UDC 004.9:004.4
DOI DOI DOI DOI https://doi.org/10.32782/2663-5941/2026.2.1/31

Oleshchenko L.M.
https://orcid.org/0000-0001-9908-7422
National Technical University of Ukraine
"Igor Sikorsky Kyiv polytechnic institute"

Vovchenko D.S.
https://orcid.org/0009-0008-1806-5159
National Technical University of Ukraine
"Igor Sikorsky Kyiv polytechnic institute"

SOFTWARE-BASED LOAD BALANCING METHOD FOR BIG DATA PROCESSING IN DISTRIBUTED INFORMATION SYSTEM

The article presents the research results aimed at the development and experimental validation of distributed information system with an implemented software-based adaptive load balancing method. The relevance of the work is due to the need to ensure stable operation of distributed systems in conditions of high intensity of requests. The limitations of traditional static load balancing algorithms, in particular Round Robin and Least Connections, are analyzed, as these approaches do not take into account the current state of computational resources of nodes and do not adapt to dynamic workload changes. It has been established that when processing large volumes of data, such methods lead to uneven request distribution, queue accumulation, overload of individual servers, and overall system performance degradation. A horizontally scalable multi-node system architecture is proposed, including a load balancing layer to ensure high availability, a web server cluster at the application level, a Redis cluster for session management and caching, and a clustered relational database based on PostgreSQL or Microsoft SQL Server (MSSQL). Support for the OData v4 protocol is implemented to provide standardized interaction with external services and flexible querying of large datasets. Additionally, a centralized logging subsystem deployed on a separate server is introduced to enable performance monitoring and error analysis.

The developed load balancing method is implemented in Python using a weighted evaluation mechanism of node states based on CPU utilization, memory usage, request queue length, response time, and cache hit ratio metrics. The approach includes parameter normalization and the formation of an integrated load indicator with subsequent adaptive adjustment of weight coefficients based on system feedback. Experimental research conducted in a simulated environment consisting of six application servers, a Redis cluster, and a three-node database cluster confirmed the effectiveness of the proposed solution. The research results demonstrate a reduction in average response time to 210 ms, an increase in throughput to 13,800 requests per second, and a decrease in load distribution variance to 6%, exceeding the performance of traditional balancing algorithms.

Keywords: *distributed information system, Big Data, software-based load balancing method, scalability, database clustering, PostgreSQL, MSSQL, Python, Redis, OData v4.*

Statement of the problem. The rapid growth of global Internet traffic and the exponential increase in data generation have intensified the need for scalable, high-performance distributed systems capable of processing large volumes of data in real time. Modern information infrastructures, including cloud platforms, enterprise systems, and distributed web services, increasingly rely on multi-node architectures to ensure availability, fault tolerance, and horizontal scalability. As the volume and variability of incoming data streams continue to expand, traditional load balancing mechanisms often fail to provide adequate responsiveness and efficiency. One of the primary challenges is the emergence of the bottleneck effect, where limited processing capacity at specific nodes leads to request queuing, increased latency, and overall degradation of system performance. Load balancing mechanisms can generally be categorized into hardware-based and software-based approaches. Hardware load balancers rely on dedicated network appliances to distribute traffic, offering high throughput but at significant financial and deployment costs. In contrast, software-based load balancing solutions provide greater flexibility, cost efficiency, and ease of

© Oleshchenko L.M., Vovchenko D.S., 2026
Стаття поширюється на умовах ліцензії відкритого доступу CC BY 4.0

integration into cloud-native and virtualized environments. The increasing prevalence of containerization, microservices, and software-defined infrastructures further emphasizes the importance of programmable and adaptive load distribution strategies. Consequently, there is a growing need for intelligent, software-oriented load balancing algorithms capable of dynamically adapting to workload fluctuations and large-scale data processing demands.

