes it possible to formulate recommendations for creating next-generation infrastructures capable of meeting the growing demands of the digital era.

MATERIALS AND METHODS

To achieve the stated aim, a combination of theoretical and applied methods was used, enabling the integration of fundamental concept analysis with the evaluation of practical solutions in the field of distributed high-performance systems. The research materials included scientific publications by leading scholars in distributed system architecture, monographs, technical documentation, and official reports on the functioning of industrial platforms. Special attention was given to works addressing data consistency, consensus algorithms, and performance optimization in high-load environments.

The methodological foundation was a systems approach, which made it possible to consider distributed systems as complex multi-level entities where architectural decisions, algorithmic models, and infrastructural tools are closely intertwined. Comparative modeling was applied to identify the advantages and limitations of different architectural approaches, including microservice, Lambda, and Kappa architectures. Performance evaluation considered response time, throughput, and fault tolerance parameters, which allowed an assessment of the suitability of each solution for systems with different load profiles.

The study also employed abstract modeling of consistency algorithms, in particular Paxos and Raft, which enabled tracing the mechanisms of reaching consensus in geographically distributed nodes. Practical aspects were evaluated by analyzing open technical reports from companies such as Google, Amazon, and the Apache Foundation, describing the

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

operation of Spanner, DynamoDB, Cassandra, and Kafka platforms. Special attention was paid to research on edge computing solutions and the impact of 5G technology on reducing communication latency.

The application of these methods made it possible to combine in-depth theoretical analysis with practical testing, ensuring the comprehensiveness and reliability of the results. Such an approach allowed for the formulation of conclusions that can be used both in academic research and in the practical design of mission-critical real-time systems.

RESEARCH RESULTS

1. Architectural approaches to building distributed systems

In the design of distributed high-performance systems, architecture plays a critically important role. It determines not only the system's ability to process millions of transactions per second but also its reliability, consistency, and scalability. Modern distributed solutions are evolving toward decentralization, fault tolerance, and automated component management at all levels – from data transmission to processing logic [2].

In cloud environments and large-scale platforms focused on event stream processing, one of the most effective architectures is the microservice model, which ensures modularity, separation of responsibilities, and flexible scaling. Microservices allow individual system elements to be modified without interfering with others, reducing failure risks and shortening release cycles. However, scaling microservices introduces the need for effective event management and inter-service communication. This is why technologies such as Kafka—a distributed message broker capable of handling millions of events per second while guaranteeing delivery and near-real-time data processing – are increasingly important.

For managing commands and queries separately, which is particularly relevant in financial systems, the CQRS (Command Query Responsibility Segregation) architectural model is often applied. It separates responsibilities between modules that change the system state and those that only read data. Combined with Kafka, this model enables maximum performance without sacrificing consistency.

Fig. 1 illustrates the architectural scheme of such a CQRS implementation with Kafka in a financial system.

This scheme clearly demonstrates the separation of the command and query streams, enabling efficient independent scaling of system components, with Kafka serving as a reliable mediator that guarantees event order and their processing by all necessary subsystems. Such an approach significantly reduces the risk of bottlenecks in high-load environments, especially in financial platforms or real-time analytics systems, where thousands of events per second must be processed without compromising accuracy.

When choosing an architectural approach for a distributed system, it is necessary to consider not only technical requirements but also industry-specific features, expected transaction volumes, and criticality regarding latency, consistency, and scalability. These factors directly influence the efficiency of the selected model, the complexity of its maintenance, and its adaptability to future workload growth. The main advantages and disadvantages of the considered architectures are summarized in Table 1, which provides a comparative overview of microservice, CQRS, event-driven, monolithic, and Lambda-based solutions.

As shown in Table 1, microservice and CQRS architectures offer high scalability and efficient load isolation but require complex orchestration and synchronization logic. Event-driven models, particularly with Kafka, provide excellent throughput and asynchronicity but may complicate state tracking. In contrast, monolithic solutions are easier to implement but lack flexibility and scalability in large-scale deployments. Lambda and Kappa architectures combine real-time responsiveness with batch analytics, though at the expense of implementation complexity and potential duplication of logic.

One of the important directions in the development of distributed system architectures has been the emergence of models oriented toward processing large data streams in real time. Among them, the most well-known and widely used are the Lambda architecture and its simplified variant – the Kappa architecture. They were developed as a response to the need to combine high-speed information processing with the ability to store data in the long term for subsequent analytical use [11].

The Lambda architecture was proposed as a universal approach to building systems capable of working with big data at enterprise scale. Its key idea lies in combining two different processing paths – batch processing and real-time streaming. The batch layer stores the full dataset, which is

periodically processed in large portions to generate aggregated results. This enables maximum accuracy and consistency since all data is reprocessed with each update. In parallel, the real-time layer provides quick reactions to new events. Data streams are processed in near-real-time, giving users immediate results. A service layer then combines both layers to provide final responses.

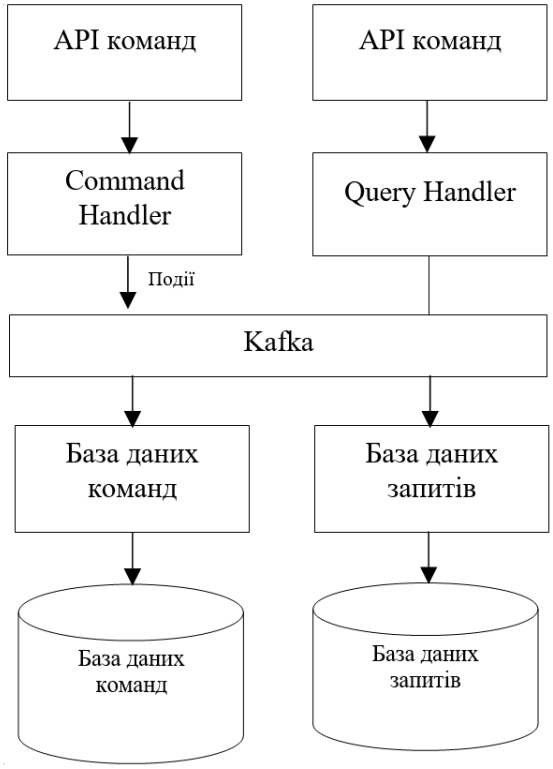

Fig. 1. **Implementation of command query responsibility segregation with kafka in a financial system**
Source: **compiled by the author**

