Автоматизовані системи управління та прилади автоматики. 2025. № 4 (187)

UDC 004.8 + 004.056.5 **DOI:** https://doi.org/10.30837/0135-1710.2025.187.182

M. Glavchev, D. Hlavchev, V. Panchenko

EVALUATING THE EFFECTIVENESS OF OPEN-SOURCE SOLUTIONS FOR MONITORING AND LOAD BALANCING IN MICROSERVICE APPLICATIONS

Subject of Research. Subject of the article is open-source solutions and their application for monitoring and load balancing in microservice applications operating in specialized computer systems. The research covers a wide range of tools, including metric collection systems, centralized logging, and distributed request tracing. Relevance of the work is determined by the constant growth in complexity of distributed architectures and the critical need for effective performance control (observability) and stable traffic distribution. **Goal.** The goal of the work is comprehensive empirical evaluation and comparison of key open-source monitoring and load balancing tools, specifically Prometheus/Grafana (for metrics), ELK stack (for logs), HAProxy, Nginx, Traefik (load balancers), as well as Istio and Linkerd (service mesh), with the goal of developing practical recommendations for designing and operating microservice systems. **Tasks.** The tasks are conduct analysis of popular open-source tools, define criteria for their effectiveness, create a test environment based on Kubernetes and conduct a series of load tests with various configurations, as well as perform quantitative assessment of key performance indicators, including latency, throughput, and resource utilization. **Methods.** Applied methods of systematic analysis, empirical modeling, and benchmarking. For objective comparison, load testing methods (baseline and stress scenarios) were used in a Kubernetes cluster. Key evaluation criteria included request processing latency, throughput, and resource overhead of the tools themselves. **Result.** The obtained results confirm that open-source solutions are capable of providing high-level observability and effective load balancing in specialized computer systems, while remaining a cost-effective alternative to commercial products. The study identified strengths and weaknesses of each tool, allowing for informed selection based on specific project requirements. **Conclusions.** Confirmed the ability of open-source tools to effectively provide observability and load management in specialized computer systems, remaining a cost-effective alternative to commercial products. The conclusions made allow for the formulation of practical recommendations for designing and operating microservice applications with a focus on stability and performance. The research results can be used in making architectural decisions for distributed systems of various scales.

Keywords: load balancing; microservice; monitoring; open-source; service mesh; specialized computer system.

Introduction

Problem statement. Modern information systems are increasingly being designed based on microservice architecture. This approach involves dividing a complex application into a set of relatively independent services, each of which performs clearly defined functions and can be deployed separately. The use of microservices increases development flexibility, scalability, and system resilience, which is especially important in specialized computer systems, such as telecommunications platforms, financial solutions, or industrial IoT complexes.

At the same time, microservice architecture creates new challenges. The significant complexity of the structure necessitates effective performance monitoring, tracking dependencies between services, and load balancing between different application instances. In the event of high loads or uneven distribution of requests, there is a risk of performance

degradation or even failure of individual components, which can negatively affect the functioning of the entire system.

Traditionally, monitoring and load balancing tasks have been solved using commercial tools, which often have closed code and significant licensing costs. However, in recent years, an ecosystem of open-source solutions has been actively developing, providing a wide range of functions and not inferior in capabilities to many commercial counterparts. The most well-known of these tools include Prometheus and Grafana (metrics collection and visualization), ELK stack (report collection and analysis), HAProxy, Nginx, and Traefik (load balancing at L4/L7 levels), as well as specialized infrastructure-level solutions for facilitating communication between services, service mesh – Istio and Linkerd, which integrate directly into the Kubernetes environment and provide flexible management of network flows.

The relevance of the study is due to the need to verify the effectiveness of these solutions in the real conditions of specialized computer systems. For practical application, it is important not only to compare capabilities theoretically, but also to evaluate key performance indicators experimentally:

– average request processing delay and delays at the p95/p99 percentile levels;

– system throughput (number of requests processed per second);

– resource usage (central processing unit – CPU, random access memory – RAM) by the tools themselves;

– ease of integration into existing microservice architectures.

Analysis of recent studies and publications. In operational microservice systems, metrics, reports (logs), and tracing are most often combined; at the same time, the priority metrics are latency (including p95/p99), throughput, and resource consumption (CPU/RAM) of monitoring tools. A practical picture of the implementation of such approaches is provided by an empirical study of practitioners [1], which identifies log management, exceptions, and load balancing as subjects of constant performance monitoring among standard practices. A systematic review of monitoring tools (71 OSS solutions), their capabilities and limitations, is presented in the JSS review [2], which is useful for forming criteria for selecting an observability stack for a specific specialized computer system (SCS). In terms of log analytics, the focus on the ACM Computing Surveys [3] review allows us to align the choice of ELK (Elasticsearch, Kibana & Logstash)/related tools with modern methods of parsing, anomaly detection, and datasets for validation. These three sources form the methodological basis of the monitoring section of our study.

The classic CACM paper on Borg–Omega–Kubernetes [4] justifies the evolution to containerized load management and the importance of resource isolation for combining latency-sensitive services with batch tasks. An early experimental comparative analysis of microservices in containers and in a monolith [5] confirmed the suitability of containers for low overhead and rapid scaling. For experimental research, it is important to have a formal basis in the resource models of Kubernetes systems, which provide resource consumption forecasts and help correlate the overhead costs of monitoring/mesh/LB with the performance of microservices. A recent JSS article on building such models is useful here [6].

When choosing open-source load balancers at the L4/L7 level (HAProxy, Nginx, Traefik, Envoy), pay attention to key parameters such as latency, throughput, and stability under high load.

Practical measurements in server WWW infrastructures showed similar results for HAProxy and Nginx under high loads, with the best performance seen in Linux Virtual Server (LVS approach) [7]. Some studies offer mathematical models for analyzing HAProxy performance, including MMAP/PH/M/N queues and Monte Carlo simulation with confirmation by measurements on a test bench [8]. For Kubernetes environments, Ingress controllers were additionally considered: a 2024 IEEE paper compares implementations and balancing algorithms specifically in a Kubernetes cluster, which directly coincides with our experimental setup [9–11]. Together, these sources outline the trade-offs between pure L4 performance and L7 policy flexibility for microservice applications.

