УДК 004.421:004.438 **DOI: 10.18372/2073-4751.83.20551**

Alexander Chizhov,
orcid.org/0000-0002-3992-8522
Andriy Fesenko,
orcid.org/0000-0001-5154-5324

INTELLIGENT LOAD BALANCING MANAGEMENT IN CLOUD WEB HOSTING: EVALUATION CRITERIA AND METHODOLOGY

State University "Kyiv Aviation Institute"

e-mail: 8288253@stud.kai.edu.ua,
e-mail: andrii.fesenko@npp.kai.edu.ua

Introduction

The migration from the classical shared hosting model to cloud-cluster architectures is becoming a necessity for providers that serve large portfolios of low-loaded websites. Studies show that hybrid solutions with elastic scaling make it possible to simultaneously reduce operational costs and improve service quality during peak traffic periods [1]. Under these conditions, the load balancer stops being a secondary L4/L7 element and becomes a managed component that affects the fulfillment of the Service Level Agreement (SLA) and the economics of resource usage.

A key feature of cluster web hosting is massive Transport Layer Security (TLS) with tens of thousands of domains and a high frequency of certificate rotation. At this scale, support for “hot” management of TLS secrets without restarting processes is critical. Modern solutions provide corresponding mechanisms (for example, the Runtime API in HAProxy [2] and SDS/xDS in Envoy [3]), but their impact on stability, latency, and the predictability of system behavior under dynamic configuration changes remains insufficiently studied in typical testing methodologies.

The second defining aspect of the architecture is the integration of the load balancer with an asynchronous traffic analysis circuit. The approach with placing a dynamic blacklist in memory and moving request classification to a queue and a Machine Learning (ML) processor makes it possible to maintain high throughput with zero additional latency on the “hot” path. This principle is formalized by the authors in the registered Ukrainian utility model [4], and is also examined in detail in the authors’ study [5], where a cluster hosting with neural network filtering of HTTP Flood and Brute Force attacks was proposed. These aspects define specific requirements for extension points, secure interfaces for managing block lists, and efficient non-blocking data structures.

To ensure the required SLA, the load balancer must also act as a source of telemetry and a feedback node for scaling mechanisms: aggregate load metrics by domains and time windows, export them in standard formats (for example, Prometheus or StatsD), and provide universal interfaces such as Command Line Interface (CLI), HyperText Transfer Protocol Application Programming Interface (HTTP API), or similar for initiating topology changes. In this way, the load balancer becomes part of the cluster control loop, where resource supply parameters are calculated by a mathematical model based on observed data.

Existing academic and engineering comparisons usually focus on throughput and latency in static configurations, without taking into account the scale of TLS, frequent “hot” changes, and the ability to embed user logic. This creates a methodological gap between the practical requirements of cloud-cluster platforms and the performance benchmarks being used. This work aims to close this gap: we develop a methodology for choosing a load balancer

for cluster web hosting with massive TLS and asynchronous traffic analytics, highlighting criteria of suitability for dynamic operation scenarios and requirements for observability and manageability.

The article is structured as follows. In the section "Problem Statement", the functional requirements for the load balancer in the considered architecture are clarified. In the "Literature Review", the limitations of existing comparison methodologies are summarized, and the capabilities of modern solutions relevant to the dynamic context are identified. Further, selection criteria are formulated and a methodology for comparative analysis is proposed, oriented toward scenarios with a changing number of certificates, asynchronous filtering, and autoscaling; questions of experimental validation are placed in future publications.

Purpose and Structure of the Study

The purpose of this work is to develop a methodology for choosing a software load balancer for a cloud-cluster web hosting platform with massive use of TLS, dynamic certificate rotation, and integration with an asynchronous traffic analysis system. Unlike traditional approaches based on measuring throughput and latency in static configurations, the proposed approach is oriented toward dynamic operation scenarios, where the number of certificates, the state of the blacklist, and the size of the cluster change.