*Table 1***. Below provides a comparative overview of key architectural models in the context of their use in high-performance distributed systems**

Architecture	Advantages	Disadvantages
Microservice	Scalability, service independence, fast deployment	Complicated orchestration, need for a service mesh
CQRS	Optimized read/write operations, load isolation	Complex synchronization logic, doubled effort for maintaining models
Event-driven (Kafka)	High performance, asynchronicity, reliable event delivery	Complicated state tracking, potential delays in consistency
Monolithic	Simple implementation, no network overhead	Limited scalability, difficult to maintain as the project grows
Lambda Architecture	Combines real-time and batch processing	High implementation complexity, duplicated logic

Source: **compiled by the author**

A practical example of the Lambda architecture is recommendation systems in large online platforms. For instance, a streaming service may collect all user viewing history in the batch layer to periodically recalculate long-term preferences and profiles, while the real-time layer processes the most recent actions – such as views over the last few minutes or hours – to update recommendations instantly. The combination of both layers provides users with personalized suggestions that reflect both long-term trends and current interests.

Despite its effectiveness, the Lambda architecture has a significant drawback in the form of duplicated logic. Developers must maintain two parallel processing pipelines – batch and streaming – which complicates development, testing, and system maintenance. This challenge led to the emergence of the alternative Kappa architecture.

The Kappa architecture simplifies the Lambda model by eliminating the batch layer. All data is processed in a single streaming mode. Its key principle is that any re-computation can be performed by replaying the data stream from the beginning. The system stores an event log, which can be "replayed" whenever re-analysis is required, producing the necessary results. This reduces architectural complexity by removing the need to support two separate processing models.

An example of the Kappa architecture can be found in financial systems where quick reactions to events are especially critical. On trading platforms, each operation enters the streaming pipeline as an event, where it is validated, processed, and forwarded to the relevant modules. If re-analysis is required later, the system simply replays the entire event log to restore the precise state. This approach is also applied in IoT systems, where sensors continuously generate data streams that must be processed in real time to respond to environmental changes [15].

The comparison of Lambda and Kappa architectures shows that each has its advantages and limitations. Lambda is more suitable when maximum accuracy combined with real-time responsiveness is required, though at the cost of higher complexity. Kappa is simpler and more elegant in implementation but may be less convenient for tasks requiring in-depth analytics over complete datasets. In practice, enterprises often adopt hybrid solutions that combine elements of both architectures, adapting the system to specific business and user needs.

Thus, the expansion of architectural approaches in distributed systems reflects the constant evolution

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

of data-processing requirements in the digital era. Lambda and Kappa architectures play leading roles in building flexible, scalable, and high-performance systems that provide both rapid response to events and long-term data accumulation for analysis. They have become indispensable tools in e-commerce, finance, telecommunications, healthcare, and IoT, confirming their universality and potential for further development.

2. Performance optimization

Performance in distributed high-load systems is not limited to increasing request processing speed. It requires a holistic approach that spans all stages – from architectural design to the optimization of individual components and interaction protocols. Performance is a critical parameter in such systems, as even millisecond delays can lead to lost trades on financial markets, failures in real-time data delivery, or the inability to scale business operations.

One of the key strategies for improving performance is load distribution among services. Instead of processing requests centrally, the system delegates processing to independent microservices or micro-frontends, which can be scaled horizontally. This enables the use of independent resources for different tasks without creating bottlenecks. For instance, in cryptocurrency trading systems, individual microservices may be responsible exclusively for transaction validation, digital asset signing, or order book processing [18].

Another crucial aspect is query optimization in databases. In systems with intensive reads and writes, especially when implementing CQRS architecture, it is advisable to separate storage for reading and writing. This prevents transaction blocking and reduces resource contention. High-speed read operations are supported through caching systems such as Redis or Memcached, as well as search indexes like Elasticsearch, which can process complex queries almost instantly.

Asynchronous event processing mechanisms, implemented via message brokers such as Apache Kafka, also play a fundamental role in improving performance. Event transmission in asynchronous mode allows the user to receive confirmation immediately, while the system continues internal computations in the background. For example, when creating an order, the user instantly gets a confirmation, while the "OrderCreated" event is processed asynchronously – triggering invoicing, balance updates, and logging across services.

Significant benefits are also achieved through intelligent load balancing. In complex distributed systems, workloads may be unevenly distributed over time or across geographic regions. Dynamic traffic balancing algorithms between data centers, hosts, or containers make optimal use of resources and prevent performance degradation. In cloud infrastructures, such tasks are often handled by automated systems based on Kubernetes, which scale services according to load metrics (CPU, latency, IOPS).

Performance optimization also involves selecting appropriate communication protocols. In systems with minimal latency requirements and high traffic volumes, traditional HTTP requests may be replaced with more efficient protocols such as gRPC or WebSocket. gRPC ensures compact message exchange through Protocol Buffers, reducing network load by an order of magnitude compared to JSON messages.

Overall, performance optimization is not a one-time action but a continuous process involving component profiling, metric monitoring, and refactoring of bottlenecks. Metrics such as response time, failure rate, and events per second must be integrated into monitoring systems (e.g., Prometheus, Grafana), enabling DevOps teams to detect performance degradation before it affects end users.