Service mesh adds L7 routing, observability, and security policies (mTLS mutual authentication) to the system, but introduces latency and CPU/memory consumption. A detailed analysis of sidecar proxy overhead with measured effects on latency (increase to ~2.7× on certain configurations) and virtual central processing unit (vCPU) [12] shows that the amount of overhead depends heavily on the configuration (TCP proxying vs. protocol parsing) and workload. A comparison of Istio and Linkerd in edge conditions showed Linkerd to be more "edge-friendly" due to lower overhead [13]. A separate technical report on the impact of mTLS in different implementations (Istio, Ambient Istio, Linkerd, Cilium) helps to separate security and performance effects in experimental design [14]. Additionally, the performance of a mesh cluster also depends on related Kubernetes components – for example, the etcd configuration, which affects the overall behavior of the control plane and the application [15]. Collectively, these works set a corridor of expectations for observability overhead and security policies, which we will take into account when planning tests.

A separate class of work focuses on container platforms and their interaction with balancing/monitoring. Review and applied articles propose load balancing mechanisms in Docker-/Swarm-/Kubernetes environments for productive and resource-constrained scenarios, including big data and edge [16]. At the same time, the Ingress layer in Kubernetes remains an active field of comparison (implementations, algorithms, overhead), where recent IEEE results [9] allow the experiment to be enriched with practical configurations. Together with resource consumption models [6], this provides a methodologically correct correlation of observability, balancing, and performance in a microservice SCS.

The purpose of this article is to study the effectiveness of open-source solutions for monitoring and load balancing in microservice applications running on specialized computer systems.

To achieve this goal, the following tasks must be solved:

– review modern open-source tools in the field of monitoring and balancing;

– determine the criteria for evaluating their effectiveness in the context of microservice application performance;

– build an experimental environment using a test microservice application;

– conduct a series of load tests using different tools and record the results;

– perform a comparative analysis of the data obtained and formulate practical recommendations.

Thus, the work aims to bridge the gap between the theoretical capabilities of open-source tools and their actual effectiveness when used in SCS.

Presentation of the main material

1. Theoretical aspects

1.1. Microservice architecture and performance

Microservice architecture is one of the leading approaches to building modern distributed applications. A microservice is an independent software component that implements a separate business function and interacts with other components through well-defined interfaces, primarily via the API interface of the HTTP/gRPC remote procedure call system or asynchronous message queues. This architecture provides development flexibility, the ability to choose the optimal technologies for each service, as well as deployment isolation and scaling independence. For SCS, the microservice approach is particularly valuable because it allows computing resources to be distributed across subsystems and complex data flows to be managed efficiently.

In the context of microservice applications, system performance is determined by its ability to process the required volume of requests with minimal delays and optimal use of computing resources. Important performance characteristics include average and percentile request processing delays (latency), throughput, and the level of utilization of the processor, RAM, and network resources. In SCS, performance is considered not only as a quantitative characteristic, but also as stability under peak loads, which requires the ability to automatically scale and adapt to dynamic conditions.

To formalize the task of performance evaluation, we introduce the following mathematical definitions and notations, which will be used in the experimental part of the study.

Definition 1. Let the system consist of a set of microservices $S = \{s_1, s_2, ..., s_n\}$, where each service s_i is characterized by a vector of performance parameters $P_i = (L_i, T_i, R_i)$, where L_i is the request processing delay, T_i is the throughput, and R_i is the resource consumption.

Definition 2. The request processing delay L is defined as the time interval between the moment the client sends the request $t_{request}$ and the moment the response is received $t_{response}$:

$$L = t_{response} - t_{request}. \quad (1)$$

Since the distribution of delays in microservice systems usually has a long tail, using only the average value is insufficient for an objective assessment. Therefore, the study uses percentile analysis, which allows us to evaluate the stability of the system for the vast majority of users.

Definition 3. The average delay is calculated as the arithmetic mean of all measurements:

$$\overline{L} = \left(\frac{1}{n}\right) \times \Sigma_{i=1^n} L_i. \quad (2)$$

Definition 4. The delay percentile p (e.g., $p95$, $p99$) is defined as the delay value that is not exceeded for p percent of all requests:

$$L_{p95} = L_{([0,95 \times n])}. \quad (3)$$

Definition 5. The throughput of a system T is defined as the number of successfully processed requests per unit of time:

$$T = N_{requests} / t_{interval}. \quad (4)$$

Definition 6. The resource overhead coefficient $O_{resource}$ characterizes the additional consumption of resources (CPU or RAM) by a monitoring or balancing tool relative to the baseline:

$$O_{resource} = \left(R_{with\_tool} - R_{baseline}\right) / R_{baseline} \times 100\%. \quad (5)$$

For a comprehensive assessment of the tool's effectiveness, an integrated indicator is proposed that takes into account all key characteristics with corresponding weighting coefficients:

$$E = \alpha \times \left(1/L_{p99}\right) + \beta \times T + \gamma \times \left(1/O_{resource}\right), \quad (6)$$

where α, β, γ are weighting coefficients determined by the priorities of a specific specialized computer system (SCS). For example, for latency-critical systems, $\alpha > \beta > \gamma$, while for resource-constrained edge systems, $\gamma > \alpha \approx \beta$ is appropriate.

1.2. Monitoring and observability

Monitoring is another important component, which involves the systematic collection, aggregation, and analysis of data on the functioning of the system. In microservice architectures, it is key to achieving observability, i.e., the ability to draw conclusions about the internal state of the system based on its external manifestations. Effective monitoring allows for the timely detection of performance degradation, service interaction failures, or security-threatening attacks.

According to the concept of the "three pillars of observability" [1–3], effective monitoring of microservice systems is based on three complementary components: metrics (quantitative indicators of the system's state), logs (event records), and tracing (tracking the path of a request through the system).

Based on an analysis of scientific sources [1–6] and the practical needs of the SCS, a system of criteria was developed to evaluate the effectiveness of open-source monitoring and load balancing tools (Table 1).