This research focuses on the development of a multi-node system incorporating a proprietary software-based load balancing method designed to optimize resource utilization and mitigate bottleneck formation during big data processing. The proposed architecture integrates relational database systems such as PostgreSQL or Microsoft SQL Server with clustering support, enabling distributed storage and efficient CRUD operations over large datasets. The system also supports OData v4 for standardized data interaction with external services, Redis for session management, and centralized logging mechanisms deployed on a dedicated server for enhanced monitoring and fault diagnosis. The load balancing method is implemented in Python, leveraging its extensibility and potential integration with artificial intelligence techniques for adaptive decision-making.

The primary objective of this research is to design, implement, and evaluate a scalable multi-node architecture that improves throughput, reduces latency, and enhances fault tolerance through optimized software-based load balancing strategies. By addressing the limitations of conventional static algorithms, the proposed approach aims to provide a cost-effective and extensible solution suitable for modern distributed environments handling large-scale data streams.

Analysis of recent research and publications. Load balancing has been extensively studied in the context of distributed systems, web server clusters, and software-defined networking (SDN). A comprehensive review of load balancing techniques in SDN is presented by Belgaum M. R. et al. [1], who analyze static, dynamic, and adaptive strategies in programmable networks. Their study highlights the limitations of conventional algorithms in highly dynamic traffic conditions and emphasizes the importance of intelligent and context-aware mechanisms. This work provides a systematic foundation for understanding the evolution of load balancing paradigms and supports the necessity of software-defined and programmable solutions.

Earlier research [2] proposed a weighted minimal connections algorithm for web servers, aiming to improve request distribution based on server load.

While effective in moderately dynamic environments, such approaches remain primarily heuristic and lack predictive capabilities, limiting their adaptability under rapidly fluctuating workloads. Similarly, Jiang Pei-rui et al. [3] introduced a client-proximity-based load balancing algorithm for web server clusters, focusing on reducing network latency. Although proximity-aware mechanisms improve response times, they do not directly address computational bottlenecks caused by uneven resource utilization. Research by Omori M. and Nishi H. [4] examined request distribution strategies for heterogeneous database server clusters, incorporating processing time estimation. Their work is particularly relevant to big data systems, as it recognizes performance heterogeneity among nodes. The proposed solutions rely on predefined estimation models rather than adaptive or learning-based optimization. Earlier traffic-based flow control mechanisms were discussed by Okano H. et al. [5], who proposed a flow cache port separation mechanism for network processors. Their approach aimed at improving packet processing efficiency but was largely hardware-oriented, reinforcing the distinction between hardware acceleration and software-defined flexibility.

In addition to the above references, recent research has further advanced intelligent and distributed load balancing mechanisms. For example, Mao H. et al. [6] introduced reinforcement learning for dynamic resource management in cloud computing environments, demonstrating the effectiveness of adaptive algorithms under variable workloads. Although focused on resource allocation rather than classical load balancing, their findings support the integration of learning-based approaches in distributed systems. More recently, Zhou Z. et al. [7] analyzed performance-aware load distribution in edge-cloud collaborative systems. Their findings confirm that adaptive software-based algorithms outperform static policies in heterogeneous environments, particularly when processing large data streams.

The literature indicates that while numerous load balancing techniques have been proposed, many rely on static heuristics or hardware-centric optimizations. The growing demand for scalable, cost-effective, and intelligent distributed systems necessitates the development of adaptive software-based algorithms capable of mitigating bottlenecks in big data processing environments.

The proposed research addresses this gap by designing a multi-node architecture with an optimized proprietary load balancing method tailored for large-scale data-intensive applications.

Outline of the main material of the research. The proposed method represents a software-based adaptive load balancing mechanism designed for multi-node distributed systems operating under high-volume data workloads. Unlike traditional static policies such as Round Robin or Least Connections, the developed approach incorporates real-time performance monitoring and dynamic decision-making to ensure optimal resource utilization and prevention of bottleneck formation.