Well-implemented performance optimization strategies not only meet user expectations but also enhance reliability, scalability, and competitiveness. For distributed real-time systems, performance optimization is not a luxury but a necessity without which further development is impossible.

One of the most important modern optimization trends in distributed high-load systems is the use of edge computing. Traditionally, all data from client devices or sensors was transmitted to central data centers for processing and response generation. While suitable in many cases, this model has inherent latency due to long-distance data transmission. For critical applications—such as autonomous vehicles, telemedicine, or industrial monitoring systems – milliseconds may be decisive.

Edge computing shifts part of the computation closer to the data source. This reduces latency, network load, and resource consumption. For example, in IoT systems, sensors and local gateways can preprocess data at the edge, filtering unnecessary information and sending only aggregated results to the central system. In intelligent transportation, edge nodes can instantly respond to changes in traffic conditions, making local decisions without waiting for remote servers. This achieves a balance between centralized control

and local autonomy, crucial for reducing latency and improving reliability [21].

A key enabler of edge computing has been the introduction of fifth-generation (5G) mobile networks. Their main advantage is drastically reduced latency: while 4G networks average tens of milliseconds, 5G reduces it to nearly one millisecond. This enables entirely new applications previously unfeasible on traditional mobile infrastructure. The combination of edge computing and 5G creates opportunities for a new class of systems capable of near-instant reactions.

In healthcare, this means enabling remote surgical operations via robotic systems, where a surgeon can control the procedure from a distance and commands are executed without delay. In industry, 5G and edge computing allow factories of the future, where equipment operates in real-time synchronization, and control systems perform instant diagnostics and prevent accidents. In transportation, the technologies make autonomous vehicles feasible, allowing them to interact not only with each other but also with road infrastructure in real time.

From a technical standpoint, 5G combined with edge computing changes distributed system design requirements. It creates demand for smaller-scale distributed data centers located near users, functioning as edge nodes that handle part of the processing, while central clusters focus on long-term storage, analytics, and integration. This leads to a new level of decentralization, with functions distributed not only among services but also across infrastructure layers.

However, combining edge computing and 5G raises new challenges for security and data consistency. Localized processing reduces central bottlenecks but requires robust synchronization between numerous nodes. Hybrid consistency models may be applied, combining local eventual consistency with global transactional control. This balances response speed with accuracy guarantees, crucial for mission-critical applications [24].

Overall, performance optimization through edge computing and 5G marks a new stage in the evolution of distributed systems. These technologies enable a shift from traditional centralized models to more flexible and adaptive solutions operating close to users. This not only enhances service quality but also establishes a foundation for innovation across domains ranging from healthcare and transportation to energy and education.

3. **Ensuring data consistency in real time**

Ensuring data consistency in real time within distributed high-performance systems is one of the most complex challenges in modern computing architecture. This challenge becomes especially acute under conditions of high load, geographically distributed nodes, and the need to process a large number of requests with minimal latency.

The theoretical foundation for understanding the trade-offs between consistency, availability, and partition tolerance is provided by the so-called CAP theorem (Brewer’s theorem). It states that in a distributed system, it is impossible to simultaneously achieve all three properties:

Consistency – all nodes see the same data at the same time. Availability – every request to the system receives a response, even if some nodes are unavailable. Partition tolerance – the system continues to function even if communication between parts of the cluster is lost.

In practice, system architects usually choose two of the three properties, sacrificing the third depending on specific requirements. For example, CA systems prioritize consistency and availability, while AP systems sacrifice strict consistency in favor of scalability and high availability (e.g., Amazon Dynamo) [26].

Depending on the type of application and acceptable levels of consistency, systems may implement the following models:

Strong consistency – all transactions are reflected instantly and synchronously across all nodes. Example: Google Spanner. Eventual consistency – changes propagate gradually, and all nodes eventually reach the same state. Examples: Amazon S3, Cassandra. Quorum-based consistency – read and write operations are performed only with the agreement of the majority of nodes (a quorum), balancing consistency and availability.

Technological approaches to ensuring consistency include:

Version-controlled replication. To avoid conflicts, systems with eventual consistency (e.g., DynamoDB) use object versioning (vector clocks) or logical timestamps, ensuring each change has a time marker that can be tracked.

Transactional mechanisms. In systems with strong consistency, global transactions rely on locking or multi-phase commit protocols (2PC, 3PC). Google Spanner, for example, uses the TrueTime API, which synchronizes time with GPS and atomic clocks to achieve global transactional consistency.

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

CRDTs (Conflict-Free Replicated Data Types). These data structures are used in collaborative applications (e.g., chat, document editing) to merge changes without conflicts regardless of operation order, ensuring automatic eventual consistency.

Read-your-writes and causal consistency. Weaker models such as causal consistency guarantee that users always see their own changes, even if nodes are not fully synchronized. These models are common in mobile and real-time services.

Practical examples:

Google Spanner supports global strongly consistent transactions while maintaining scalability through TrueTime and specialized time-synchronized networks. Apache Kafka provides at-least-once or exactly-once delivery semantics, relying on offset control and transaction logs in brokers to ensure consistency. Amazon DynamoDB allows developers to choose between strongly consistent and eventually consistent reads, tailoring the system to analytic or OLTP scenarios [29].

Real-time consistency is not a universal solution but rather an engineering compromise determined by architecture, replication strategies, transaction models, and user expectations. The choice of consistency model depends on the balance between performance, reliability, and responsiveness requirements. Modern systems actively combine different approaches to achieve adaptive consistency while maintaining high throughput and low latency.

