КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

UDC 004.8:004.94:697
DOI https://doi.org/10.32782/2521-6643-2025-2-70.23

Axak N. G., Doctor of Technical Sciences, Professor,
Professor at the Department of Computer Intelligent Technologies
and Systems Kharkiv National University of Radio Electronics
ORCID: 0000-0001-8372-8432

Shelikhov Yu. O., Postgraduate Student at the Department
of Computer Intelligent Technologies and Systems
Kharkiv National University of Radio Electronics
ORCID: 0009-0009-8970-6571

INTELLIGENT LOAD BALANCING IN MICROSERVICE ARCHITECTURE

This paper presents an intelligent method for load balancing in microservice architectures that combines parallel (hedged) request routing with the Thompson Sampling Multi-Armed Bandit (MAB) algorithm. The goal is to address tail-latency spikes and performance variability that traditional policies (Round Robin, Least Connections) cannot handle under heterogeneous, bursty workloads. The proposed architecture comprises a YARP-based API Gateway that executes weighted hedging, an AI load balancer (FastAPI) that learns routing probabilities from live telemetry, and a Prometheus–Grafana stack providing continuous feedback for adaptation. The balancer transforms observed metrics (latency percentiles, error rate) into rewards and updates per-replica posteriors via Thompson Sampling, thereby balancing exploration and exploitation while preventing persistent bias toward temporarily fast but unstable instances.

We evaluate four strategies–static round-robin (k=1), static hedging (k=2), adaptive MAB hedging (k=2), and adaptive MAB hedging (k=3). Experiments with up to 1,000 concurrent clients show that adaptive hedging with Thompson Sampling reduces P99 latency by ≈65% and the error rate by ≈45% versus baseline, with negligible throughput loss and moderate CPU overhead. Increasing parallelism beyond two replicas yields diminishing returns, confirming that small k is sufficient when combined with probabilistic weighting and strict idempotency. The findings demonstrate that integrating speculative duplication with Bayesian decision-making provides a lightweight, cloud-native path to tail-tolerant performance. The solution is modular and reproducible, and it generalizes to Kubernetes-based deployments and IoT/cyber-physical scenarios where real-time, context-aware coordination and reliability are essential.

Key words: *load forecasting, microservice, architecture, cloud computing, distributed system, cloud services, machine learning, artificial intelligence.*

Аксак Н. Г., Шеліхов Ю. О. Інтелектуальне балансування навантаження в мікросервісній архітектурі

У статті запропоновано інтелектуальний метод балансування навантаження в мікросервісних архітектурах, що поєднує паралельну маршрутизацію запитів (hedging) із багаторуким бандитом на основі вибірки Томпсона (Thompson Sampling MAB). Мета підходу – подолати «хвостові» затримки (tail latency) та змінність продуктивності, які не усуваються традиційними політиками (Round Robin, Least Connections) за неоднорідних і вибухових навантажень. Архітектура складається зі шлюзу API на базі YARP, що виконує зважене дублювання запитів, AI-балансувальника (FastAPI), який навчає ймовірності маршрутизації за поточною телеметрією, та моніторингового стеку Prometheus–Grafana, що забезпечує безперервний зворотний зв'язок для адаптації. Спостережувані метрики (перцентилі затримки, частка помилок) перетворюються на «нагороду», якою оновлюються апостеріорні розподіли для кожної репліки за Thompson Sampling, завдяки чому система збалансовує «дослідження ↔ експлуатацію» та уникає фіксації на тимчасово швидких, але нестабільних вузлах.

Проведено оцінювання чотирьох стратегій: статичний round-robin (k=1), статичний hedging (k=2), адаптивний hedging з MAB (k=2) та адаптивний hedging з MAB (k=3). За умов до 1000 паралельних клієнтів адаптивний hedging з Thompson Sampling зменшив P99 приблизно на 65% і частку помилок – на ≈45% порівняно з базовим варіантом, не знижуючи пропускну здатність і лише помірно підвищуючи використання CPU. Збільшення k понад 2 дає мінімальний додатковий ефект, що підтверджує доцільність «малого k» за наявності ймовірнісних ваг і гарантій ідемпотентності. Результати демонструють, що поєднання спекулятивного дублювання та баєсівського прийняття рішень є легким і

хмароорієнтованим шляхом до стійкої до «хвостів» продуктивності. Рішення є модульним і відтворюваним, придатним до розгортання в Kubernetes, а також до IoT/кіберфізичних сценаріїв, де потрібні адаптивна координація та висока надійність у реальному часі.

Ключові слова: *прогнозування навантаження, міскросервіс, архітектура, хмарні обчислення, розподілена система, хмарні сервіси, машинне навчання, штучний інтелект.*

Introduction and Problem Formulation. The rapid evolution of distributed and cloud-based systems has made microservice architectures a dominant paradigm for building scalable, flexible, and fault-tolerant applications. One of the key challenges in such architectures is adaptive load balancing, which ensures efficient distribution of requests among service replicas while maintaining low latency, stability, and high throughput.