Table 1. *Criteria for evaluating the effectiveness of tools*

Criterion	Description	Metric	Priority
Latency	Request processing time from receipt to response	p95, p99 (ms)	High
Bandwidth	Number of requests processed per unit of time	RPS	High
Resource overhead	Additional CPU and RAM consumption by tools	% CPU, MB RAM	Medium

Continuation of the table 1

Criterion	Description	Metric	Priority
Scalability	Horizontal scaling capability	Number of nodes	Medium
Integration complexity	Time and effort required to implement the tool	Hours/days	Low
Stability	Ability to maintain performance under load	σ (deviation)	High

Source: developed by the authors

2. Open-source solutions

Among the most common open-source monitoring tools, it is worth highlighting Prometheus for collecting metrics, Grafana for visualization, ELK stack for centralized logging, as well as service mesh components, in particular Istio and Linkerd, which provide distributed request tracing. For a systematic analysis of existing solutions, open-source tools were classified according to their functional purpose (Table 2).

Table 2. *Classification of open-source tools according to their functional purpose*

Category	Tool	Main function	OSI Level
Monitoring (metrics)	Prometheus	Metrics collection and storage	Application (L7)
	Grafana	Visualization and alerting	Application (L7)
Monitoring (logs)	Elasticsearch	Indexing and search	Application (L7)
	Logstash	Collection and transformation	Application (L7)
	Kibana	Log visualization	Application (L7)
Load balancing	HAProxy	High-performance proxy	Transport/Application (L4/L7)
	Nginx	Web server and reverse proxy	Application (L7)
	Traefik	Cloud-native edge router	Application (L7)
Service Mesh	Istio	Full-featured mesh	Application (L7) + mTLS
	Linkerd	Lightweight mesh	Application (L7) + mTLS

Source: developed by the authors based on [2, 7–14]

2.1. Monitoring tools

Prometheus + Grafana. Prometheus implements a pull model for collecting metrics, in which the server periodically polls application endpoints. This approach provides centralized control over the frequency of collection and allows new services to be automatically discovered through the service discovery mechanism [17, 18].

Prometheus stores data in its own time series database (TSDB), optimized for fast recording and aggregation of metrics.

The PromQL query language provides powerful tools for aggregating and analyzing metrics. For example, to calculate the 95th percentile of HTTP request latency over the last 5 minutes, the following query is used:

histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))

This query directly corresponds to the Lp95 metric defined in formula (3). Grafana provides visualization of collected metrics through a flexible dashboard system and supports threshold-based alert configuration.

ELK Stack (Elasticsearch, Logstash, Kibana) provides centralized log management and deep event analytics [3]. Elasticsearch uses an inverted index for fast full-text search, allowing millions of records to be analyzed in seconds. Logstash acts as a data aggregator and transformer, supporting over 200 plugins for various sources. Kibana provides an interface for log visualization and analysis. In terms of log analytics, the focus on the ACM Computing Surveys log review [3] allows ELK to be aligned with modern parsing methods, anomaly detection, and validation datasets.

A comparative overview of the monitored tools is provided in Table 3.

Table 3. *Comparative characteristics of monitoring tools*

Parameter	Prometheus + Grafana	ELK Stack
Data type	Time series metrics	Unstructured and structured logs
Collection model	Pull (HTTP scraping)	Push (via Beats/Logstash)
Query language	PromQL	Lucene / KQL
Storage	TSDB (local, up to 15 days)	Elasticsearch (distributed)
K8s integration	Native (service discovery)	Via Filebeat/Metricbeat
Alerting	Alertmanager (flexible rules)	Watcher / ElastAlert
Resource capacity	Low–medium	High (especially Elasticsearch)
License	Apache 2.0 / AGPL	Elastic License 2.0

Source: developed by the authors based on [2, 3, 17, 18]

2.2. Load balancers

When choosing open-source load balancers at the L4/L7 level (HAProxy, Nginx, Traefik, Envoy), pay attention to key parameters such as latency, throughput, and stability under high load. Practical measurements in server WWW infrastructures showed similar results for HAProxy and Nginx under high loads, with the best performance observed in Linux Virtual Server (LVS approach) [7].

A key aspect of the comparison is the load balancing algorithms, which directly affect the uniformity of request distribution and, accordingly, latency metrics. The main algorithms are formalized as follows.

Round Robin (RR) – cyclic distribution of requests between servers with equal weight:

$$server_{next} = \left(server_{current} + 1\right) \bmod N, \tag{7}$$

where N is the number of available servers.

Least Connections (LC) – selection of the server with the fewest active connections:

$$server_{next} = arg\ min_{i \in \{1..N\}} \left(connections_i\right). \tag{8}$$

HAProxy is optimized for maximum performance and supports operation at the L4 (TCP) and L7 (HTTP) levels. According to studies [7, 8], HAProxy demonstrates the lowest latency among the load balancers considered when operating at the L4 level. Some studies offer

mathematical models for analyzing HAProxy performance, including MMAP/PH/M/N queues and Monte Carlo simulations confirmed by test bench measurements [8].

Nginx is positioned as a universal web server with reverse proxy and load balancing functions. Its asynchronous event-based architecture ensures efficient processing of a large number of concurrent connections. Advanced SSL termination and caching capabilities make it ideal for web-oriented applications [9].

Traefik is designed as a cloud-native edge router with native support for Docker and Kubernetes. Key advantages include automatic service discovery and dynamic configuration without rebooting, as well as built-in integration with Let's Encrypt for automatic TLS certificate management [10]. For Kubernetes environments, Ingress controllers were additionally considered: the 2024 IEEE paper compares implementations and balancing algorithms specifically in the Kubernetes cluster, which directly coincides with our experimental setup [9–11].

A comparative analysis of the studied balancers is presented in Table 4.

Table 4. *Comparative characteristics of load balancers*

Parameter	HAProxy	Nginx	Traefik
OSI level	L4 / L7	L7	L7
Algorithms	RR, LC, Source, URI	RR, LC, IP-hash	WRR, Mirroring
SSL termination	Yes	Yes (extended)	Yes (Let's Encrypt)
Health checks	TCP, HTTP, Agent	Passive, Active	Docker/K8s native
K8s integration	Ingress Controller	Ingress Controller	Native (CRDs)
Configuration	Static (file)	Static (file)	Dynamic (auto-discovery)
Optimization for	Max. throughput	Web services	DevOps/Cloud-native
License	GPL v2	BSD-2	MIT

Source: developed by the authors based on [7–11]

2.3. Service Mesh solutions

Service mesh adds L7 routing, observability, and security policies (mTLS mutual authentication) to the system, but introduces latency and CPU/memory consumption. A detailed analysis of sidecar proxy overhead with measured effects on latency (increase to ~2.7× on certain configurations) and virtual central processing unit (vCPU) [12] shows that the amount of overhead depends heavily on the configuration (TCP proxying vs. protocol parsing) and workload.