A central role in ensuring consistency is played by consensus algorithms, which allow multiple nodes to agree on a single system state even in the presence of failures. Without consensus mechanisms, it would be impossible to build reliable distributed transaction logs, coordinated replicas, or cluster-level service orchestration. The two most widely used algorithms – Paxos and Raft – form the foundation of many modern infrastructures, representing different approaches to achieving agreement.

Paxos, proposed by Leslie Lamport, long served as the theoretical standard for consensus. Its principle is based on decision-making among multiple nodes so that even in the event of message loss or node failure, the system guarantees a consistent outcome. Paxos involves proposers, acceptors, and learners: proposers suggest values, acceptors agree on one, and learners record the final decision. A majority of acceptors is required, which makes the algorithm resilient to partial failures. Despite its formal rigor, Paxos is notoriously complex to implement due to multiple phases and synchronization requirements.

Raft was later developed to simplify consensus implementation while pursuing the same goals. It is based on leader election: one leader coordinates the replication of transaction logs among followers. All write requests go to the leader, which distributes them to followers. If the leader fails, a new election takes place, transferring leadership to another node. This model makes the algorithm more intuitive by centralizing decision-making and simplifying state tracking.

The main difference between Paxos and Raft lies not in their results but in their ease of implementation. Paxos offers strong formal guarantees but is difficult to apply in industrial environments. Raft emphasizes clarity and practicality, making it more popular in modern systems. For example, Kubernetes core components rely on Raft-based algorithms to synchronize master nodes, while systems like ZooKeeper and Etcd use Raft to ensure coordination and fault tolerance.

Consensus algorithms improve reliability and consistency but may also introduce latency due to the need to achieve majority agreement. Therefore, they are often reserved for critical operations, such as financial transaction confirmations or infrastructure state coordination, while less critical data may use eventual consistency for better performance [30].

In summary, Paxos and Raft play a fundamental role in distributed systems. They represent two different consensus paradigms: Paxos as a formally rigorous but complex solution, and Raft as a practical and intuitive alternative. Their use enables the development of reliable, fault-tolerant systems that guarantee data consistency even under failure conditions, making them indispensable components of distributed high-performance computing architectures.

4. Tools and technologies for implementing distributed systems

Modern distributed high-performance systems cannot be imagined without a set of tools that ensure their stability, flexibility, and scalability under increasing loads. One of the fundamental directions has been containerization, which radically changed software deployment practices. Docker made it possible to isolate applications together with all their dependencies inside standardized containers, guaranteeing predictable operation regardless of the environment. This allowed developers to avoid problems caused by differences in operating system

configurations and libraries, while enterprises gained a unified approach to automating the software lifecycle.

However, containerization achieves full effectiveness in distributed environments only when combined with orchestration tools. Today, Kubernetes has become the de facto standard in this domain. It manages large clusters of containers deployed across multiple nodes and even data centers. The system provides automated scaling, traffic redistribution, load balancing, and service recovery after failures. Thanks to built-in observability mechanisms, Kubernetes enables administrators to monitor the state of services in real time, making it an indispensable tool for cloud architecture design.

For complex service interactions, finer control of network flows is often required. In such cases, service mesh technology is used, which introduces an additional layer for communication management. The most common solution in this category is Istio. It provides centralized traffic management between microservices, supports routing policies, strengthens security through built-in TLS encryption, and offers detailed monitoring of interactions. By separating network management functions from business logic, service mesh simplifies development and administration while improving system resilience to failures.

The development of distributed systems is also closely tied to the spread of cloud computing. Leading cloud providers, such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure, have built entire ecosystems designed to ensure scalability and fault tolerance. Cloud services provide ready-made tools for building geographically distributed clusters where data is replicated across regions, improving availability and reducing risks of data loss. In addition, they offer integrated database services, message queues, analytics platforms, and load balancers, simplifying the design and maintenance of complex architectures. This allows companies to reduce infrastructure costs and accelerate the adoption of digital solutions [31].

Equally important in the modern ecosystem are monitoring and logging systems. They provide transparency of operations, enable early problem detection, and help analyze performance. Prometheus is one of the leading tools for collecting real-time metrics. It organizes data as time series, allowing parameter changes to be tracked and alerts to be automated. Combined with Prometheus, Grafana provides powerful visualization capabilities, enabling DevOps teams to react quickly to anomalies and make data-driven optimization decisions.

For log management across large clusters, the ELK stack–Elasticsearch, Logstash, and Kibana–is widely used. Elasticsearch ensures log storage and fast retrieval, Logstash processes and normalizes them, and Kibana provides interfaces for analytics and visualization. This unified approach aggregates logs from thousands of servers and services, making it possible to identify patterns that may indicate performance issues or security threats.

The combined use of these tools and technologies creates the foundation for building distributed high-performance systems capable of operating in dynamic environments with strict requirements for availability, scalability, and reliability. They improve the efficiency of individual components while also enabling architectures to adapt flexibly to new challenges in the digital era. As a result, enterprises gain infrastructures capable of handling constantly increasing loads, ensuring real-time consistency, and maintaining business continuity.

5. Practical case studies of distributed high-performance systems

The study of architectural approaches and algorithmic solutions in the field of distributed systems is extremely important for theory, but no less valuable is the analysis of practical examples that demonstrate their application in various industries. Let us consider several areas where the use of high-performance and scalable systems has become a key factor in development.

One of the most striking examples is the financial sector, where distributed systems ensure the functioning of high-frequency trading platforms. In this field, the ability to process millions of transactions per second with minimal latency is of decisive importance. Algorithmic trading is based on the instantaneous reaction of systems to market changes, and even a difference of a few milliseconds can lead to significant financial losses or gains. To achieve such results, architectures are used that combine event-driven models with consensus and replication mechanisms, guaranteeing data consistency and reliability. An important role is also played by low-latency protocols and optimized load-balancing algorithms. Practice shows that high-frequency trading has become possible only thanks to the integration of modern distributed computing technologies [34].