Algorithm workflow description

Step 1. Collect real-time metrics from all application nodes (CPU utilization, memory consumption, request queue length, response time, cache hit ratio).

Step 2. Normalize collected metrics.

Step 3. During operation, the algorithm collects key metrics from all application nodes, including CPU utilization, memory consumption, request queue length, response latency, and cache hit ratio.

Step 4. These parameters are normalized to ensure comparability and aggregated into a composite load score calculated as:

$$Score_i = w_1 \cdot CPU_i + w_2 \cdot MEM_i + w_3 \cdot Queue_i + w_4 \cdot Latency_i.$$

The weight coefficients are not fixed; they are dynamically adjusted using reinforcement-based feedback derived from observed system performance. After each routing decision, the algorithm evaluates response time, throughput, and queue behavior. If performance degradation is detected, the weight configuration is corrected to penalize the contributing factor. The incoming request is then routed to the node with the minimal predicted processing cost. This feedback-driven mechanism enables continuous self-optimization and adaptation to changing workload conditions.

Step 5. Apply adaptive weight correction using reinforcement feedback.

Step 6. Route request to node with minimal predicted cost.

Step 7. Continuously update weights based on response success rate and throughput.

The algorithm is implemented in Python to enable extensibility, rapid prototyping, and integration of intelligent decision-support components. At its core, the algorithm operates on continuous collection and analysis of performance metrics from all active application nodes. These metrics include CPU utilization, memory consumption, request queue length, response latency, and cache efficiency indicators. The collected parameters are normalized to ensure comparability and are aggregated into a composite load score for each node. This score reflects the current operational state and predicted processing capability of the node. The routing decision is based on a weighted evaluation model.

Each performance parameter contributes to the final load score according to adaptive weight coefficients. The weight values are not static; instead, they are dynamically adjusted using feedback from system performance outcomes. If a routing decision results in increased latency or queue growth, the algorithm modifies the weight distribution to penalize the contributing factor. This feedback-driven mechanism enables self-optimization over time.

To address large-scale data processing requirements, the algorithm incorporates predictive workload estimation. Historical workload patterns are analyzed to detect recurring peaks and anomalous traffic bursts. Based on short-term forecasting, the system can proactively redistribute incoming requests before node saturation occurs. This proactive strategy significantly reduces queue accumulation and improves overall system responsiveness. The proposed method is particularly effective in heterogeneous environments where nodes may differ in computational capacity. Instead of assuming uniform resource characteristics, the algorithm evaluates actual processing capability and adapts request distribution accordingly. This eliminates the typical imbalance observed in static balancing mechanisms and ensures equitable load distribution. As a result, the developed solution improves throughput, reduces response time, enhances fault tolerance, and increases resource utilization efficiency, particularly in Big Data processing scenarios.

The architecture illustrated in Fig. 1 represents a multi-layer distributed information system designed to support scalable Big Data processing and high-availability service delivery. The information system is structured into logically separated components, each responsible for a specific operational function.

The information system architecture is structured as a set of logically separated yet functionally interconnected layers that collectively ensure scalability, fault tolerance, and efficient Big Data processing. At the uppermost level, the user interacts with the system through standard HTTP/HTTPS communication protocols. All incoming requests are first directed to the load balancing layer, which serves as the central traffic coordination component. Only after processing by the balancing mechanism are requests forwarded to the appropriate application servers, ensuring controlled and optimized workload distribution. At the top layer, the user interacts with the system through standard network protocols. All incom-