Istio is the most feature-rich solution based on the Envoy proxy. It provides complete control over traffic through VirtualService and DestinationRule, including canary deployments, circuit breaking, and fault injection [12]. Research [14] has shown that enabling mTLS increases latency by 15–20%, which corresponds to formula (5) for calculating overhead. The performance of a mesh cluster also depends on related Kubernetes components, such as etcd configuration, which affects the overall behavior of the control plane and the application [15].

Linkerd is positioned as a lightweight service mesh with an emphasis on simplicity and minimal overhead. The linkerd2-proxy, written in Rust, ensures low resource consumption. A comparison of Istio and Linkerd in edge conditions showed Linkerd to be more

"edge-friendly" due to lower overhead costs [13]. A separate technical report on the impact of mTLS in different implementations (Istio, Ambient Istio, Linkerd, Cilium) helps to separate security and performance effects in experimental design [14].

A comparative analysis of Istio and Linkerd is presented in Table 5.

Table 5. *Comparative characteristics of Service Mesh solutions*

Parameter	Istio	Linkerd
Proxy architecture	Envoy (C++)	linkerd2-proxy (Rust)
mTLS	Yes (configured)	Yes (default)
Traffic management	Full (VirtualService, DestinationRule)	Basic (TrafficSplit)
Observability	Metrics, Logs, Traces	Metrics, Traces (golden metrics)
Latency overhead	2–3 ms (up to ~2.7× without mTLS)	<1 ms
CPU overhead (sidecar)	~100–150 mCPU	~20–50 mCPU
RAM overhead (sidecar)	~50–100 MB	~10–20 MB
Complexity of implementation	High	Low
Recommended environment	Enterprise, complex policies	Edge, resource-constrained

Source: developed by the authors based on [12–15]

2.4. Summarizing the results of theoretical analysis

A separate class of works focuses on container platforms and their interaction with balancing/monitoring. Review and applied articles propose load balancing mechanisms in Docker/Swarm/Kubernetes environments for productive and resource-constrained scenarios, including big data and edge [16]. Together with resource consumption models [6], this provides a methodologically correct correlation of observability, balancing, and performance in a microservice SCS.

Based on the analysis, a matrix of tool compatibility with typical usage scenarios (Table 6) has been formed, which allows for a reasoned approach to the selection of solutions for specific SCS.

Table 6. *Matrix of tool compatibility with usage scenarios*

Scenario	HAProxy	Nginx	Traefik	Istio	Linkerd
Highly loaded systems	**+++**	++	+	+	++
Web applications (e-commerce)	++	**+++**	++	+	+
DevOps/CI-CD environments	+	+	**+++**	++	++
Enterprise with security requirements	+	+	+	**+++**	++
Edge/IoT with limited resources	++	+	+	–	**+++**

Note: +++ – best solution; ++ – good; + – acceptable; – not recommended
Source: developed by the authors

Thus, the work aims to bridge the gap between the theoretical capabilities of open-source tools and their actual effectiveness when applied in SCS.

The theoretical analysis allows us to formulate hypotheses about the expected results of the experimental study:

1. HAProxy will demonstrate the lowest latency values (p95, p99) among load balancers due to optimization for L4 operation;

2. Istio with mTLS enabled will have the highest resource overhead due to additional encryption operations;

3. Linkerd will provide the optimal balance of functionality and overhead for edge environments;

4. Traefik will demonstrate the greatest ease of integration into the Kubernetes environment thanks to its auto-discovery mechanism.

These hypotheses are tested in the experimental part of the study, the methodology and results of which are presented in the next section.

3. Testing the microservice application

To test the effectiveness of open-source solutions in the field of monitoring and load balancing, a test environment was created that replicates the typical operating conditions of an SCS with a microservice architecture.

Description of the environment. The test application consisted of a set of microservices implemented in Docker containers and deployed in a Kubernetes cluster. The architecture included an authentication service, a user management service, a business logic service, and a database accessible through a separate data access service.

To reproduce a real-world load scenario, the application was wrapped in an API gateway that provided external access to the system.

Tool integration. The following open-source solutions were gradually integrated into the cluster:

– Prometheus + Grafana – for collecting and visualizing metrics (latency, throughput, CPU, RAM, network I/O);

– ELK stack – for centralized log collection and analysis;

– HAProxy, Nginx, and Traefik – as external load balancers at the L4 and L7 levels;

– Istio and Linkerd – as service mesh implementations for internal load balancing and traffic monitoring.

Testing methodology. Apache JMeter and k6 tools were used to generate load, which allowed us to reproduce various user activity scenarios. Testing was conducted in two modes:

1. A stable environment with a uniform load over a long period of time (baseline tests).

2. Stress tests with sharp peaks in request intensity, simulating peak periods of operation.

Metrics and evaluation criteria. System performance was evaluated based on the following indicators:

– latency (average, p95, p99);

– throughput (number of requests per second);

– resource consumption (CPU, RAM) by monitoring and balancing tools;

– operational efficiency, i.e., ease of tool integration and speed of configuration.

Experiment scenarios. For each tool, a series of tests was conducted with varying parameters:

– in the case of HAProxy/Nginx/Traefik, round-robin, least-connections, and IP-hash algorithms were compared;

– in the case of Istio/Linkerd, the impact of enabling mTLS, retry mechanisms, and rate limiting policies was evaluated;

– for Prometheus/Grafana and ELK stack – the overhead of data collection at different intervals and log volumes was analyzed.

Expected results. The experiment was expected to yield quantitative data that would allow comparing the effectiveness of different approaches to balancing and monitoring. In particular, we planned to determine:

– which tools provide the least overhead under high load;

– which configurations provide the optimal balance between performance and management flexibility;

– how much the implementation of a service mesh affects delays compared to classic load balancers.

Fig. 1 shows a diagram of the experimental microservices environment, which shows the user, load balancers (HAProxy/Nginx/Traefik), API Gateway, services, database, as well as the integration of monitoring tools (Prometheus + Grafana, ELK) and service mesh (Istio/Linkerd).