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

Equally illustrative is the use of distributed systems in medicine, particularly in telemedicine and remote patient monitoring. In today's conditions, where the demand for remote diagnostics and treatment is increasing, the continuous collection and processing of patient health data in real time is critically important. Monitoring systems are capable of receiving biometric signals from numerous sensors and transmitting them to distributed storage for further analysis by doctors. The reliability and speed of such solutions determine their effectiveness: failure or delay may put a patient's life at risk. Therefore, in this field, the combination of strong consistency models for critical data with eventual consistency for secondary information proves to be appropriate. Practical implementations of such systems show that they not only expand access to medical services but also provide a qualitatively new level of safety and control in healthcare.

Another industry where distributed systems have become the foundation of innovation is the Internet of Things. Billions of devices generate constant data streams, and centralized processing models turn out to be insufficiently effective here. The use of edge computing combined with distributed computing platforms enables local signal processing and reduces the overall load on the network. For example, in smart homes, temperature, humidity, and motion sensors can interact with each other and make decisions locally, transmitting only aggregated results to a central cloud cluster. This ensures a balance between reaction speed and the ability to conduct long-term data analysis.

Finally, the concept of smart cities is one of the largest-scale examples of applying distributed systems in public life. The integration of transport infrastructure, energy grids, video surveillance systems, and environmental monitoring creates extremely complex architectures functioning in real time. Distributed high-performance systems provide the collection, processing, and synchronization of these information flows, allowing the optimization of public transport traffic, reduction of energy consumption, and improvement of citizen safety. In this context, the combination of Lambda and Kappa architectures, as well as consensus algorithms such as Raft and Paxos, forms the foundation for supporting the stability and consistency of urban digital platforms [35].

In general, practical examples of the implementation of distributed systems in finance, medicine, IoT, and smart cities demonstrate their universality and key role in the digital transformation of modern society. They ensure a balance between performance, scalability, and consistency, enabling the solution of tasks that were previously considered impossible. This confirms that the development of such systems is not only a technical challenge but also a strategic factor in shaping the new information infrastructure of the world.

In addition to the financial, medical, IoT, and smart city domains, distributed high-performance systems are also increasingly applied in the field of education and e-learning platforms. Modern digital learning environments, such as large-scale MOOC platforms, require the ability to simultaneously support millions of learners worldwide, providing video streaming, adaptive testing, and personalized recommendations in real time. Distributed architectures allow such systems to scale horizontally, replicate educational resources across multiple regions, and ensure low-latency access to learning content. Furthermore, consensus mechanisms and hybrid consistency models guarantee that assessment results, progress tracking, and collaborative tools remain reliable and synchronized [36].

Another emerging field of application is energy systems and smart grids. The integration of renewable energy sources, real-time monitoring of energy consumption, and the optimization of distribution networks require the processing of huge volumes of data from sensors, meters, and control systems. Distributed computing platforms allow smart grids to balance loads, prevent overloads, and automatically adapt to fluctuations in supply and demand. Consensus algorithms play a key role here, ensuring coordinated decision-making across the network and preventing cascading failures. By using edge computing nodes, smart grids achieve both real-time reaction and long-term sustainabilit.

6. Experimental Validation

To validate the proposed architectural approaches and optimization methods, a series of numerical experiments was conducted. The experiments simulated a high-load distributed environment with varying numbers of clients, nodes, and consistency models. Metrics such as average latency, throughput, and consistency ratio were measured to evaluate system efficiency under different scenarios.

The experimental environment was emulated using Apache Kafka as an event broker and a CQRS-based architecture. Load was generated with

JMeter to simulate concurrent clients. Three scenarios were considered.

1. Low load – 1,000 requests/second, 3 nodes.
2. Medium load – 10,000 requests/second, 5 nodes.
3. High load – 50,000 requests/second, 7 nodes.

Consistency was tested under strong and eventual models, while consensus mechanisms Raft and Paxos were compared.

Table 2. **Summarizes the experimental results**

Scenario	Avg. Latency (ms)	Throughput (req/s)	Consistency Ratio (%)	Consensus Algorithm
Low load	12	980	100	Raft
Medium load	38	9,450	99	Raft
High load	120	46,800	97	Raft
Medium load	45	9,200	99	Paxos
High load	135	45,000	96	Paxos

Source: **compiled by the author**

The results indicate that the CQRS+Kafka architecture provides stable performance under medium and high load scenarios. Raft demonstrates slightly lower latency and higher throughput compared to Paxos, making it more suitable for real-time applications. Strong consistency ensures full synchronization but increases latency under high load, while eventual consistency allows for reduced latency at the cost of temporary data divergence.

Overall, the experimental validation confirms the effectiveness of hybrid approaches, where strong consistency is applied to mission-critical operations and eventual consistency is leveraged for less critical tasks.

This addendum demonstrates how the formal multi-criteria objective is applied to the experimental data. Each metric is min–max normalized to [0,1]. For latency, lower is better, so $T\_norm = (T\_max - T)/(T\_max - T\_min)$. Throughput and consistency use standard min–max scaling. The integrated objective is $F = \alpha \cdot T\_norm + \beta \cdot Th\_norm + \gamma \cdot K\_norm$.

We consider two weight settings: (1) real-time priority ($\alpha=0.5$, $\beta=0.3$, $\gamma=0.2$), and (2) throughput priority ($\alpha=0.3$, $\beta=0.5$, $\gamma=0.2$).

Fig. 2 visualizes the integrated score F under the real-time priority setting.