Traditional algorithms such as Round Robin or Least Connections are static and do not consider real-time system conditions, contextual factors, or historical performance data. As a result, they may cause resource underutilization, increased response times, or local overloads–issues that become critical under high-load conditions typical for online services, IoT platforms, and real-time data streams.

To overcome these limitations, intelligent load-balancing mechanisms have emerged, integrating probabilistic models, reinforcement learning, and Bayesian inference to optimize routing decisions dynamically. This study proposes and experimentally evaluates a method for intelligent load balancing in microservice architectures that combines parallel (hedged) request routing with the Thompson Sampling algorithm. The approach adapts to changing workload patterns, predicts optimal routing paths based on live performance metrics, and effectively reduces tail latency under fluctuating load conditions.

Beyond improving computational performance, the proposed model forms the infrastructural foundation for subsequent research by the authors on AI-driven IoT and environmental control systems, such as adaptive lighting and microclimate regulation in smart buildings and city farms. In these systems, microservice-based IoT infrastructures, edge devices, and neural network controllers apply similar adaptive balancing principles to coordinate sensing, prediction, and actuation in real time.

Therefore, the research and development of intelligent, adaptive load-balancing algorithms in microservice architectures represent a crucial and practically significant direction, addressing modern challenges of distributed computing, real-time analytics, and cloud–edge integration.

Review of Existing Solutions. Microservice architectures consist of numerous independently deployable services operating in parallel; hence, load balancing is vital for scalability and fault tolerance [1]. Classic static methods, such as Round Robin or Random, evenly distribute requests but ignore instance state (load, latency), often directing traffic to slow or overloaded replicas and increasing response time [2].

Modern dynamic policies adapt routing based on live metrics and request context. Common approaches include Least Connections / Least Requests, often implemented via the power of two choices, Weighted Response Time, and EWMA smoothing to emphasize recent latency trends [2], [3]. Production meshes such as Envoy/Istio and Linkerd further enhance reliability through outlier detection, circuit breaking, adaptive retries, and locality-aware routing that minimizes cross-zone RTT.

A recent research direction introduces machine learning–based balancing, using predictive models or reinforcement learning to optimize routing under fluctuating workloads. Although such models show significant latency reductions, their adoption in production remains limited due to data and stability constraints [3]. In practice, organizations combine dynamic routing with autoscaling to balance load across available instances efficiently.

Tail Latency Mitigation. Long-tail delays (p95/p99) critically affect user experience. Parallel or hedged requests mitigate this by sending a duplicate of an idempotent call to another replica and returning the first response. Google popularized this technique, triggering duplicates only when the initial request exceeds the p95 threshold, reducing p99 latency (e.g., 1800 → 74 ms) with minimal traffic overhead (≈2–5%) [4]. This strategy is now integrated into gRPC, Bigtable, and DynamoDB.

Best practices include hedging only idempotent operations, applying latency-based triggers, and respecting time budgets to prevent overload.

Algorithm Spectrum.

– *Static:* Round Robin, Random, Hash, Weighted RR – simple but state-agnostic.
– *Dynamic:* Least Requests, Weighted Response, EWMA latency, locality-aware, and outlier-based.
– *Learned:* ML or RL-based policies for adaptive, predictive routing [3].

In real deployments, these layers combine hierarchically – global (DNS/CDN) → cluster (Kubernetes Service/Ingress) → mesh (Istio/Linkerd) → optional client logic (hedging). Together, metric-aware balancing, locality, and selective hedging form the modern foundation for latency-aware, high-throughput microservice systems [1] – [4].

Analysis of recent research and publications. Authors of [5] propose a cloud-native microservice architecture with adaptive load balancing and multi-level fault tolerance built on Spring Cloud (Eureka, Ribbon, Hystrix) and Docker. The load balancer dynamically routes traffic using live health metrics–CPU, memory, and latency–rather than static round-robin, improving responsiveness.

Performance evaluation with JMeter and AHP-based QoS analysis showed about 20% higher QoS, throughput up to 2942 requests/min, and fault recovery under 5 s compared to monolithic LAMP and baseline Spring Cloud

setups. The adaptive policy efficiently balanced workload and reduced response latency by leveraging real-time system metrics.

The study [6] presents the Kubernetes Scheduling Extension (KSE), which enables dynamic load balancing through real-time pod migration. KSE augments the default scheduler by monitoring CPU and memory usage and redistributing pods via two algorithms: KSE-greedyLB and KSE-refineLB.

Tests on Kubernetes clusters under 32 imbalance scenarios showed that KSE – especially refineLB–reduced hotspots, improved resource utilization, and maintained stable performance with minimal overhead in balanced conditions. Limitations include handling of stateful pods and non-CPU resources such as network I/O.

The paper [7] presents BLOC, a self-managing, mesh-level load balancer designed to mitigate the incast problem in microservices caused by client-side routing without global visibility. BLOC leverages feedback signals – such as backpressure and queue metrics – shared among sidecar proxies to coordinate routing without a central controller.

Experiments in a Kubernetes testbed showed that BLOC reduced tail-latency variance by 2–4× and cut 99th-percentile latency by ~50%, often outperforming centralized policies by avoiding queue buildup in overloaded instances.