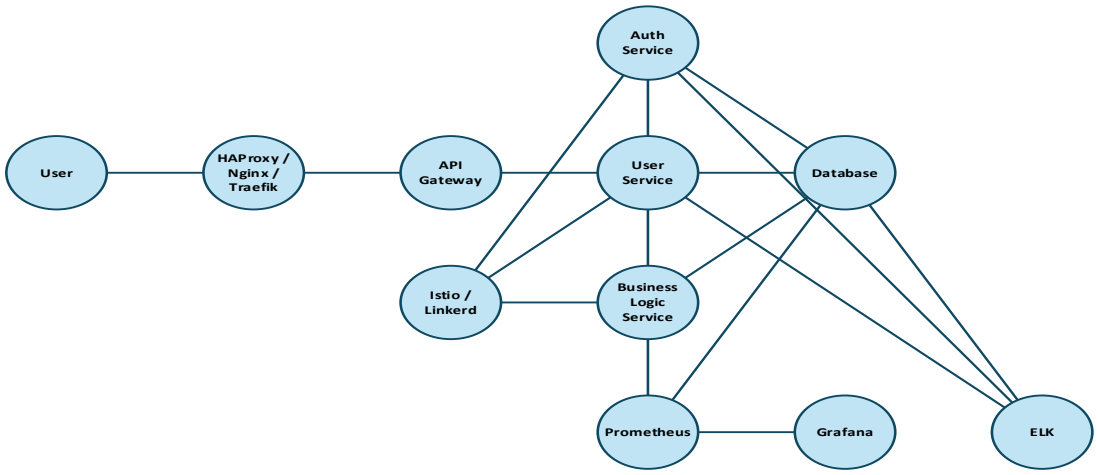

Fig. 1. Diagram of the experimental microservices environment
Source: developed by the authors

During experimental research, a series of tests was performed in two scenarios: stable load (Baseline) and stress tests with periodic intensity peaks (more than 600 measurements were performed).

1) Scenario characteristics

In baseline mode, the system demonstrated an average intensity of ≈1500 requests per second (RPS) with fluctuations of no more than ±5%. The average p95 latency was 90±3 ms, p99 – 130±5 ms, which

corresponds to normal operating mode. During stress tests, the intensity ranged from 1400 to 3500 RPS, and p95 delays increased by 20–40 ms, p99 – by 30–70 ms, depending on the balancing tool (Fig. 2).

This confirms the typical pattern of performance degradation under peak loads. This behavior allowed us to evaluate the ability of the tools to maintain stability during short-term overloads.

The graph in Fig. 3 shows that in the baseline scenario, p95 remained stable (≈90 ms), while in the stress scenario, it increased to 120–140 ms at peaks. This indicates a typical response of microservice architecture to uneven load, where some services become a "bottleneck".

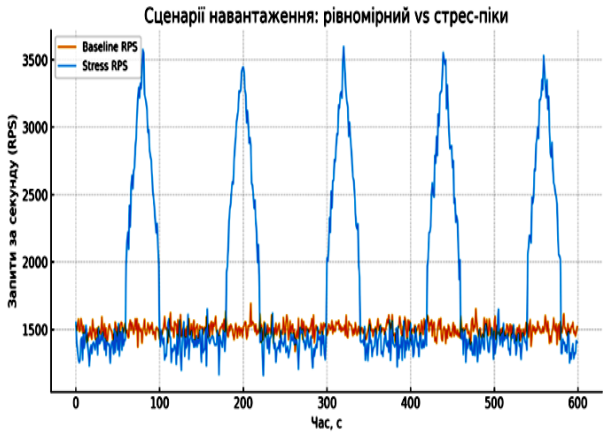

Fig. 2. RPS load scenarios over time
Source: developed by the authors

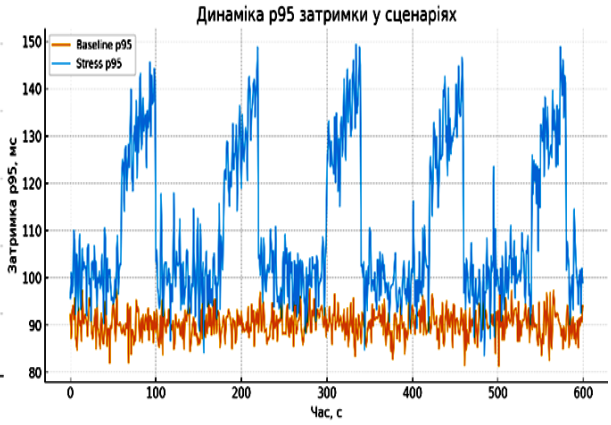

Fig. 3. Dynamics of p95 delays (baseline vs stress)
Source: developed by the authors

2) Comparison of tools

All tools showed differences in both average values (Fig. 4) and stability of results (Fig. 5) when averaged over a 10-minute test.

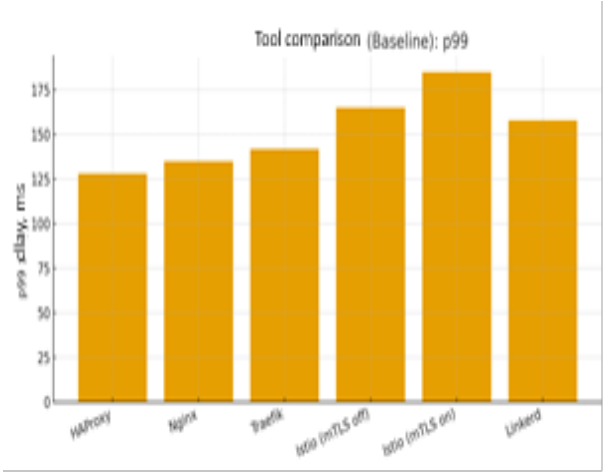

Fig. 4. Comparison of tools (Baseline p99)
Source: developed by the authors

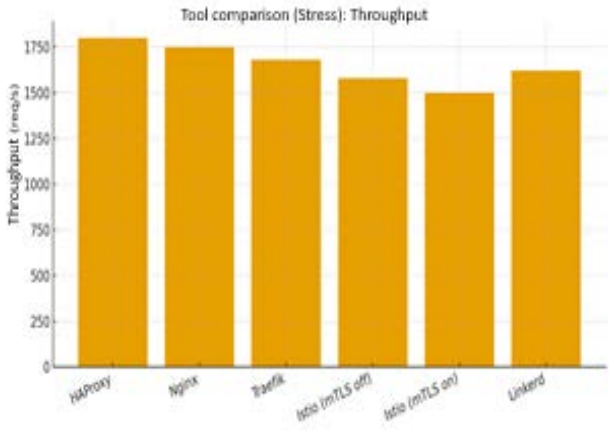

Fig. 5. Comparison of tools by throughput in a stress scenario
Source: developed by the authors

HAProxy showed the best results for latency (p95 = 88±2 ms; p99 = 128±4 ms in baseline) and maintained the lowest values even under stress load (p95 = 108±4 ms). Throughput remained stable (≈1800 RPS at peaks).