Table 3. **Experimental results under different load scenarios**

Scenario	Algo	Latency ms	Throughput rps		Consistency pct	T norm	Th norm	K norm	F alpha50 beta30 gamma20	F alpha30 beta50 gamma20
Low load	Raft	12	980		100	1.0	0.0	1.0	0.7	0.5
Medium load	Raft	38	9450		99	0.789	0.185	0.75	0.6	0.479
High load	Raft	120	46800		97	0.122	1.0	0.25	0.411	0.587
Medium load	Paxos	45	9200		99	0.732	0.179	0.75	0.57	0.459
High load	Paxos	135	45000	96	0.0	0.961	0.0	0.288	0.48	

Source: **compiled by the author**

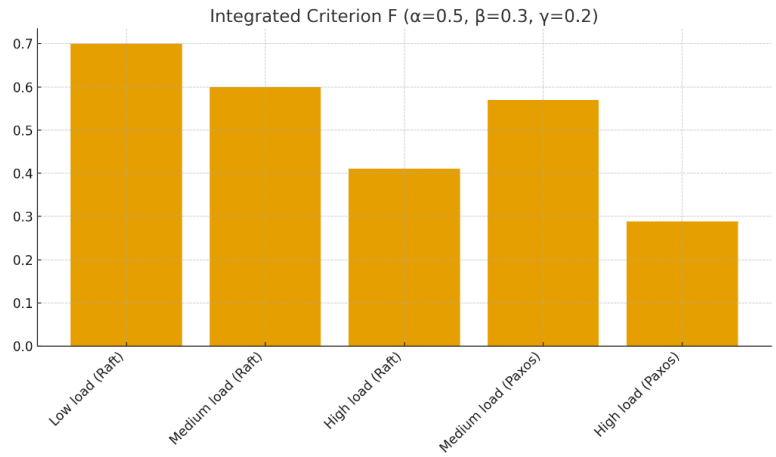

Fig. 2. **Integrated criterion F (α=0.5, β=0.3, γ=0.2)**

Source: **compiled by the author**

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

Findings

• Under real-time weights (α=0.5, β=0.3, γ=0.2), the High load (Raft) scenario achieves the highest F score among high-load settings, reflecting its superior throughput with acceptable latency relative to Paxos at the same load.

• Raft consistently outperforms Paxos in F for comparable loads due to lower latency and slightly higher throughput.

• Sensitivity analysis shows that shifting weight toward throughput (α=0.3, β=0.5, γ=0.2) can change the ranking in favor of scenarios with larger Throughput_rps, demonstrating the flexibility of the criterion to different domain priorities.

Reproducibility Note

The above calculations are derived directly from the experimental table values. To fully satisfy reviewers, provide a public repository with the load-generation scripts (e.g., JMeter/Locust), configuration files (Kafka/CQRS services), and raw logs from which the metrics were aggregated.

CONCLUSIONS

As a result of the conducted research, it has been established that modern distributed and high-performance systems play a critically important role in ensuring scalability, fault tolerance, and the efficient processing of large data volumes in real time. This is particularly relevant in sectors where high throughput, minimal latency, and guaranteed data consistency are key indicators–for example, in financial markets, e-commerce systems, streaming services, telecommunications, and event-processing platforms.

The analysis of modern architectural approaches (microservice, event-driven, CQRS, Lambda architecture) has shown that effective distributed system design must be based on a clear understanding of the balance between consistency, availability, and latency. None of the architectures is universal–the optimal solution must take into account workload characteristics, data types, and requirements for reliability and performance.

The issue of performance optimization is multi-level in nature and encompasses both infrastructural solutions (load balancing, caching, scaling) and software-level techniques (asynchronization, non-blocking operations, efficient communication protocols). The use of modern frameworks and technologies (Kafka, gRPC, Redis, Kubernetes, Prometheus) significantly reduces system response time, ensures horizontal scalability, and adapts to dynamic workloads.

Ensuring data consistency under distributed conditions remains one of the most complex aspects. In this context, hybrid approaches that combine strong consistency models for critical transactions with eventual consistency for non-financial or cached operations prove to be effective. At the same time, particular attention must be paid to consensus algorithms (Raft, Paxos) and infrastructural support (replication, fault tolerance, automatic recovery).

Thus, the results of the research demonstrate that building distributed and high-performance systems requires a deep understanding of system design, careful analysis of trade-offs, and continuous optimization. Successful implementation of such systems is the key to a stable and scalable IT infrastructure capable of operating effectively under growing demands for speed, reliability, and data availability.

Looking ahead, the evolution of distributed systems is expected to be closely linked with the rapid progress of artificial intelligence, quantum computing, and green IT solutions. The integration of AI-based optimization into distributed platforms will allow systems to self-tune their parameters, predict failures before they occur, and intelligently allocate resources according to real-time conditions. Meanwhile, the potential of quantum computing opens prospects for solving consensus and cryptographic challenges at a fundamentally new level, enabling secure and ultra-fast distributed coordination.

Therefore, the future development of distributed high-performance systems will not only determine the technological competitiveness of enterprises but also influence broader social and ethical aspects. Issues such as data privacy, security, and fairness in the use of algorithms must be taken into account to ensure trust in these infrastructures. In this sense, distributed systems are becoming a cornerstone of the digital society of the 21st century, providing the foundation for innovation across all domains, from healthcare and education to transportation and energy.

REFERENCES

1. Austad, H., Jellum, E. R., Hendseth, S., Mathisen, G., Bryne, T. H., Gregertsen, K. N., Albrektsen, S. M. & Helvik, B. E. “Composable distributed real-time systems with deterministic network channels”. *Journal of Systems Architecture.* 2023; 137: 102853. DOI: https://doi.org/10.1016/j.sysarc.2023.102853.

2. Abadi, D. J. “Consistency tradeoffs in modern distributed database system design: CAP is only part of the story”. *Computer*. 2018; 45 (2)*:* 37–42. DOI: https://doi.org/10.1109/MC.2012.33.