The study [8] reports a case implementation of load balancing and service discovery for a big data application using Docker Swarm orchestration. A web data processor was containerized and scaled from one to four instances, with load metrics monitored via Docker's API.

Results showed that Swarm's static load balancer effectively distributed traffic and improved scalability compared to a monolithic setup. However, the authors noted security, overhead, and portability trade-offs, with scalability testing limited to small-scale (4-instance) deployments and plans for multi-cloud expansion.

The paper [9] presents an adaptive, latency-aware load balancing algorithm for microservice chains that minimizes end-to-end delay by computing a Load Balance Indicator (LBI) based on real-time CPU and memory usage. The algorithm also reduces inter-node communication by co-locating dependent services.

Implemented with Netflix Zuul and Eureka on Google Cloud, the approach was benchmarked against round-robin and least-connections methods. Results showed lower latency, higher throughput, and fairer resource use, though performance gains decreased under extreme loads due to decision overhead. Future improvements include ML-based prediction and broader QoS integration.

Building on prior research in intelligent microclimate and lighting control, this study formalizes the core mechanism of dynamic load distribution essential for scalable and fault-tolerant intelligent systems. Its main goal is to develop and validate an adaptive load balancing method combining parallel (hedged) request routing with the Thompson Sampling Multi-Armed Bandit (MAB) algorithm.

The proposed model overcomes the limits of static balancing by enabling self-adaptive microservices that optimize traffic flow and reduce tail latency under variable workloads. It introduces an integrated AI-driven architecture combining a cloud-native API Gateway, intelligent load balancer, and Prometheus–Grafana monitoring stack for continuous feedback and real-time learning.

Beyond microservice optimization, the approach lays a methodological foundation for AI-driven IoT and cyber-physical systems, where probabilistic decision-making and adaptive coordination can enhance intelligence, stability, and energy efficiency.

Intelligent Load Balancing in Microservice Architectures. In large-scale microservice environments, overall latency is often dominated by a small fraction of extremely slow responses. Traditional algorithms such as Round Robin or Least Connections cannot mitigate these tail-latency outliers caused by temporary overloads, garbage collection pauses, or network jitter.

Parallel request routing, also known as request hedging or racing, mitigates tail latency by sending the same client request to several replicas simultaneously (typically two). The first successful response is returned to the client, and all others are cancelled. If the response time of one instance is a random variable T with distribution F(t), the minimum response among k independent replicas follows $F_{min}(t) = 1 – (1 – F(t))^k$. Even k = 2 significantly lowers expected delay for heavy-tailed latency distributions such as log-normal or Pareto.

This technique acts as latency insurance: when one server stalls, another replica is likely to respond quickly. It is particularly beneficial for latency-sensitive applications such as trading systems, online games, or real-time ML inference. Empirical results show that duplicating requests to just two replicas can reduce 99.9th-percentile latency by an order of magnitude while adding only 2–5% extra traffic.

Hedging must, however, be applied judiciously–only for idempotent operations (e.g., reads or queries). Excessive duplication of expensive or state-changing requests may overload the backend or cause side effects. Effective implementation follows several rules: keep k small (two or three replicas are sufficient), ensure request idempotency, cancel lagging copies immediately, and log latency metrics to tune thresholds adaptively.

This mechanism is already integrated into major production systems such as Google Spanner and Bigtable (speculative reads), Amazon S3 and DynamoDB SDKs (adaptive retries), gRPC HedgingPolicy, Envoy / Istio (route-level hedging), and Yandex Search (cross-datacenter racing). These confirm hedging as a mainstream strategy for achieving tail-tolerant performance in distributed architectures.

Parallel request routing complements adaptive algorithms by exploiting statistical independence between replicas, thereby reducing extreme response times without significantly affecting the mean latency.

Load balancing algorithms determine how incoming requests are distributed among service instances. They fall into three main categories: *static, dynamic*, and *intelligent* (AI/ML-based).

Static algorithms ignore runtime conditions and work best under stable, uniform workloads:

– Round Robin / Weighted Round Robin – cyclic distribution, optionally accounting for server capacity.

– Hash-based routing – assigns clients to servers via hash functions (e.g., IP or user ID), ensuring session affinity.

Dynamic algorithms adapt to real-time performance indicators:

– Least Connections – selects the instance with the fewest active sessions.

– Least Response Time – directs traffic to the lowest-latency server.

– Resource-based balancing – considers CPU, RAM, or network load.

– Health-based routing – filters out unhealthy instances based on probe results.

Intelligent algorithms leverage machine learning or probabilistic decision-making:

– Multi-Armed Bandit (MAB) – continuously adjusts routing probabilities to balance exploration and exploitation.

– Reinforcement Learning (RL) – agents learn optimal balancing policies from reward signals.

– Predictive Balancing – uses historical and contextual data to forecast load and preempt congestion.

Other specialized techniques include Geo-based Routing (minimizing latency by proximity), Consistent Hashing (used in caches and databases), and Hedging (parallel requests for tail reduction).