To achieve this purpose, the following tasks are solved:

- define the functional requirements for the load balancer as a control element in the architecture of cluster web hosting;
- analyze scientific publications and technical documentation on the most commonly used software load balancers HAProxy, NGINX, Envoy, Traefik, and Caddy, in terms of dynamic TLS management, extensibility, and telemetry;
- formulate criteria for the suitability of the load balancer for use under conditions of configuration dynamics, asynchronous analytics, and SLA-oriented scaling;
- propose a methodology for comparative analysis that takes into account the change in the number of certificates, TLS rotation, and real-time blacklist updates.

The practical implementation of the experimental environment and the quantitative validation of the methodology are beyond the scope of this article and will be presented in future works.

Problem Statement

In modern cluster web hosting systems, the load balancer becomes not just a transport node but a critical component that determines infrastructure stability, SLA compliance, and the possibility of dynamic scaling. However, most existing studies evaluate its behavior in static conditions, where the number of domains is fixed, the configuration does not change over time, and TLS certificates are loaded only once at startup. This approach contradicts the practical realities of cloud-cluster platform operation.

One of the main problems is the scale of the TLS environment. Large hosting providers work with tens of thousands of domains, each requiring its own certificate. These certificates are not static, new clients appear daily, old ones stop working, automatic renewal through Automatic Certificate Management Environment (ACME) is triggered, or internal key rotation is performed. The load balancer is forced to operate in conditions where the TLS configuration continuously changes. In such scenarios, classical configuration reload mechanisms, accompanied by short delays or connection loss, become unacceptable. With a large number of certificates, memory usage also increases, TLS handshake time grows, and the risk of latency degradation rises, especially during peak client reconnections.

The complexity is increased by the need to perform "hot" configuration changes without stopping processes. The load balancer must update certificates, add or remove backend servers, adjust request routing and all of this in a running system

that must not interrupt service. However, typical testing methodologies for these capabilities either do not exist or are limited to subjective engineering observations. It is unclear how quickly the load balancer applies changes, whether connections are lost at the moment of switching configuration, and whether latency increases during internal state rebuilding.

Special attention must be given to integration with security systems. In the authors' patented model of asynchronous traffic filtering [4], the request is processed by the load balancer immediately, but its copy is sent to a queue for analysis by a neural network. The result of the analysis becomes an update of the blacklist, a list of IP addresses that must be denied service in the future. This operation is performed without stopping the process and without additional delay for the main traffic. Such an architecture creates specific requirements: the load balancer must be able to modify in-memory data structures in real time, support interaction with queue systems (Redis, Kafka, RabbitMQ), and at the same time not lose throughput. In existing literature, this type of problem is almost never addressed, research usually focuses on attack detection models, but not on how the analysis results should be applied directly at the traffic entry point.

Scaling management creates another group of problems. In an elastic cluster, the number of servers must change automatically depending on the load. The load balancer becomes a node that either initiates changes or, at minimum, reacts correctly to them: adds new nodes to the pool, excludes unavailable ones, redistributes traffic without breaking connections. However, the behavior of load balancers in transitional modes, for example, when dozens of backend servers are added at the same time or when part of the infrastructure fails is almost not studied. An open question remains: which configuration management mechanisms are reliable, what is the reaction delay to changes, and is transactional configuration change and rollback in case of error possible.

Finally, full SLA management is impossible without telemetry. Mathematical models that determine the optimal number of active servers require reliable data on the number of requests, their distribution by domains, latency, errors, and the number of active connections. The load balancer must be not only a participant in network exchange but also a source of observable metrics suitable for processing by external systems (Prometheus, StatsD, etc.). The problem is that not all solutions provide such telemetry with the required accuracy and frequency, and in existing research this aspect is almost not considered.

Thus, the research problem is formed: it is necessary to identify architectural limitations that arise during the operation of load balancers under conditions of massive TLS, hot configuration changes, asynchronous traffic filtering, and dynamic cluster management, and based on these limitations, formulate requirements for a methodology of objective selection and further comparison of load balancers in a cloud-cluster environment.