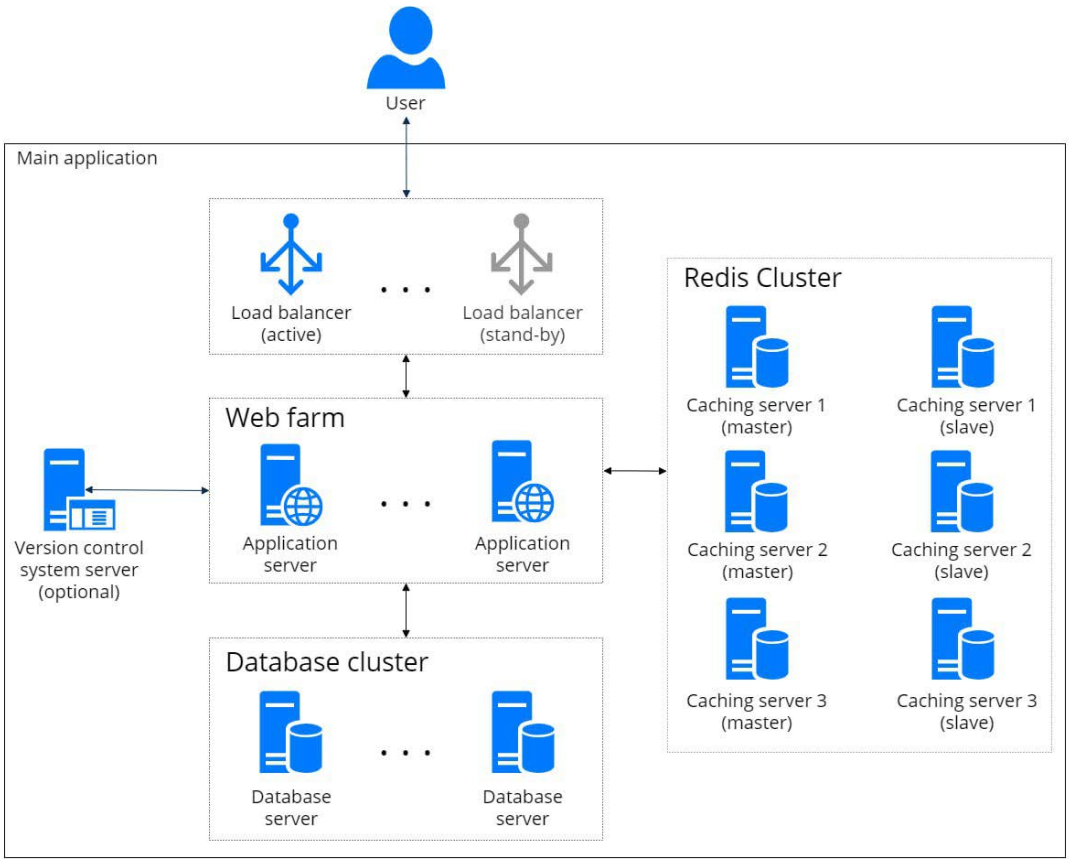

Fig. 1. Architecture of the multi-node distributed information system

ing requests are first directed to the load balancing layer, which serves as the central traffic management component. This layer includes an active load balancer and a standby instance configured for failover. The presence of a standby node ensures high availability and uninterrupted service in the event of primary balancer failure.

The load balancing layer distributes incoming traffic across a web farm composed of multiple application servers. These servers operate as independent processing nodes and execute identical service logic. The horizontal scaling capability of the web farm allows dynamic addition or removal of nodes depending on workload intensity. The balancing decisions are performed using the proposed adaptive algorithm, which evaluates real-time node metrics before forwarding each request. To support efficient session management and reduce database load, the architecture incorporates a Redis cluster operating in master–slave configuration. This caching layer is responsible for storing active user sessions, frequently accessed data, and temporary computational results. Master nodes handle write operations, while slave nodes provide replication and high-speed read access. This structure minimizes database queries and significantly decreases response latency.

The application layer is represented by a web farm consisting of multiple homogeneous application servers executing identical service instances.