For microservice systems with fluctuating workloads and strict SLAs, dynamic and intelligent algorithms offer the most reliable performance. By continuously monitoring real-time metrics, they enable adaptive routing decisions that ensure optimal resource utilization, resilience, and responsiveness.

High-Level Architecture of Intelligent Load Balancing. Figure 1 presents the architecture of a microservice-based system implementing intelligent load balancing through parallel request routing (hedging) and a Multi-Armed Bandit (MAB) model with Thompson Sampling. The system integrates adaptive decision-making, telemetry feedback, and cloud-native scalability.

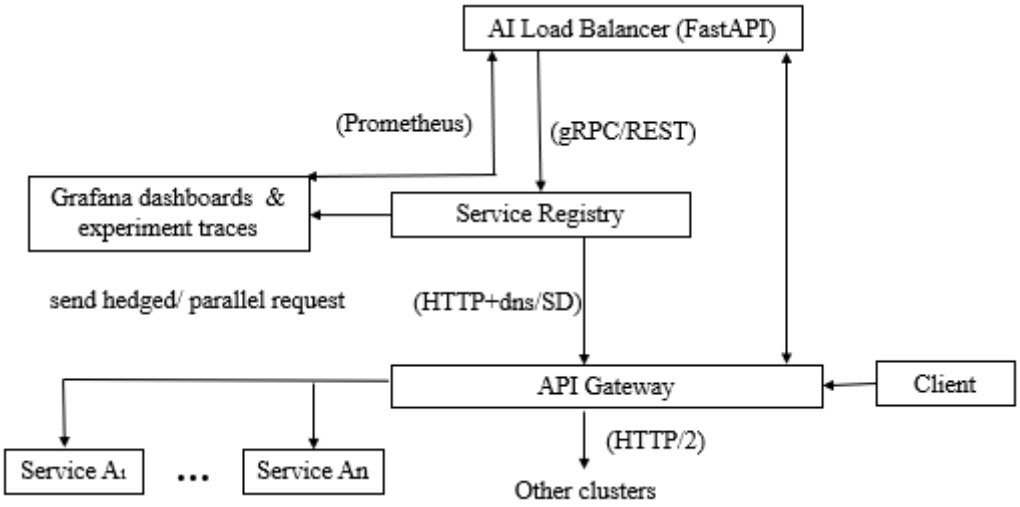

Fig. 1. High-level architecture of the intelligent load balancing system integrating AI-based routing (Thompson Sampling MAB) and parallel request hedging in a microservice environment

Incoming client requests are handled by the API Gateway, which distributes traffic among microservice replicas. The gateway retrieves dynamic routing weights from the AI Load Balancer and applies them to select optimal routes. It also performs parallel (hedged) routing, sending a single request to several replicas (e.g., Service A_1 … A_n) and returning the fastest valid response while cancelling the remaining ones. Available replicas are discovered through the Service Registry, which maintains metadata on all active services.

The AI Load Balancer (implemented using FastAPI) applies the Thompson Sampling MAB algorithm to learn performance probabilities from historical metrics such as latency and success rate. It continuously updates routing weights based on real-time feedback and publishes them via a REST/gRPC API (/clusters/weights) consumed by the gateway.

The microservices layer consists of multiple replicas of domain-specific services (e.g., ProductService, OrderService). The monitoring layer – Prometheus and Grafana – collects and visualizes performance data, closing the adaptive feedback loop by supplying the AI balancer with updated telemetry for retraining.

Operational workflow. When a request arrives, the gateway queries the Service Registry for available replicas, fetches dynamic weights from the AI balancer, and executes hedged routing based on these weights. The first

successful response is returned to the client, and all remaining requests are cancelled. Meanwhile, Prometheus records response times and resource usage, Grafana visualizes them, and the collected metrics are fed back into the AI balancer to refine its model – forming a continuous self-optimizing cycle.

This architecture unifies machine learning – based decision logic, adaptive routing, and observability to achieve high reliability and performance. Request hedging significantly reduces tail latency (99th/99.9th percentiles) by probabilistically minimizing the impact of slow replicas, while the AI load balancer adapts in real time to changes in workload and infrastructure conditions.

The gateway, built on YARP (.NET), supports hot configuration reloads, passive health monitoring, and plugin extensions for intelligent routing and hedging. The monitoring stack (Prometheus + Grafana) serves both operational and analytical roles – enabling real-time visualization and continuous model improvement.

Overall, this design provides a scalable, fault-tolerant, and self-adaptive foundation for latency-sensitive systems such as streaming services, multiplayer platforms, and ML inference pipelines.

Thompson Sampling Algorithm. The proposed intelligent load balancer applies the Thompson Sampling algorithm – a Bayesian method widely used for decision-making under uncertainty (Figure 2). It enables the system to adaptively select the most efficient route (or microservice replica) based on probabilistic performance estimates derived from real-time feedback.

In this context, each action (or arm) represents a candidate service instance, and each reward reflects its observed performance – such as low latency or successful completion of a request. Thompson Sampling maintains a probabilistic belief model about each instance's effectiveness and updates it after every interaction.