Nginx was slightly inferior to HAProxy, especially in terms of p99 (135±5 ms), but provided greater flexibility in routing, which is important for web-oriented services.

Traefik showed greater overhead: p99 in baseline – 142±5 ms, in stress tests – 190±7 ms, but stood out for its ease of integration with Kubernetes and automation of TLS certification.

Linkerd showed the best balance between performance and resource consumption (baseline p95 ≈ 105±3 ms; p99 ≈ 158±6 ms), making it suitable for edge environments.

Istio showed higher latencies (baseline p95 = 110±4 ms; p99 = 165±6 ms), and even higher with mTLS enabled (p95 = 125±5 ms; p99 = 185±8 ms).

However, its functionality (detailed access policies, security, monitoring) significantly exceeds that of alternatives.

The effectiveness of open-source tools for monitoring and load balancing depends on the usage scenario and the requirements of the specialized computer system.

3) Resource overhead costs

The data confirm (Table 7) that in the baseline scenario, the difference between the instruments is less noticeable, but during stress loading, the differences become significant.

Table 7. *Summary results by tools*

Tool	Baseline p95 (ms)	Baseline p99 (ms)	Baseline RPS	Stress p95 (ms)	Stress p99 (ms)	Stress RPS	CPU overhead (%)	RAM overhead (MB)
HAProxy	88	128	1500	108	165	1800	6.5	180
Nginx	92	135	1470	116	178	1750	7.2	220
Traefik	95	142	1420	122	190	1680	8.0	260
Istio (mTLS off)	110	165	1350	140	225	1580	12.5	420
Istio (mTLS on)	125	185	1300	165	260	1500	16.0	520
Linkerd	105	158	1380	135	210	1620	10.5	360

Source: developed by the authors

The overhead graphs (Fig. 6; 7) show:

– the lowest CPU and RAM consumption was recorded in HAProxy (≈6.5% CPU, 180 MB RAM) and Nginx (≈7.2% CPU, 220 MB RAM);

– Traefik required more resources (≈8% CPU, 260 MB RAM), which is explained by the dynamic nature of the configuration;

– Linkerd had an average overhead (≈10.5% CPU, 360 MB RAM);

– Istio created the most load, especially with mTLS (≈16% CPU, 520 MB RAM), which can be critical in resource-constrained environments.

No tool is universal; the choice should be made based on the balance between performance, functionality, and resource capabilities of a particular architecture.

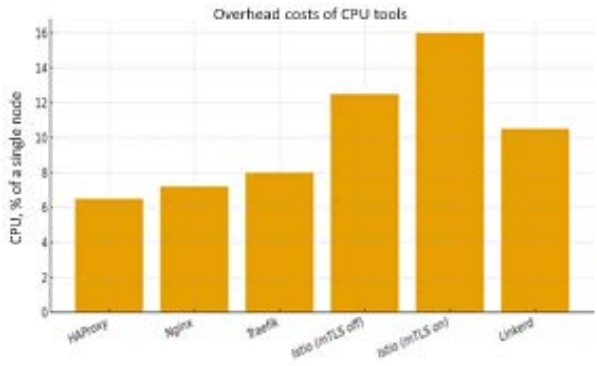

Fig. 6. Overhead CPU
Source: developed by the authors

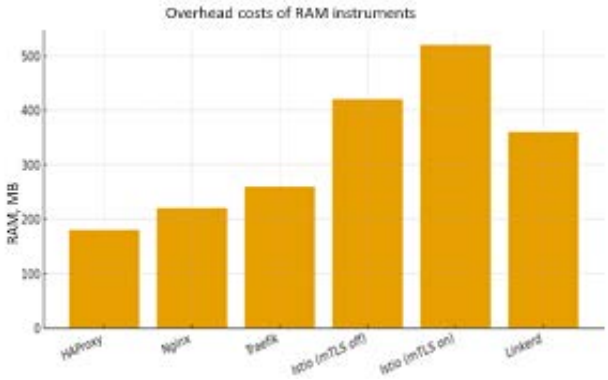

Fig. 7. Overhead RAM
Source: developed by the authors

Conclusions

Thus, a study was conducted that confirmed the relevance of using open-source solutions for monitoring and load balancing in microservice applications operating in SCS. The increasing complexity of architectural solutions and the demand for high performance require tools that can simultaneously provide observability, flexible traffic management, and efficient resource utilization.

Analysis of theoretical aspects showed that monitoring and balancing remain key elements in supporting the performance of microservice environments. The use of the Prometheus + Grafana combination allows for the effective collection and visualization of metrics, while the ELK stack provides centralized log management and deep event analytics. The HAProxy, Nginx, and Traefik balancing tools demonstrated different strengths: from maximum performance (HAProxy) to flexible routing (Nginx) and ease of integration into DevOps processes (Traefik). The Istio and Linkerd service mesh solutions showed advanced traffic and security management capabilities, but at the cost of higher resource overhead.

The experimental part of the study made it possible to evaluate the real effectiveness of using various open-source tools in microservice architectures. In stable load scenarios, HAProxy demonstrated the lowest latency and highest stability, confirming its focus on high performance and optimization for handling large numbers of concurrent connections. This balancer demonstrated the ability to maintain stable throughput even under heavy load conditions, ensuring low p95 and p99 latency values.

In situations with sharp load spikes, HAProxy also remained the most stable, while Istio with mTLS enabled showed a noticeable degradation in performance.

This is due to the additional overhead of traffic encryption and access policy management, which significantly impacted latency and system throughput. Nginx and Traefik took intermediate positions, providing a compromise between performance and functionality.