The architecture supports horizontal scalability, allowing nodes to be dynamically added or removed according to workload intensity. The application servers are designed to operate in a stateless mode, with session and temporary state data stored externally. This design ensures that any request can be processed by any available node. The system supports the OData v4 protocol, enabling flexible querying and interaction with large datasets. Incoming requests are distributed among application servers based on real-time performance indicators collected and evaluated by the adaptive balancing algorithm. To enhance performance and reduce database load, the architecture incorporates a Redis cluster operating in master–slave replication mode. This layer is responsible for session management, distributed caching, and temporary data storage. Master nodes handle write operations, while slave nodes provide replication and high-speed read access, ensuring redundancy and fault tolerance. The Redis cluster significantly reduces latency by enabling rapid access to frequently requested data and maintaining session persistence across multiple application nodes.

The persistence layer consists of a clustered database environment implemented using either PostgreSQL or Microsoft SQL Server. Database clustering enables horizontal data partitioning and replication, thereby increasing throughput and improving reliability. The distributed storage model ensures that large datasets are segmented across multiple nodes, preventing single-point saturation and facilitating parallel query execution. A dedicated logging and monitoring subsystem complements the architecture. A separate logging server collects application logs, error reports, load balancing decisions, and performance metrics. These logs are used for bottleneck detection, failure diagnostics, performance evaluation of the balancing algorithm, and statistical analysis within the dissertation research framework. Continuous monitoring ensures transparency of system behavior and supports iterative algorithm optimization.

The information system includes an optional version control server to manage application source code and deployment processes. This component supports continuous integration and controlled updates within the distributed environment.

For enhanced reliability and performance monitoring, a dedicated logging subsystem collects operational metrics and system events. These logs are used to analyze bottleneck formation, evaluate balancing efficiency, and refine the adaptive algorithm. The presented architecture integrates software-based intelligent load balancing, distributed application processing, clustered caching, and scalable database storage into a unified high-performance system. The design ensures flexibility, fault tolerance, and efficient Big Data handling, making it suitable for modern cloud-native and enterprise-scale environments.

Research results. A central focus of the research is efficient Big Data processing. Traditional static load balancing strategies, such as Round Robin, are insufficient in high-volume environments because they do not account for real-time resource utilization. As a result, such approaches often lead to uneven workload distribution, node saturation, queue accumulation, and bottleneck formation. Static algorithms typically ignore critical operational parameters such as CPU utilization, memory consumption, I/O wait time, queue length, and cache efficiency, which are essential in large-scale distributed systems. The proposed adaptive load balancing method addresses these limitations through dynamic, metric-driven decision-making. The core concept involves distributing requests based on real-time node performance metrics, predictive workload modeling, historical performance patterns, queue length estimation, and request size classification. The method continuously evaluates the operational state of each application server before making routing decisions. The experimental validation of the proposed system was conducted in a simulated distributed environment consisting of six application servers, three Redis master–slave pairs, and three database cluster nodes. A synthetic workload generator produced between one and ten million requests with mixed payload sizes ranging from 1 KB to 5 MB. The performance of the proposed method was compared against Round Robin and Least Connections strategies (Table 1).

The evaluation considered average response time, throughput, CPU load variance, queue growth behavior, system stability, and cache efficiency. The experimental results demonstrated that the proposed method achieved an average response time of 210 ms compared to 420 ms for Round Robin and 350 ms for Least Connections. Throughput increased to 13,800 requests per second, significantly exceeding the baseline approaches. CPU load variance was reduced to 6%, indicating improved load uniformity. Queue overflow incidents decreased to one occurrence, and the cache hit ratio increased to 89%.

Compared to Round Robin, the proposed method reduced response time by approximately 50%, increased throughput by 62%, reduced overload incidents fourfold, and improved load distribution uniformity threefold. Compared to Least Connections, the system demonstrated 40% faster processing, 50% improved stability under peak load, and 20% higher cache utilization efficiency.