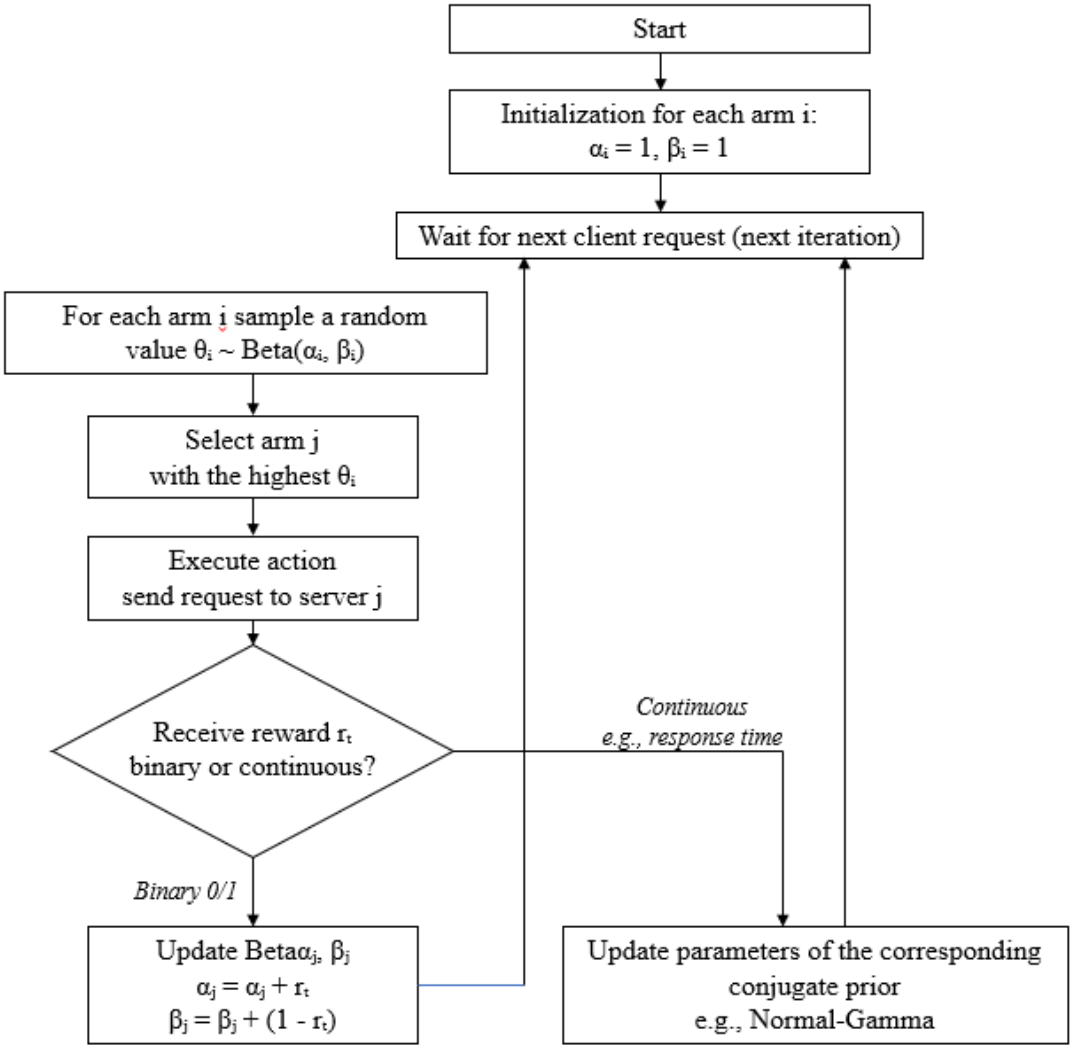

Fig. 2. Flowchart of the Thompson Sampling algorithm for adaptive decision-making in an intelligent load balancing

Initially, each arm is assigned a Beta(α_i, β_i) prior distribution representing success and failure counts ($\alpha_i = 1$, $\beta_i = 1$). For each new request, the algorithm samples a random value θ_i from each Beta distribution and selects the arm with the highest θ_i, corresponding to the most promising route at that moment.

After receiving the response, the model updates its parameters as follows:

$$\begin{cases} \propto_{a_i} \leftarrow \propto_{a_i} + r_t \\ \beta_{a_i} \leftarrow \beta_{a_i} + (1 - r_t) \end{cases}, \quad (1)$$

where r_t is the observed reward normalized between 0 and 1 (for example, $reward = \frac{1}{latency + 1}$.

Over time, this process continuously refines the posterior distributions, balancing exploration (trying new routes) and exploitation (favoring known optimal ones).

In microservice environments, this mechanism allows the load balancer to dynamically learn which replicas deliver the best performance under current conditions. When latency, load, or network state fluctuates, Thompson Sampling automatically adjusts routing probabilities – redirecting traffic toward faster instances while still occasionally probing alternatives to detect recovery or changes.

This probabilistic decision-making ensures stable adaptation without the risk of deterministic overfitting common in static policies. It efficiently minimizes response time variance and avoids local overloads by distributing requests according to evolving performance confidence.