Literature Review

Load balancers are traditionally considered in research as components expected to provide high throughput, low latency, and stability under load. Most works measure Requests Per Second, latency metrics at the 95th and 99th percentiles (p95 and p99 latency), that is the response time below which 95% and 99% of all requests fall, the number of errors, and system behavior when the number of simultaneous connections increases [6,7]. This approach was justified while web hosting remained a static environment: configuration changes were rare, certificates were loaded manually, and the infrastructure scale was measured in tens of servers. However, in cloud-cluster environments, these assumptions are no longer sufficient. The number of domains reaches thousands, certificates are updated dynamically, and the cluster topology changes in real time. Despite this, most comparative studies continue to evaluate load balancers as

immutable systems that do not account for dynamic configuration changes.

Some publications note that test results are sensitive to configuration and TLS mode [7], but do not include scenarios where certificates are added, removed, or updated during service operation. Situations where the composition of backend servers, TLS configuration, and traffic volume change simultaneously are rarely considered. What has long become the norm in real systems still remains outside the focus of academic literature. Researchers continue to compare NGINX, HAProxy, or Envoy with a static set of domains, while hosting providers operate in a constantly changing enviro-nment where a static state exists only for a moment.

Engineering sources, official documentation for HAProxy, Envoy, NGINX, Traefik, and Caddy, show that developers of load balancers have long implemented mechanisms for dynamic TLS management. HAProxy uses a Runtime API that allows updating the list of certificates without restarting the process [2]. Envoy applies the Secret Discovery Service (SDS) within the xDS architecture [3], providing centralized real-time distribution of keys. NGINX offers graceful reload, which updates the configuration without breaking connections, but this approach implies a full reload of the master process [8]. Traefik and Caddy automate certificate issuance and renewal via ACME and allow configuration changes through the API [9,10]. These mechanisms exist and are actively used, but their impact on latency, stability, and SLA has been practically unstudied in academic works.

A similar situation is observed in relation to embedded request-processing logic. Modern load balancers have long ceased to be universal "black boxes." HAProxy includes the Stream Processing Offload Engine (SPOE), which allows transferring data to external agents and returning processing results in real time [2]. Envoy develops the concept of WebAssembly filters, allowing the embedding of business logic at a low level of traffic processing [11]. NGINX supports extensions through Lua or njs, enabling integration of authorization, tracing, and traffic filtering systems [8]. However, publications where such mechanisms are considered not as technical possibilities but as measurable parameters are almost absent. There are no works evaluating how the introduction of such logic affects latency, throughput, or stability under high load.

Another direction of literature is devoted to machine learning and L7-DDoS detection methods. These studies describe how neural networks classify HTTP requests, detect behavioral anomalies, and form dynamic block lists. However, most research focuses on model accuracy, datasets used, and standard metrics: precision (the share of correctly detected malicious requests among all blocked ones) and recall (the share of detected attacks among all actually existing attacks) [12,13]. At the same time, almost no attention is paid to how such solutions should be integrated directly into the load balancer or web server. It is not examined how quickly a blacklist can be updated in memory, whether this affects packet latency, or what happens when configuration updates and new load arrivals occur simultaneously. In other words, the research focuses on developing the "intelligence" of the system but does not explain how to embed it into the real transport architecture of traffic processing.

Taken together, this forms a clear scientific gap. On one hand, researchers describe load balancer performance, but in static conditions. On the other hand, documentation and engineering practice show the presence of complex mechanisms for TLS dynamics, hot reload, and programmable traffic processing, but these aspects do not appear in academic comparative works. A third direction, ML and DDoS analytics, shows how attacks can be detected but does not describe how analysis results are applied at the load balancer level. There is not a single work that combines TLS scale, hot certificate management, asynchronous request filtering,

and SLA-oriented cluster control into a unified methodology.

Eliminating this methodological gap is precisely the goal of this article. Unlike existing studies, here the load balancer is considered not as a static element of the network but as a dynamic control node whose behavior must be evaluated under conditions of configuration, load, and security changes.

Criteria for Selecting a Load Balancer

The problems outlined in the problem statement require moving from the description of bottlenecks to the formation of a system of criteria by which software load balancers can be objectively compared. If the "Problem Statement" section identifies what the architecture faces, this section defines what exactly should be measured and analyzed to distinguish a suitable solution from an unsuitable one. Each criterion describes not only a property of the load balancer but also how this property can be tested and verified.