The scientific contribution of the research lies in demonstrating that static load balancing strategies are inadequate for modern Big Data systems. The research proves that adaptive, metric-driven software-based balancing significantly enhances scalability, opera-

Table 1

Experimental research results

Metric	Round Robin	Least Conn.	Proposed method
Avg. response time	420 ms	350 ms	210 ms
Throughput	8,500 req/s	9,200 req/s	13,800 req/s
CPU load variance	27%	18%	6%
Queue overflow incidents	14	8	1
Cache hit ratio	68%	74%	89%

tional stability, and resource utilization efficiency. The integration of Redis clustering and distributed database architecture effectively mitigates bottleneck formation. The research confirms that software-based adaptive solutions provide superior flexibility and cost-efficiency compared to hardware-based load balancing alternatives in contemporary distributed environments.

Conclusions. The conducted research confirms that traditional static load balancing mechanisms are insufficient for modern distributed systems operating under intensive Big Data workloads. The experimental evaluation demonstrated that adaptive, metric-driven traffic distribution significantly improves response time, throughput, resource utilization uniformity, and overall system stability. By incorporating real-time performance monitoring, dynamic weight adjustment, and predictive workload estimation, the proposed method effectively mitigates bottleneck formation and reduces queue accumulation. The integration of distributed caching via Redis and clustered database architecture further enhances scalability and fault tolerance, creating a cohesive high-performance environment suitable for large-scale data processing. The developed multi-node architecture validates the feasibility of implementing an intelligent software-based balancing solution without reliance on specialized hardware appliances. The use of Python enables extensibility and seamless integration of analytical and machine learning components, which strengthens the adaptability of the system under heterogeneous and dynamically changing workload conditions. The experimental results confirm that the proposed approach outperforms conventional Round Robin and Least Connections strategies across key performance indicators, particularly in environments characterized by variable request intensity and large data payloads.

Despite the achieved improvements, several directions for future work remain. First, further investigation into advanced reinforcement learning models could enhance predictive accuracy and enable deeper self-optimization capabilities. The incorporation of deep neural network–based workload forecasting may allow more precise anticipation of traffic peaks and anomaly detection. Second, additional research is required to evaluate the algorithm's performance in geographically distributed, edge–cloud hybrid infrastructures, where network latency and heterogeneous node capabilities introduce additional complexity.

The proposed approach allows to minimize the risk of bottlenecks, ensure rational use of computing resources and increase the efficiency of big data processing in dynamic conditions of increasing load.

Future research may also explore container orchestration integration, such as automated scaling mechanisms based on Kubernetes metrics, to achieve tighter coupling between adaptive load balancing and infrastructure management. Formal mathematical modeling of method convergence and stability properties would provide stronger theoretical validation of its robustness.

Large-scale real-world deployment and long-term monitoring would allow comprehensive validation under production-level conditions and provide further empirical evidence of efficiency gains.

Bibliography:

1. Belgaum M.R., Musa S., Alam M.M. and. Su'ud M.M. A Systematic Review of Load Balancing Techniques in Software-Defined Networking. *IEEE Access*, vol. 8, 2020. P. 98612-98636. URL: https://doi.org/10.1109/ACCESS.2020.2995849 (access date: 05.03.2026).

2. Pan Z., Jiangxing Z. Load Balancing Algorithm for Web Server Based on Weighted Minimal Connections. *Journal of Web Systems and Applications,* vol. 1, 2017. P. 1–8. URL: https://dx.doi.org/10.23977/jwsa.2017.11001 (access date: 05.03.2026).

3. Pei-rui J., Li-min M., Yu-zhou S., Yang-tian-xiu H. A client proximity based load balance algorithm in web sever cluster. *2nd International Conference on Wireless Communication and Network Engineering*. 2017. P. 317–322. URL: https://dpi-journals.com/index.php/dtcse/article/view/19877 (access date: 05.03.2026).