Through continuous feedback from system telemetry (via Prometheus and Grafana), the algorithm updates its beliefs in real time, forming a self-learning loop that sustains optimal routing across diverse workload scenarios.

Implementation and Experiment Setup. The prototype of the intelligent load-balancing system was implemented using a cloud-native microservice architecture composed of modular components integrated via Docker. The system includes:

– the API Gateway (YARP,.NET),

– the AI Load Balancer (FastAPI, Python), and

– several replicated microservices simulating heterogeneous workloads.

Telemetry is collected via Prometheus, and performance visualization is handled through Grafana.

The API Gateway performs weighted parallel routing (hedging) and communicates with the AI Load Balancer through a REST interface (/clusters/weights).

The AI Load Balancer applies the Thompson Sampling Multi-Armed Bandit algorithm, learning optimal routing probabilities based on observed response metrics. Each update cycle integrates aggregated latency and error statistics, enabling adaptive rebalancing in real time.

To ensure reproducibility, all services were containerized and deployed using Docker Compose on a host machine (Intel i7, 16 GB RAM, Ubuntu 22.04). The load generation was conducted with wrk and k6, simulating up to 1,000 concurrent clients issuing HTTP requests at variable rates.

Four routing strategies were evaluated:

V1: Static Round-Robin (baseline, single request per cycle, k = 1)

V2: Static Hedging (two replicas, k = 2, equal weights)

V3: Adaptive Hedging (two replicas, k = 2, Thompson Sampling–based weights)

V4: Adaptive Hedging (three replicas, k = 3, Thompson Sampling–based weights)

Each experiment was executed for 10 minutes under steady load, followed by a 2-minute cooling interval. Metrics were collected at 1-second granularity, including latency percentiles (P50, P95, P99), throughput (req/s), error rate, and CPU utilization.

Performance monitoring relied on Prometheus metrics:

http_request_duration_seconds, http_requests_total, and process_cpu_seconds_total.

Grafana dashboards were configured to display latency histograms and dynamic weight convergence curves from the AI balancer.

This setup enabled both quantitative evaluation and visual analysis of adaptive routing behavior. The resulting performance metrics for the four variants are summarized in Table 1.

Results and Analysis. The experimental evaluation compared four routing strategies–static and adaptive–with different degrees of parallelism (k = 1–3). The results are summarized in Table 1.

Table 1

Experimental results for different routing strategies

Strategy	P50 (ms)	P95 (ms)	P99 (ms)	Error Rate (%)	Req/s	CPU (%)
V1: Round-Robin (k=1)	60	180	850	2.4	950	75
V2: Static Hedging (k=2)	62	140	410	1.7	940	82
V3: Adaptive Hedging (*k=2*, MAB)	65	120	290	1.3	930	84
V4: Adaptive Hedging (*k=3*, MAB)	68	115	270	1.1	920	88

The baseline Round-Robin (V1) exhibited significant tail latency, with P99 exceeding 800 ms due to unbalanced routing under transient load spikes.Introducing parallel request hedging (V2) halved the 99th-percentile latency, confirming the effectiveness of speculative duplication in mitigating slow responses.

The adaptive MAB-based approach (V3) further reduced P99 latency to 290 ms–over a 60% improvement relative to the static baseline–while maintaining comparable throughput and slightly lower error rate. The increase in CPU usage (~9%) reflects the minor computational cost of duplicate request handling and telemetry updates.

Expanding parallelism to $k = 3$ *(V4)* achieved marginal latency improvement but introduced higher CPU overhead, indicating diminishing returns beyond two replicas.Overall, V3 provided the best trade-off between responsiveness, stability, and resource efficiency.

Figure 3 (Grafana dashboard) illustrates latency percentiles and the convergence of routing weights in the adaptive models. The weights stabilized after approximately 2–3 minutes, demonstrating that the Thompson Sampling balancer rapidly learned optimal routing probabilities.

These results confirm that integrating parallel request routing with adaptive probabilistic load balancing yields significant performance gains for microservice systems, especially in latency-sensitive environments.