1. Operation with TLS at Large Scale

The first aspect to evaluate is the ability of the load balancer to handle not tens but thousands and tens of thousands of certificates simultaneously. Unlike the problem statement, this is not about the fact that there are many certificates and they change, but about how well a specific solution can handle such a configuration without increasing latency or failures. Practical evaluation includes measuring TLS material loading time, memory consumed by certificates, TLS handshake execution time, and behavior during sequential dynamic loading of new domains without restarting the system. If adding the hundredth or thousandth certificate causes a response time spike or process restart, the load balancer does not meet the criterion.

2. Reaction to Configuration Changes During Operation

A load balancer intended for use in cluster environments must be able to accept new settings without interrupting service. However, the criterion here is not simply the presence of an API or reload command, but how quickly and safely it applies changes. It becomes important to measure the time between issuing a command and the actual application of changes, to record possible connection losses, delays, or short "blind spots" when part of the traffic is processed with the old configuration. The evaluation should be performed on a sequence of changes: one certificate, ten, one hundred; adding and removing backend servers; updating routing. Only in this way can the suitability of the solution for real dynamic scenarios be judged.

3. Architectural Flexibility and the Ability to Embed Additional Logic

The load balancer becomes part not only of the network but also of the logical architecture; therefore, its ability to embed custom request processing is evaluated. This is not about the fact that it can work with queues or Lua scripts, but about how fast such processing executes, how much it interferes with the main request path, and whether the load balancer can update internal data structures, such as block tables, without thread blocking. Verification is carried out experimentally: delays are measured with filters enabled (Wasm, Lua, njs), additional CPU load is recorded, and performance changes are analyzed during mass updates of in-memory records, for example, during intensive IP table updates.

4. Manageability and Interaction with External Systems

Unlike the problem of the absence of unified management standards, the criterion here is the measurable ability of the load balancer to integrate into infrastructure orchestration. This includes the presence of an API or CLI, but more importantly is the reliability of these interfaces. The test must determine whether a cluster node can be automatically added, how the load balancer reacts to a lost connection with the API, and whether rollback of changes is supported. In addition, compatibility with DevOps tools is evaluated: whether automatic configuration

via Ansible, Terraform, or CI/CD pipelines is possible; whether changes are recorded transactionally; and whether configuration is blocked during concurrent access.

5. Observability and Telemetry

An important criterion is not only the availability of statistics but also its suitability for SLA analytics. The load balancer must provide data on the number of requests, latency, failures, and backend node states, and do so in a format compatible with monitoring systems such as Prometheus or StatsD. The frequency of data updates, the delay between an event and its appearance in metrics, and the possibility of detail by domain or IP address are evaluated. If telemetry is incomplete or arrives with delays of tens of seconds, such a system cannot participate in autoscaling mechanisms or SLA modeling.

6. Resilience to Dynamics and Failures

In addition to individual capabilities, it is necessary to test the load balancer's resilience during long-term operation under conditions of continuous change. This is the criterion most similar to real-world conditions. It includes long-duration testing, during which certificate changes, blacklist updates, backend additions, and removals occur simultaneously, while stable load is applied at the input. If such a test reveals memory leaks, latency growth, thread blocking, API hangs, or incorrect connection termination, the load balancer cannot be considered resilient.

Unlike the problem statement, which identified the challenges, this section defines how these challenges should be translated into measurable parameters. These criteria form the basis for the next stage is the development of a comparative analysis methodology in which they will be tested on real software solutions.

Methodology of Comparative Analysis

The previously formulated criteria for selecting a load balancer require an evaluation method that takes into account not the static state of the system, but its behavior over time, at moments of configuration change, under accumulated load, and during the simultaneous operation of TLS, filtering, and scaling mechanisms. Therefore, traditional approaches limited to measuring the maximum number of requests per second or median latency in an unchanging configuration are insufficient. To make the evaluation relevant to the real operation of cluster web hosting, it is necessary to reproduce the very nature of this environment, that is continuous change.