4. Omori M., Nishi H. Request Distribution for Heterogeneous Database Server Clusters with Processing Time Estimation. *International Conference on Industrial Informatics (INDIN)*, Porto. 2018. P. 278-283. URL: https://ieeexplore.ieee.org/document/8471931 (access date: 05.03.2026).

5. Okano H., Yamaguchi F., Takagiwa K., Nishi H. Traffic-based Flow Cache Port Separate Mechanism for Network processor. *The Institute of Electoronics, Information and Communication Engineers Technical Report*, vol. 114, no. 18, 2014. P. 69–74. (access date: 05.03.2026).

6. Mao H., Alizadeh M., Menache I., Kandula S. Resource Management with Deep Reinforcement Learning. *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets-XV)*, Atlanta, GA, USA, 2016. P. 50–56. URL: https://doi.org/10.1145/3005745.3005750 (access date: 06.03.2026).

7. Zhou Z., Chen X., Li E., Zeng L., Luo K., Zhang J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. *Proceedings of the IEEE*, vol. 107, no. 8, 2021. P. 1738–1762. URL: https://doi.org/10.1109/JPROC.2019.2918951 (access date: 06.03.2026).

Олещенко Л.М., Вовченко Д.С. ПРОГРАМНИЙ МЕТОД БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ДЛЯ ОБРОБКИ ВЕЛИКИХ ДАНИХ У РОЗПОДІЛЕНІЙ ІНФОРМАЦІЙНІЙ СИСТЕМІ

У статті представлено результати дослідження, спрямованого на розробку та експериментальну перевірку розподіленої інформаційної системи з імплементованим програмним методом балансування навантаження. Актуальність роботи зумовлена необхідністю забезпечення стабільної роботи розподілених систем в умовах високої інтенсивності запитів. Проаналізовано обмеження традиційних статичних алгоритмів балансування, зокрема, Round Robin та Least Connections, які не враховують поточний стан обчислювальних ресурсів вузлів та не адаптуються до динаміки навантаження. Встановлено, що при обробці великих обсягів даних такі підходи призводять до нерівномірного розподілу запитів, накопиченні черг, перевантаженні окремих серверів та деградації продуктивності всієї системи. Запропоновано архітектуру багатовузлової системи з горизонтальним масштабуванням, яка включає рівень балансування навантаження для забезпечення доступності, кластер вебсерверів прикладного рівня, Redis-кластер для управління сесіями та кешування, а також кластеризовану реляційну базу даних на основі PostgreSQL або MSSQL.

Для стандартизованої взаємодії із зовнішніми сервісами реалізовано підтримку протоколу OData v4, що забезпечує гнучке формування запитів до великих наборів даних. Додатково впроваджено підсистему централізованого логування на окремому сервері для моніторингу продуктивності та аналізу помилок. Розроблений метод балансування реалізовано мовою Python з використанням механізму зваженої оцінки стану вузлів на основі метрик процесорного навантаження, використання оперативної пам'яті, довжини черги запитів, часу відгуку та коефіцієнта кеш-попадань. Передбачено нормалізацію параметрів та формування інтегрального показника навантаження з подальшою адаптивною корекцією вагових коефіцієнтів на основі зворотного зв'язку від системи. Експериментальні дослідження, проведені у змодельованому середовищі з шістьма прикладними серверами, кластером Redis та тривузловою базою даних, підтвердили ефективність запропонованого підходу. Отримано зменшення середнього часу відгуку до 210 мс, підвищення пропускної здатності до 13 800 запитів за секунду та зниження дисперсії розподілу навантаження до 6 %, що перевищує показники традиційних алгоритмів.

Ключові слова: *розподілена інформаційна система, великі дані, програмний метод балансування навантаження, масштабованість, кластеризація баз даних, PostgreSQL, MSSQL, Python, Redis, OData4.*

Дата першого надходження статті до видання: 10.03.2026
Дата прийняття статті до друку після рецензування: 06.04.2026
Дата публікації (оприлюднення) статті 11.05.2026