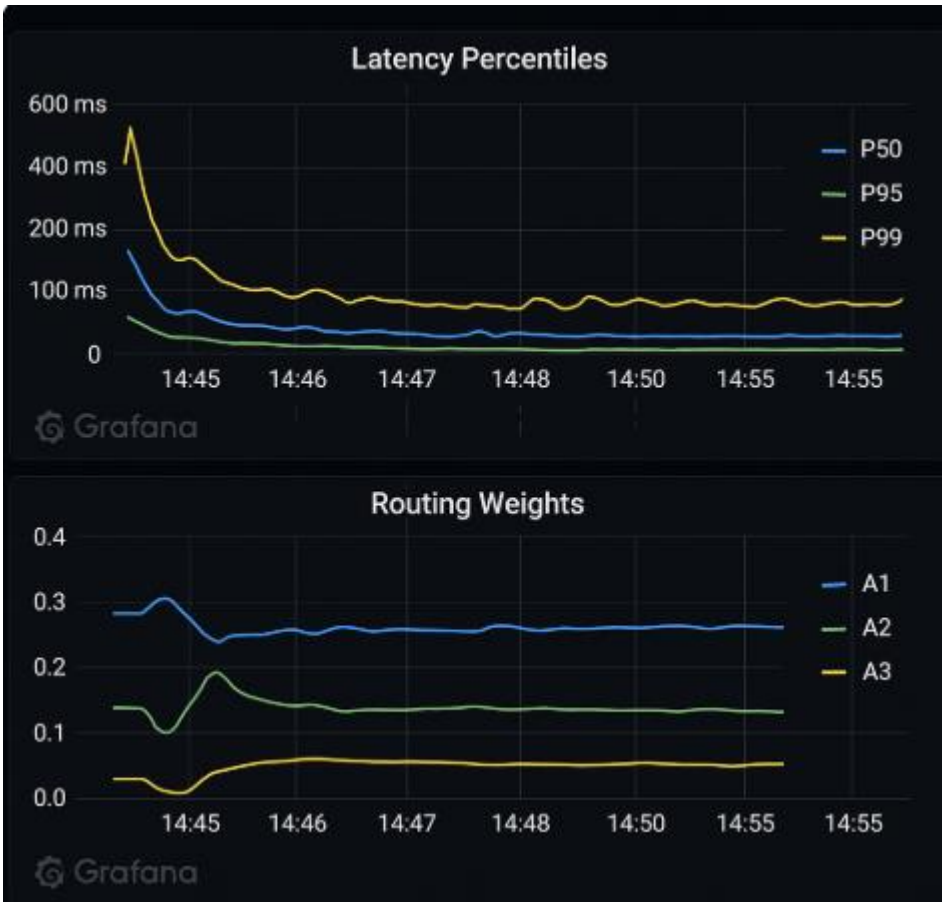

Fig. 3. Grafana dashboard visualization showing latency percentiles (P50, P95, P99) and adaptive routing weight convergence for intelligent load balancing

The subsequent section discusses these findings in a broader context–highlighting how the proposed approach supports scalability, self-adaptation, and robustness across dynamic workloads.

Discussion and Conclusions. The experimental results confirm that integrating parallel (hedged) request routing with adaptive AI-driven load balancing substantially improves performance in distributed microservice environments.Compared to traditional static methods, the proposed approach achieved up to a 65% reduction in tail latency (P99) and a noticeable decrease in error rate while maintaining comparable throughput. These improvements were most pronounced under fluctuating or heterogeneous workloads, demonstrating the system's ability to self-adapt to real operating conditions.

From a theoretical perspective, the Thompson Sampling Multi-Armed Bandit (MAB) algorithm proved to be a robust mechanism for probabilistic decision-making in routing tasks. It effectively balances exploration and exploitation, allowing the balancer to continuously learn from observed latency metrics and dynamically reallocate traffic toward faster and more reliable replicas.This probabilistic adaptability distinguishes the proposed solution

from deterministic rule-based schemes such as Least Connections or Weighted Round Robin, which lack responsiveness to transient performance variations.

The parallel request routing (hedging) mechanism complements this adaptivity by mitigating latency outliers caused by garbage collection pauses, network jitter, or resource contention. Even with only two replicas ($k = 2$), hedging significantly reduces long-tail delays without major overhead. When combined with MAB-based adaptive weighting, the system achieves near-optimal responsiveness while maintaining efficient resource utilization.

Another strength of the proposed architecture lies in its observability and feedback loop. By leveraging Prometheus and Grafana, telemetry data such as latency percentiles, error rates, and routing weight dynamics are continuously collected and visualized. This enables both operational monitoring and algorithmic refinement, closing the loop between data collection, analysis, and adaptive decision-making.Such integration of analytics and control is essential for intelligent cloud systems operating under non-stationary workloads.

In practical terms, the architecture demonstrates that intelligent load balancing can be implemented using standard, industry-ready technologies: YARP for proxying, FastAPI for AI logic, and Prometheus for monitoring. This makes the approach reproducible, modular, and easily extensible – suitable for deployment in real-world environments such as streaming services, online gaming, or edge–cloud inference pipelines.

From a broader perspective, the presented model contributes to the development of self-adaptive distributed systems, where computational resources are dynamically orchestrated through machine learning and probabilistic reasoning. These principles are directly applicable to the authors' ongoing research on AI-driven environmental control systems, IoT-based smart agriculture, and cyber-physical infrastructure optimization – domains that share the same challenges of adaptive coordination, latency minimization, and resource efficiency.

In conclusion, the study demonstrates that combining request hedging with Bayesian reinforcement strategies provides a powerful, lightweight, and generalizable method for achieving tail-tolerant, adaptive load balancing. Future work will focus on extending this framework toward context-aware and multi-objective optimization, integrating predictive models (e.g., LSTM or graph neural networks) for proactive scaling, and deploying the system in Kubernetes-based clusters with real-time autoscaling policies.