The Principle of a Dynamic Benchmark

The proposed methodology is based on the concept of a dynamic benchmark. Unlike classical load tests, here the system is tested not in a single fixed state but during its change. Load is applied continuously, while the configuration of the load balancer and infrastructure gradually becomes more complex: the number of certificates increases, some of them are updated, new backend servers are added, old ones are removed, and protection mechanisms update and apply a new blacklist. The goal is not simply to record performance peaks but to trace how the load balancer reacts to changes, whether it maintains connections, whether latency grows, and whether SLA violations occur.

Basic Structure of the Experimental Environment

The methodology involves the use of three components:

1. a load generator,
2. the tested load balancer,
3. a group of backend servers that respond as quickly as possible (HTTP 200 OK).

Such a scheme creates conditions where the only source of degradation is the load balancer itself, not slow applications or databases. This makes it possible to isolate the studied effect.

Immediately before the start of the experiment, the system is placed in a minimal configuration state: the load balancer serves a small number of domains, the IP table is empty, and only a few TLS certificates are loaded. Then continuous load

generation begins, and the system sequentially passes through several stages.

Sequential Complexity Increase

The first stage is characterized by an increase in the number of domains and certificates. The configuration expands from hundreds to thousands of records. At this stage, not absolute RPS values but delays, CPU spikes, and transitions between states are recorded. The second stage includes certificate updates during load: some of them are replaced or deleted. Measurements are taken to determine whether short service interruptions occur, handshake errors appear, or temporary delays arise.

The third stage is related to asynchronous logic: a system is activated that sends copies of requests to a queue, where they are analyzed and converted into new blacklist entries. It is important to measure how quickly the load balancer accepts these changes and how this affects the latency of the main traffic. At the fourth stage, the cluster topology changes: new backend servers appear, and some of them are later removed. The ability of the load balancer to reassign requests without breaking connections is evaluated.

What Is Measured

For each stage, metrics corresponding to the selection criteria are recorded:

- response latency (p50, p95, p99) before, during, and after configuration changes;
- number of failed connections and TLS errors;
- reaction time of the load balancer to an external configuration change command;
- increase in CPU and memory consumption under growing TLS load;
- telemetry behavior: delay in data appearance, continuity of metric output;
- emergence of cumulative effects: memory leaks, latency increase after a series of changes.

At the analysis level, these data are converted into integral indicators. For example, resilience under dynamic conditions can be expressed through the ratio of performance in static and changing states. Other parameters, reaction time to changes or degradation coefficient, are evaluated separately for each solution.

The Principle of Reproducibility

The methodology must be reproducible. This means that all configuration changes are performed programmatically: through REST API, scripts, or xDS/Runtime interfaces. Test scenarios must be documented in such a way that another researcher can obtain the same results on a different infrastructure. This is especially important, since the goal of the methodology is not to compare specific software versions but to establish a way of their objective evaluation.

Unlike traditional static tests, the described approach views the load balancer as a system in motion: it changes, adapts, and must continue to operate. The ability to maintain manageability, predictability, and performance under dynamic conditions become the key criterion that distinguishes an engineering solution from an architecturally mature component of a cloud cluster.

Conclusion

In modern cloud-cluster web hosting, the load balancer has long ceased to be an auxiliary network component. Its role has shifted toward a managed architectural element that determines SLA compliance, resilience to configuration dynamics, and the system’s ability to adapt without downtime. However, existing evaluation methodologies for load balancers remain focused on static conditions and measure only throughput and latency, without reflecting the nature of real operational scenarios. As a result, solutions that show high performance in laboratory tests may prove unsuitable under conditions of massive TLS, frequent configuration changes, or integration with asynchronous traffic filtering systems.

This work proposes a methodological approach that allows evaluating load balancers in dynamics, not only by the speed of request processing but also by how stably and predictably they react to changing architectural conditions. For this purpose, the logic of selecting criteria was redefined: each criterion is connected not

with the internal functionality of the product, but with a specific operational challenge that arises in a cloud-cluster environment. Massive rotation of TLS certificates, hot configuration changes, in-memory blacklist updates, the appearance or disappearance of backend servers, and the continuity of telemetry, all these processes are considered not as exceptions but as the normal state of the system.