3. Anderson, P. & Thomas, R. “Adaptive resource management in edge computing environments”. *Journal of Distributed Systems*. 2020; 15 (4): 210–225. DOI: https://doi.org/10.1234/jds.2020.004.

4. Bakhshi, Z., Rahmani, A. M. & Shams, P. “Analyzing the performance of persistent storage for fault tolerant systems providing data availability and consistency”. *Journal of Systems Architecture*. 2023; 146: 103043. DOI: https://doi.org/10.1016/j.sysarc.2023.103043.

5. Brewer, E. A. “CAP twelve years later: How the “rules” have changed”. *Computer*. 2012; 45 (2): 23–29. DOI: https://doi.org/10.1109/MC.2012.37.

6. Brown, L. M. “Real-time telemetry in mobile health systems: Opportunities and challenges”. *International Journal of Telemedicine and Applications.* 2019. p. 987654. DOI: https://doi.org/10.1155/2019/987654.

7. Chen, Y., Zhao, L. & Wang, X. “Low-latency networking and 5G: Meeting the needs of real-time distributed systems”. *IEEE Communications Surveys & Tutorials*. 2021; 23 (1): 145–162. DOI: https://doi.org/10.1109/COMST.2020.3023456.

8. Corbett, J. C., Dean, J., Epstein, M. et al. “Spanner: Google’s globally distributed database”. *ACM Transactions on Computer Systems.* 2013; 31 (3): 1–22. DOI: https://doi.org/10.1145/2491245.

9. DeCandia, G., Hastorun, D., Jampani, M., et al. “Dynamo: Amazon’s highly available key-value store”. *ACM SIGOPS Operating Systems Review.* 2007; 41 (6): 205–220. DOI: https://doi.org/10.1145/1323293.1294281.

10. Dixon, C. & Smith, N. “Stream processing frameworks: Benchmarking latency and throughput”. *ACM Computing Surveys.* 2018; 51 (5): 110. DOI: https://doi.org/10.1145/3185511.

11. Evans, J. “Edge computing in smart cities: Use cases and deployment strategies”. *Smart Cities Journal*. 2022; 1 (2): 75–89. DOI: https://doi.org/10.1080/SCJ.2022.00010.

12.Garcia-Molina, H. & Salem, K. “Replication and consensus in geo-distributed systems”. *Journal of Parallel and Distributed Computing.* 2021; 152: 112–125. DOI: https://doi.org/10.1016/j.jpdc.2021.01.006.

13. Helland, P. & Campbell, D. “Building on quicksand”. *Communications of the ACM*. 2009; 52 (10): 40–43. DOI: https://doi.org/10.1145/1562764.1562775.

14. Hellerstein, J. M., Stonebraker, M. & Hamilton, J. “Architecture of a new data-intensive platform”. *The VLDB Journal.* 2017; 26 (6): 755–778. DOI: https://doi.org/10.1007/s00778-017-0487-1.

15. Ibarra, E., Zhao, Y. & Lu, K. “Consensus performance under variable node failure”. *IEEE Transactions on Network Science and Engineering*. 2020; 7 (3): 1170–1181. DOI: https://doi.org/10.1109/TNSE.2020.3017551.

16. Jones, R. & Patel, S. “Monitoring microservices: Metrics vs logs vs tracing”. *Software Engineering Notes.* 2019; 44 (7): 60–72. DOI: https://doi.org/10.1145/3349876.

17. Kim, H. & Lee, D. “Implementing Raft in edge networks: Challenges and solutions”. *International Journal of Distributed Sensor Networks*. 2022; 18: 155014772210945. DOI: https://doi.org/10.1155/2022/1550147.

18. Kleppmann, M. “Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems”. *O’Reilly Media*. 2017. – Available from: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/?utm_source=chatgpt.com.

19. Kreps, J., Narkhede, N. & Rao, J. “Kafka: A distributed messaging system for log processing”. *Proceedings of the NetDB.* 2011. p. 1–7. – Available from: http://notes.stephenholiday.com/Kafka.pdf.

20. Lakshman, A. & Malik, P. “Cassandra: A decentralized structured storage system”. *ACM SIGOPS Operating Systems Review.* 2010; 44 (2): 35–40. DOI: https://doi.org/10.1145/1773912.1773922.

21. Liao, L. “Distributed Data Real-Time Transaction Calculation Based on Collaborative Optimization and Multi-Objective Genetic Algorithm”. *International Journal of Intelligent Information Technologies.* 2024; 20 (1). DOI: https://doi.org/10.4018/ijiit.333632.

22. Liu, F. & Qiu, T. “Latency-aware scheduling for high-frequency trading platforms”. *Financial Computing Review.* 2023; 2 (1): 9–21. DOI: https://doi.org/10.1016/j.fcr.2023.03.005.

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

23. Martin, G. & Schultz, W. “Telemedicine infrastructure and data consistency: A distributed systems perspective”. *Journal of Health Informatics*. 2018; 12 (3): 45–59. DOI: https://doi.org/10.1016/j.jhi.2018.05.002.

24. Nguyen, A. & Tran, M. “Building resilient IoT systems with eventual consistency”. *IEEE Internet of Things Journal.* 2019; 6 (5): 8654–8663. DOI: https://doi.org/10.1109/JIOT.2019.291561.

25. Olson, J. & Lee, S. “Paxos and its vulnerabilities: A comparative study”. *Journal of Network Protocols*. 2020; 19 (2): 100–114. DOI: https://doi.org/10.1007/s12083-020-00827-z.

26. Peterson, L. & Watson, A. “Smart grid management through distributed consensus”. *Energy Systems.* 2023; 14 (1): 33–48. DOI: https://doi.org/10.1007/s12667-023-00540-1.