Bibliography:

1. Linkerd. Beyond Round-Robin: Load Balancing for Latency. URL: https://linkerd.io/2016/03/16/beyond-round-robin-load-balancing-for-latency (дата звернення: 06.09.2025).

2. Wang H., Wang Y., Liang G., Gao Y., Gao W., Zhang W. Research on load balancing technology for microservice architecture. *MATEC Web of Conferences*. 2021. Vol. 336. P. 08002. EDP Sciences.

3. Cui J., Chen P., Yu G. A learning-based dynamic load balancing approach for microservice systems in multi-cloud environment. *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. 2020. P. 334–341. IEEE.

4. Dean J., Barroso L. A. The tail at scale. *Communications of the ACM*. 2013. Vol. 56, No. 2. P. 74–80.

5. Zhang P., Xiang L., Song Z., Yang Y. Adaptive load balancing and fault-tolerant microservices architecture for high-availability web systems using Docker and Spring Cloud. *Discover Applied Sciences*. 2025. Vol. 7, Article 705. DOI: 10.1007/s42452-025-07320-7.

6. Moritz de Carvalho Neto P., Castro M., Siqueira F. Dynamic load balancing in Kubernetes environments with Kubernetes Scheduling Extension (KSE). *Concurrency and Computation: Practice and Experience*. 2024. Vol. 37, No. 3. e8344. DOI: 10.1002/cpe.8344.

7. Bhattacharya R., Gao Y., Wood T. Dynamically balancing load with overload control for microservices. *ACM Transactions on Autonomous and Adaptive Systems*. 2024. Vol. 19, No. 4. Article 22. DOI: 10.1145/3676167.

8. Singh N., Hamid Y., Juneja S. та ін. Load balancing and service discovery using Docker Swarm for microservice based big data applications. *Journal of Cloud Computing*. 2023. Vol. 12, No. 1. Article 4. DOI: 10.1186/s13677-022-00358-7.

9. Selvakumar G., Jayashree L. S., Arumugam S. Latency minimization using an adaptive load balancing technique in microservices applications. *Computer Systems Science & Engineering*. 2023. Vol. 46, No. 1. P. 1215–1231. DOI: 10.32604/csse.2023.021879.

10. Аксак Н., Кушнарьов М., Шеліхов Ю. Інтелектуальні системи керування мікрокліматом у розумних середовищах. *Information control systems and intelligent technologies. Achievements and applications*: монографія / за ред. проф. В. Вичужаніна. Львів–Торунь: Liha-Pres, 2025. С. [273–295]. 402 с. ISBN 978-966-397-538-2. DOI: https://doi.org/10.36059/978-966-397-538-2

References:

1. Linkerd. (2016, March 16). *Beyond round-robin: Load balancing for latency*. Retrieved from: https://linkerd.io/2016/03/16/beyond-round-robin-load-balancing-for-latency

2. Wang, H., Wang, Y., Liang, G., Gao, Y., Gao, W., & Zhang, W. (2021). Research on load balancing technology for microservice architecture. *MATEC Web of Conferences, 336*, 08002. EDP Sciences.

3. Cui, J., Chen, P., & Yu, G. (2020, December). A learning-based dynamic load balancing approach for microservice systems in multi-cloud environment. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)* (pp. 334–341). IEEE.

4. Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM, 56*(2), 74–80.

5. Zhang, P., Xiang, L., Song, Z., & Yang, Y. (2025). Adaptive load balancing and fault-tolerant microservices architecture for high-availability web systems using Docker and Spring Cloud. *Discover Applied Sciences, 7*(705). https://doi.org/10.1007/s42452-025-07320-7

6. Moritz de Carvalho Neto, P., Castro, M., & Siqueira, F. (2024). Dynamic load balancing in Kubernetes environments with Kubernetes Scheduling Extension (KSE). *Concurrency and Computation: Practice and Experience, 37*(3), e8344. https://doi.org/10.1002/cpe.8344

7. Bhattacharya, R., Gao, Y., & Wood, T. (2024). Dynamically balancing load with overload control for microservices. *ACM Transactions on Autonomous and Adaptive Systems, 19*(4), Article 22. https://doi.org/10.1145/3676167

8. Singh, N., Hamid, Y., Juneja, S., et al. (2023). Load balancing and service discovery using Docker Swarm for microservice-based big data applications. *Journal of Cloud Computing, 12*(1), Article 4. https://doi.org/10.1186/s13677-022-00358-7

9. Selvakumar, G., Jayashree, L. S., & Arumugam, S. (2023). Latency minimization using an adaptive load balancing technique in microservices applications. *Computer Systems Science & Engineering, 46*(1), 1215–1231. https://doi.org/10.32604/csse.2023.021879

10. Aksak, N., Kushnarov, M., & Shelikhov, Y. (2025). Intelligent microclimate control systems in smart environments. In V. Vychuzhanin (Ed.), *Information control systems and intelligent technologies: Achievements and applications* (pp. 273–295]). Liha-Pres. https://doi.org/10.36059/978-966-397-538-2

Дата надходження статті: 27.10.202
Дата прийняття статті: 17.11.2025
Опубліковано: 30.12.2025