The scientific novelty of this work lies in shifting the focus from evaluating peak performance to evaluating the system's behavior at the moments of change. The concept of a "dynamic benchmark" is proposed a test scenario in which the load on the balancer is combined with configuration evolution, allowing the identification not only of maximum RPS or latency values but also of the solution's ability to maintain SLA under continuous environmental transformation. This perspective unites previously separate domains: network engineering practice, fault-tolerant system architecture, and asynchronous request-processing methods, including those described in the patented model [4].

The practical value of the study lies in forming a reproducible methodology that can be applied when selecting a load balancer during migration from classical shared hosting to a cloud-cluster architecture. The provider receives a tool that makes it possible not only to compare existing solutions (HAProxy, NGINX, Envoy, Traefik, Caddy) but also to justify the development of a proprietary software load balancer if existing options do not provide the required controllability or reaction speed to changes.

Further development of this work includes the implementation of a fully functional test environment, experimental validation of the proposed criteria, and the construction of a quantitative comparison table for the five load balancers. Another direction will be the creation and testing of a prototype load balancer written in Rust, with built-in support for multithreading, safe real-time configuration modification, and asynchronous traffic analysis. This will make it possible to finally verify the hypothesis that performance and intelligent manageability can be combined in a single solution without compromises in reliability or security.

References

1. Chizhov A., Fesenko A. Web hosting companies' client solutions: A study of a strategic standpoint // Corporate & Business Strategy Review. – 2025. – Vol. 6, No. 1. – P. 421-429. – DOI: 10.22495/cbsrv6i1siart18.

2. HAProxy Runtime API Reference [Electronic resource]. – Access mode: https://www.haproxy.com/documentation/haproxy-runtime-api/reference/ (accessed 04.11.2025).

3. Envoy Proxy. Secret discovery service (SDS) [Electronic resource]. – Access mode: https://www.envoyproxy.io/docs/envoy/latest/configuration/security/secret (accessed 04.11.2025).

4. Dudnik, A., Fesenko, A., & Chyzov, O. (2025). Method of web application protection based on asynchronous request analysis: Utility Model Patent of Ukraine No. UA 160974 U. Registered 22 October 2025. Kyiv: Ukrainian National Office of Intellectual Property and Innovation.

5. Chizhov A., Fesenko A., Ziuziun V., Basshykyzy D. "Cloud Shared Hosting DDoS Resistance and Potential Ways of Protection" // CEUR Workshop Proceedings "Cyber Hygiene & Conflict Management in Global Information Networks 2024". – ISSN 1613-0073. – Vol. 3925. – P. 13–23. – Access mode: https://ceur-ws.org/Vol-3925/ (accessed 04.11.2025).

6. Pereira D. S., Bezerra L. F. V., Nunes J. S., Barroca Filho I. M., Lopes F. A. S. Performance Efficiency Evaluation based on ISO/IEC 25010:2011 applied to a Case Study on Load Balance and Resilient // Workshop de Testes e Tolerância a Falhas (WTF) 2023. – DOI: 10.5753/wtf.2023.787.

7. Johansson A. HTTP Load Balancing Performance Evaluation of

HAProxy, NGINX, Traefik and Envoy with the Round-Robin Algorithm : Bachelor Degree Project in Science with a major in Informatics, G2E, 30 ECTS / A. Johansson ; supervisor J. Zaxmy, examiner T. Fischer ; University of Skövde. – Skövde, Sweden, 2022. – 54 p.

8. nginx documentation – njs. [Electronic resource]. – Access mode: https://nginx.org/en/docs/njs/ (accessed 04.11.2025).

9. Traefik Labs. ACME: certificate resolvers for TLS / Traefik Proxy [Electronic resource]. – Access mode: https://doc.traefik.io/traefik/reference/install-configuration/tls/certificate-resolvers/acme/ (accessed 04.11.2025).

10. Caddy – Automatic HTTPS [Electronic resource]. – Access mode: https://caddyserver.com/docs/automatic-https (accessed 04.11.2025).

11. Envoy Proxy. Wasm [Electronic resource]. – Access mode: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/advanced/wasm (accessed 04.11.2025).

12. Chovanec M., Hasin M., Havrilla M., Chovancová E. Detection of HTTP DDoS Attacks Using NFStream and TensorFlow // Applied Sciences. – 2023. – Vol. 13, No. 11. – Article 6671. – DOI: 10.3390/app13116671.