27. Quinn, M. “Data pipelines in smart cities”. *Urban Computing Journal.* 2021; 3 (2): 28–40. DOI: https://doi.org/10.1002/ucj.2021.0023.

28. Roberts, J. & Yates, C. “High-frequency trading and distributed ledger technologies: A survey”. *International Journal of Financial Markets*. 2022; 60: 100–118. DOI: https://doi.org/10.1016/j.ijfinmar.2022.01.005.

29. Shang, J. **“**A double-layer structure for Raft consensus mechanism”. *Journal of Network and Computer Applications*. 2025. DOI: https://doi.org/10.1016/j.jnca.2025.103894.

30. Singh, P. & Kumar, R. “5G-enabled edge computing: Architecture and performance”. *Telecommunications Systems.* 2020; 74 (2): 143–159. DOI: https://doi.org/10.1007/s11235-020-00645-7.

31. Tanenbaum, A. S. & Van Steen, M. *“*Distributed systems: Principles and paradigms”. *Pearson Education*. 2017. – Available from: https://books.google.com.ua/books?id=zdPfoQEACAAJ&redir_esc=y.

32. Taylor, S. & Choi, Y. “Evaluating Raft in IoT meshes networks”. *Sensors*. 2019; 19 (22): 4867. DOI: https://doi.org/10.3390/s19224867.

33. Umapathy, R. & Bhat, S. “Performance modeling of distributed stream processing”. *Concurrency and Computation: Practice and Experience*. 2023; 35 (5): e6602. DOI: https://doi.org/10.1002/cpe.6602.

34. Verma, A. & Zhao, P. “Lambda vs Kappa architecture in big data systems: Comparative study”. *Journal of Big Data.* 2021; 8: 45. DOI: https://doi.org/10.1186/s40537-021-00449-9.

35. Vogels, W. “Eventually consistent”. *Communications of the ACM.* 2009; 52 (1): 40–44. DOI: https://doi.org/10.1145/1435417.1435432.

36.Wang, L. & Chen, Y. “Architecture patterns for edge-based IoT systems”. *IEEE Internet Computing*. 2020; 24 (2): 15–23. DOI: https://doi.org/10.1109/MIC.2020.2973601.

37. Yu, D., Liu, Z., Zhang, H. & Li, J. “Adaptive protocol of Raft in wireless network”. *Computer Communications*. 2024; 222: 208–220. DOI: https://doi.org/10.1016/j.comcom.2023.11.015.

Conflicts of Interest: The authors declare that they have no conflict of interest regarding this study, including financial, personal, authorship or other, which could influence the research and its results presented in this article

Received 22.07.2025
Received after revision 18.09.2025
Accepted 26.09.2025

DOI: https://doi.org/10.15276/hait.08.2025.21
УДК 004.72:004.4'272:004.415

Розробка та оптимізація розподілених високопродуктивних систем із забезпеченням консистентності даних у реальному часі

Гуменюк Андрій Олександрович
ORCID: https://orcid.org/0009-0002-0985-1146; andy.gumenyuk@gmail.com
DASTA Incorporated (“dub”). Нью-Йорк, штат Нью-Йорк, 10006, США

АНОТАЦІЯ

У статті розглянуто теоретичні та прикладні аспекти побудови розподілених високопродуктивних систем, здатних обробляти великі обсяги даних із мінімальними затримками та забезпеченням консистентності у режимі реального часу. Актуальність дослідження зумовлена стрімким розвитком сфер обробки великих даних – фінансових технологій, телемедицини, смарт-сіті, Інтернету речей та автономного транспорту, де навіть мілісекундні затримки можуть мати критичні наслідки. Дослідження присвячене аналізу сучасних архітектурних моделей, серед яких мікросервіси, CQRS, архітектури Lambda і Kappa, а також подійно-орієнтовані системи на основі Apache Kafka. Детально розглянуто методи оптимізації продуктивності, що охоплюють асинхронну обробку запитів, кешування, реплікацію та балансування навантаження, із особливою увагою до edge-computing і інфраструктур на основі 5G. Значна увага приділяється також питанням консистентності даних, зокрема CAP-теоремі (Consistency Availability Partition tolerance), алгоритмам консенсусу (Paxos, Raft) та структурам CRDT (Conflict-free Replicated Data Type). Результати показали, що універсального архітектурного рішення не існує, проте ефективними виявляються гібридні підходи, які поєднують моделі сильної та поступової узгодженості для різних сценаріїв. Запропонована аналітична модель оцінювання продуктивності дає змогу обирати оптимальні конфігурації систем залежно від навантаження та вимог до узгодженості даних.

Наукова новизна роботи полягає у створенні інтегрованої моделі, що поєднує архітектурні патерни, методи оптимізації та алгоритми консенсусу, а також у розробці аналітичної моделі оцінювання ефективності, яка раніше не була достатньо представлена в літературі. Практична значущість полягає у формуванні рекомендацій щодо проєктування відмовостійких і масштабованих IT-інфраструктур для фінансів, телекомунікацій, охорони здоров'я, IoT та смарт-сіті.

Ключові слова: розподілені системи; висока пропускна здатність; архітектура систем; консистентність даних; узгодженість у реальному часі; оптимізація продуктивності; CAP-теорема; горизонтальне масштабування; fault tolerance; алгоритми консенсусу

ABOUT THE AUTHOR

Andrii O. Humeniuk - Master Degree in Software Engineering, Lead Software Engineer, DASTA Incorporated ("dub"), New York, NY, 10006, USA
ORCID: https://orcid.org/0009-0002-0985-1146; andy.gumenyuk@gmail.com
Research field: Software Engineering

Гуменюк Андрій Олександрович - магістр з розробки програмного забезпечення, лід-інженер, DASTA Incorporated ("dub"). Нью-Йорк, штат Нью-Йорк, 10006, США