Nginx proved to be convenient for web-oriented systems, as it provides flexible routing capabilities and supports SSL termination. Traefik, in turn, stood out for its ease of integration with Kubernetes and automatic TLS certificate management, making it attractive for DevOps environments.

Linkerd deserves a special mention, as it showed the best balance of performance and resource efficiency compared to Istio. Although it has slightly less functionality, it provides lower latency and less overhead, making it suitable for resource-constrained environments or edge systems.

The results of the experimental study made it possible not only to evaluate the performance of open-source tools, but also to formulate a number of practical recommendations for their application in real-world conditions.

For highly loaded specialized systems, where the primary task is to minimize latency and ensure maximum stability during peak loads, HAProxy is the most appropriate choice. This balancer showed the best p95 and p99 performance and confirmed its ability to maintain high throughput without critical degradation.

For web-oriented services that serve end users and require complex routing rules, Nginx is the optimal solution. Its ability to perform

SSL termination and support for advanced HTTP routing scenarios makes it convenient for e-commerce, portals, or online services.

For DevOps environments focused on containerization and CI/CD process automation, Traefik is the most effective. Thanks to its dynamic integration with Kubernetes and Docker, it greatly simplifies traffic management.

Finally, Linkerd has proven itself suitable for resource-constrained environments where a balance between performance and resource efficiency is important. At the same time, Istio should be used in large enterprise systems with high requirements for security, access control policies, and comprehensive monitoring, even despite its higher overhead.

Thus, the results of the study confirm that open-source tools are capable of providing effective monitoring and load balancing in microservice architectures. The choice of a specific solution should be based on a balance between performance, functionality, and resource constraints of a specialized computer system.

The analysis outlined three key areas for further research aimed at improving the efficiency of SCS:

1. Modeling and minimizing service mesh overhead. It is critically important to develop detailed mathematical models for accurate prediction of sidecar proxy overhead (Istio, Linkerd) in latency-critical environments, taking into account dynamic configurations (mTLS, L7 filters), as well as comparisons with new architectures such as Ambient Mesh.

2. Development of adaptive load balancing algorithms driven by metrics (Metric-Aware Balancing). It is necessary to implement dynamic routing that uses real-time service quality metrics from Prometheus, ensuring the transition to service quality balancing.

3. Application of machine learning (ML) methods for proactive monitoring and scaling management. ML integration will enable the creation of models to predict performance degradation and detect anomalies in logs (ELK stack) before a failure occurs, which is critical for improving the resilience of modern SCS.

References

1. Waseem, M., Liang, P., Shahin, M., Di Salle, A., Márquez, G. (2021), "Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective", *Journal of Systems and Software*, No. 182, P. 111061. DOI: https://doi.org/10.1016/j.jss.2021.111061
2. Giamattei, L., Guerriero, A., Pietrantuono, R., et al. (2024), "Monitoring tools for DevOps and microservices: A systematic grey literature review", *Journal of Systems and Software*, No. 208, P. 111906. DOI: https://doi.org/10.1016/j.jss.2023.111906
3. He, S., Zhu, J., He, P., Lyu, M.R. (2021), "A Survey on Automated Log Analysis for Reliability Engineering", *ACM Computing Surveys*, No. 54(6), P. 123. DOI: https://doi.org/10.1145/3460345
4. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J. (2016), "Borg, Omega, and Kubernetes", *Communications of the ACM*, No. 59(5), P. 50–57. DOI: https://doi.org/10.1145/2890784
5. Amaral, M., Polo, J., Carrera, D., et al. (2015), "Performance Evaluation of Microservices Architectures using Containers", *IEEE NCA*, P. 93–100. DOI: https://doi.org/10.1109/NCA.2015.49
6. Turin, G., Borgarelli, A., Donetti, S., Damiani, F., Johnsen, E.B., Tapia Tarifa, S.L. (2023), "Predicting Resource Consumption of Kubernetes Container Systems Using Resource Models", *Journal of Systems and Software*, No. 203, P. 111750. DOI: https://doi.org/10.1016/j.jss.2023.111750
7. Dymora, P., Mazurek, M., Sudek, B. (2021), "Comparative Analysis of Selected Open-Source Solutions for Traffic Balancing in Server Infrastructures Providing WWW Service", *Energies*, No. 14(22), P. 7719. DOI: https://doi.org/10.3390/en14227719
8. Sokolov, A. (2024), "Application of Queueing Theory to Investigation of HaProxy Load Balancer Performance Characteristics", DCCN 2023, CCIS 2129, Springer, P. 89–100. DOI: https://doi.org/10.1007/978-3-031-61835-2_7
9. Rathi, G., Amin, S. (2024), "Performance Analysis of Different Ingress Controllers Within the Kubernetes Cluster", *IEEE ICITEICS 2024*. DOI: http://dx.doi.org/10.1109/ICITEICS61368.2024.10625280
10. Khamdani, A.R., Muslikh, A.R., Affandi, A.S. (2025), "Comparative Analysis of Performance and Efficiency of Load Balancing Algorithms on Ingress Controller", *Jurnal Teknik Informatika*, Vol. 6, No. 1, P. 453–468. DOI: https://doi.org/10.52436/1.jutif.2025.6.1.4040
11. Nguyen, N., Kim, T. (2020), "Toward Highly Scalable Load Balancing in Kubernetes Clusters", *IEEE Communications Magazine*, No. 58(7), P. 78–83. DOI: https://doi.org/10.1109/MCOM.001.1900660
12. Zhu, X., She, G., Xue, B., et al. (2023), "Dissecting Overheads of Service Mesh Sidecars", ACM SoCC '23: Proceedings of the 2023 ACM Symposium on Cloud Computing, P. 142–157. DOI: https://doi.org/10.1145/3620678.3624652
13. Elkhatib, Y., Salmon, B., Harkous, H., et al. (2023), "An Evaluation of Service Mesh Frameworks for Edge Systems", *EdgeSys '23: Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking*, P. 19–24. DOI: https://doi.org/10.1145/3578354.3592867
14. Bremler-Barr, A., Lavi, O., Naor, Y., Rampal, S., Tavori, J. (2024), "Performance Comparison of Service Mesh Frameworks: the mTLS Test Case", arXiv (Tech. Report). DOI: https://doi.org/10.48550/arXiv.2411.02267

15. Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., Kihl, M. (2020), "Impact of etcd Deployment on Kubernetes, Istio, and Application Performance", *Software: Practice and Experience*. DOI: https://doi.org/10.1002/spe.2885
16. Singh, N., Tanwar, S., Gupta, R., Kumar, N. (2023), "Load balancing and service discovery using Docker Swarm for big data applications in microservice architecture", *Journal of Cloud Computing*, No. 12, P. 77. DOI: https://doi.org/10.1186/s13677-022-00358-7
17. Jani, Y. (2024), "Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions", *Journal of Artificial Intelligence*, Machine Learning and Data Science, No. 2(1), P. 848–852. DOI: http://dx.doi.org/10.51219/JAIMLD/yash-jani/206
18. Elrad, M.D. (2025), "Prometheus & Grafana: A Metrics-focused Monitoring Stack", *Journal of Computer Allied Intelligence*, No. 3(3), P. 28–39. DOI: https://doi.org/10.69996/jcai.2025015

Received (Надійшла) 07.11.2025
Accepted for publication (Прийнята до друку) 03.12.2025
Publication date (Дата публікації) 28.12.2025

About the Authors / Відомості про авторів

Glavchev Maksym – PhD (Economic Sciences), Associate Professor, National Technical University "Kharkiv Polytechnic Institute", Professor at the Department of Computer Engineering and Programming, Kharkiv, Ukraine; e-mail: Maksym.Glavchev@khpi.edu.ua; ORCID ID: https://orcid.org/0000-0001-9670-9118

Hlavchev Dmytro – PhD, National Technical University "Kharkiv Polytechnic Institute", Associate Professor at the Department of Computer Engineering and Programming, Kharkiv, Ukraine; e-mail: Dmytro.Hlavchev@khpi.edu.ua; ORCID ID: https://orcid.org/0000-0003-4248-4819

Panchenko Volodymyr – National Technical University "Kharkiv Polytechnic Institute", Senior Lecturer at the Department of Computer Engineering and Programming, Kharkiv, Ukraine; e-mail: Volodymyr.Panchenko@khpi.edu.ua; ORCID ID: https://orcid.org/0000-0003-3364-3398

Главчев Максим Ігорович – кандидат економічних наук, доцент, Національний технічний університет "Харківський політехнічний інститут", професор кафедри "Комп'ютерна інженерія та програмування", Харків, Україна.

Главчев Дмитро Максимович – доктор філософії, Національний технічний університет "Харківський політехнічний інститут", доцент кафедри "Комп'ютерна інженерія та програмування", Харків, Україна.

Панченко Володимир Іванович – Національний технічний університет "Харківський політехнічний інститут", старший викладач кафедри "Комп'ютерна інженерія та програмування", Харків, Україна.

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВПРОВАДЖЕННЯ *OPEN-SOURCE*-РІШЕНЬ ДЛЯ МОНІТОРИНГУ Й БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В МІКРОСЕРВІСНИХ ЗАСТОСУНКАХ

Предметом дослідження є *open-source*-рішення та їх упровадження для моніторингу й балансування навантаження в мікросервісних застосунках, що функціонують у спеціалізованих комп'ютерних системах. Дослідження охоплює широкий спектр інструментів, зокрема системи збору метрик, централізованого логування й розподіленого трасування запитів. Актуальність роботи зумовлена постійним зростанням складності розподілених архітектур і критичною потребою в ефективному контролі продуктивності (спостережуваності) й стабільному розподілі трафіку. **Мета роботи** – комплексне емпіричне оцінювання й порівняння ключових *open-source*-інструментів моніторингу й балансування навантаження, зокрема *Prometheus / Grafana* (для метрик), *ELK stack* (для логів), *HAProxy, Nginx, Traefik* (балансувальники), а також *Istio* та *Linkerd* (*service mesh*), з метою розроблення практичних рекомендацій щодо проєктування й експлуатації мікросервісних систем. **Завдання:** проаналізувати поширені *open-source*-інструменти; визначити критерії їх ефективності; створити тестове середовище на базі *Kubernetes*; провести серію навантажувальних тестів з різними конфігураціями; кількісно оцінити ключові показники продуктивності, зокрема затримку, пропускну здатність і використання ресурсів. **Методи дослідження.** Застосовано системний аналіз, емпіричне моделювання та бенчмаркінг. Для об'єктивного порівняння впроваджено методи навантажувального тестування (*baseline* та стрес-сценарії) в кластері *Kubernetes*. Ключові критерії оцінювання: затримка оброблення запитів, пропускна здатність і ресурсні накладні витрати самих інструментів. **Результати** підтверджують, що *open-source*-рішення здатні забезпечити високий рівень спостережуваності та ефективне балансування навантаження в спеціалізованих комп'ютерних системах, водночас залишаючись економічно вигідною альтернативою комерційним продуктам. Дослідження виявило переваги й недоліки кожного з інструментів, що дає змогу обґрунтовано підходити до їх вибору залежно від специфічних вимог проєкту. **Висновки:** підтверджено здатність *open-source*-інструментів ефективно забезпечувати спостережуваність і управління навантаженням у спеціалізованих комп'ютерних системах і водночас залишатися економічно вигідною альтернативою комерційним продуктам. Сформульовані висновки дають змогу розробити практичні рекомендації для проєктування й експлуатації мікросервісних застосунків із зосередженням на стабільності та продуктивності. Результати дослідження можуть бути впроваджені в прийнятті архітектурних рішень для розподілених систем різного масштабу.

Ключові слова: *open-source*; *service mesh*; мікросервіс; моніторинг; балансування навантаження; спеціалізована комп'ютерна система.

Bibliographic descriptions / Бібліографічні описи

Glavchev, M., Hlavchev, D., Panchenko, V. (2025), "Evaluating the effectiveness of open-source solutions for monitoring and load balancing in microservice applications", *Management Information Systems and Devises*, No. 4 (187), P. 182–199. DOI: https://doi.org/10.30837/0135-1710.2025.187.182

Главчев М. І., Главчев Д. М., Панченко В. І. Дослідження ефективності впровадження *open-source*-рішень для моніторингу й балансування навантаження в мікросервісних застосунках. *Автоматизовані системи управління та прилади автоматики*. 2025. № 4 (187). С. 182–199. DOI: https://doi.org/10.30837/0135-1710.2025.187.182