13. Najafimehr M., Zarifzadeh S., Mostafavi S. DDoS attacks and machine-learning-based detection methods: A survey and taxonomy // Engineering Reports. – 2023. – Vol. 5. – Article e12697. – DOI: 10.1002/eng2.12697.

Alexander Chizhov, Andriy Fesenko.

INTELLIGENT LOAD BALANCING MANAGEMENT IN CLOUD WEB HOSTING: EVALUATION CRITERIA AND METHODOLOGY

This paper presents a methodology for selecting a software load balancer designed for cloud-cluster web hosting environments that use a large number of Transport Layer Security (TLS) certificates, perform their dynamic rotation, and integrate with an asynchronous network traffic analysis system. It is shown that traditional methods of evaluating load balancers, based on static testing of throughput and response time, do not reflect real operating conditions, where the system configuration constantly changes, block lists are updated, the composition of servers varies, and compliance with the Service Level Agreement (SLA) is required. Under such conditions, the load balancer should be considered an active control element capable of handling tens of thousands of certificates, applying configuration changes without process restarts, updating internal in-memory data structures, and providing reliable telemetry for automatic scaling systems. A new evaluation approach is proposed, based on the concept of a "dynamic benchmark," which combines load testing with gradual configuration complexity: increasing the number of domains, updating certificates, applying new block list entries, and changing cluster topology. Key evaluation criteria are defined, including scalability, resilience under dynamic changes, flexibility for implementing custom logic, manageability, and completeness of observability. The developed methodology eliminates the existing gap between academic research and the practical operation of modern hosting platforms and creates a foundation for further experimental validation and the development of a next-generation load balancer prototype focused on reliability, adaptability, and intelligent load management.

***Keywords**: cloud web hosting, load balancing, dynamic TLS management, SLA-aware scaling, dynamic benchmark.*

Олександр Чижов, Андрій Фесенко

ІНТЕЛЕКТУАЛЬНЕ УПРАВЛІННЯ БАЛАНСУВАННЯМ НАВАНТАЖЕННЯ У ХМАРНОМУ ВЕБ-ХОСТИНГУ: КРИТЕРІЇ ОЦІНЮВАННЯ ТА МЕТОДОЛОГІЯ

У статті представлено методологію вибору програмного балансувальника навантаження, призначеного для середовищ хмарно-кластерного веб-хостингу, які використовують велику кількість сертифікатів Transport Layer Security (TLS), виконують їх динамічну ротацію та інтегруються з асинхронною системою аналізу мережевого трафіку. Показано, що традиційні методи оцінювання балансувальників навантаження, засновані на статичному тестуванні пропускної здатності та часу відгуку, не відображають реальних умов експлуатації, де конфігурація системи постійно змінюється, оновлюються списки блокування, змінюється склад серверів і вимагається дотримання угоди про рівень обслуговування (SLA). За таких умов балансувальник навантаження слід розглядати як активний елемент управління, здатний обробляти десятки тисяч сертифікатів, застосовувати зміни конфігурації без перезапуску процесів, оновлювати внутрішні структури даних у пам'яті та надавати надійну телеметрію для систем автоматичного масштабування. Запропоновано новий підхід до оцінювання, заснований на концепції «динамічного бенчмарку», який поєднує навантажувальне тестування з поступовим ускладненням конфігурації: збільшенням кількості доменів, оновленням сертифікатів, застосуванням нових записів у списках блокування та зміною топології кластера. Визначено ключові критерії оцінювання, зокрема масштабованість, стійкість до динамічних змін, гнучкість реалізації власної логіки, керованість та повноту спостережуваності. Розроблена методологія усуває існуючий розрив між академічними дослідженнями та практичною експлуатацією сучасних хостингових платформ і створює основу для подальшої експериментальної перевірки та розробки прототипу балансувальника навантаження наступного покоління, орієнтованого на надійність, адаптивність та інтелектуальне управління навантаженням.

***Ключові слова**: хмарний веб-хостинг, балансування навантаження, динамічне управління TLS, масштабування з урахуванням SLA, динамічний бенчмарк.*