www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 14 Issue 06 June 2025, Page No. 27340-27345
ISSN: 2319-7242 DOI: 10.18535/ijecs/v14i06.5157

Integration of machine learning into high-load web application architectures

Serhii Savchenko

Senior Full Stack Developer, IT Development (Web Dev)
New York, NY, USA

Abstract.
The article analyzes the feasibility of implementing machine learning in high-load web application architectures. An architectural technique for integrating machine learning (ML) modules into microservice high-load Kubernetes-based web applications is considered. The basis is the "Sidecar + event-driven" pattern, which allows each service to be supplemented with load forecasting on LSTM models and anomaly detection through auto-encoders. The ML Predictor, Decision Manager, and Actuator components are embedded as separate containers that communicate via the Kafka event bus. The methodological basis of the work, which made it possible to broadly consider the features of the machine learning implementation process in high-load web application architectures, was based on the results of other studies. The information presented in the article is of considerable interest to architects of distributed systems and DevOps engineers specializing in building fault-tolerant, scalable high-traffic web platforms that need to integrate machine learning models without performance degradation and while maintaining SLA requirements. In addition, the materials of the article will be useful to researchers in the field of MLOps and data engineers involved in optimizing data pipelines and ensuring low latency of inference under extreme loads, as well as graduate students developing methods for adaptive resource balancing in heterogeneous computing environments.

Keywords: highly loaded web applications; auto-scaling; machine learning; load forecasting; anomaly detection; Kubernetes; Sidecar; event-driven.

1. Introduction

Modern web applications frequently experience abrupt surge loads—driven by flash sales, promotional events or large-scale online gatherings—that demand architectures capable of maintaining high availability and rapid response times under all conditions. Conventional scaling mechanisms—static horizontal or vertical autoscaling rules—often fail to react swiftly enough to instantaneous traffic spikes, resulting in degraded service quality and potential SLA breaches. Machine learning has demonstrated efficacy in time-series traffic forecasting, anomaly detection and real-time resource management, making the integration of ML modules into high-throughput web architectures for precise, adaptive scaling a pressing area of investigation [1, 2].

The objective of this paper is to analyze the distinctive challenges associated with embedding machine-learning capabilities into architectures of high-load web applications.

The scientific novelty lies in the comprehensive methodological development and systematization of a “Sidecar + event-driven” pattern for deploying LSTM-based load-forecasting and autoencoder-based anomaly-detection ML modules as sidecar services within Kubernetes-microservice, high-throughput web architectures—thereby enabling adaptive autoscaling and SLA compliance without altering core business logic.

The central hypothesis posits that incorporating ML-driven forecasting and anomaly-detection as sidecar services will reduce reaction times to traffic peaks and decrease downtime compared to standard horizontal autoscaling approaches.

This study's methodological foundation—a broad review of prior research—provides the basis for examining the integration of machine learning into high-load web-application architectures.

2. Literature Review

Mihai D. et al. [1] propose an "integrated high-throughput services" architecture for remote-learning platforms, in which ML models trained on historical user-session data and infrastructure metrics predict traffic peaks. This enables proactive scaling of application and database containers, minimizing latency when thousands of students connect simultaneously. Rabiman R., Nurtanto M., and Kholifah N. [4] describe an LMS solution for large-scale educational resources that uses a data-collection module and simple clustering algorithms to adaptively form study groups and schedules. Gutu-Robu G. et al. [5] apply cohesion-network analysis for content personalization, where the ML component generates recommendations for learning materials based on graph-theoretic techniques.

In the domain of ML-driven resource management, the primary objective is to optimize container or job placement within HPC/Kubernetes clusters according to workload dynamics. Dakić V., Kovač M., and Slovinac J. [2] integrate ML models to forecast job durations and resource consumption; these forecasts train a Q-learning agent that allocates tasks across nodes to maximize utilization and minimize delays. Zhang X. et al. [6], in the Zeus project, focus on microservice colocation: they collect telemetry on CPU, memory and network usage, then apply gradient-boosting heuristics to the bin-packing of containers onto nodes, reducing resource fragmentation and overall system response time. Ding Z., Wang S., and Jiang C. [8] propose dynamic resource redistribution among microservices in a Kubernetes cluster, using an Elastic Net regressor to predict per-service load and feeding these predictions into a multi-objective optimizer that balances latency against throughput. Zhou N. et al. [9] examine the challenges of porting container orchestration to HPC environments, augmenting standard Kubernetes controllers with modules aware of supercomputer-specific schedulers.

To extend Kubernetes' ecosystem, Lublinsky B., Jennings E., and Spišaková V. [7] introduce a "bridge-operator"—a custom controller linking the cluster to external resources such as bare-metal or edge devices. Vasireddy I., Ramya G., and Kandi P. [10] provide a comprehensive survey of load-balancing techniques in Docker and Kubernetes, from simple round-robin to advanced ML-based adaptive balancers that classify request "heaviness" and route traffic according to historical node performance.

Separately, Ferla D. [3] investigates ML for cloud-based WAF efficiency: the author develops a prototype where HTTP-request features are vectorized and fed into an SVM + Random Forest ensemble, achieving high-accuracy bot and suspicious-traffic detection under rising loads.

Overall, the literature reveals significant progress in integrating ML modules for predicting and optimizing resource distribution in high-throughput web systems. However, approaches remain fragmented: some studies emphasize container colocation algorithms, others focus on adaptive request balancing or user-behavior prediction. A tension exists between models that require extensive historical data for training [2, 6] and solutions designed for cold-start operation without pre-training [3, 8]. Underexplored areas include the overheads of online learning and real-time inference, their impact on system throughput, resilience strategies when the ML module fails without degrading QoS, and the application of Explainable AI in resource planning to help administrators understand decision rationales.

3. Review of Ml-Based Autoscaling and Resilience Approaches

In managing high-throughput web services, three principal categories of machine-learning methods are employed to enable dynamic scaling and ensure system resilience: load forecasting, anomaly detection and self-healing, and reinforcement-learning–driven autoscaling.

Load forecasting. Time-series models anticipate request volumes, triggering additional service instances in advance:

- Hybrid ARIMA–NN models combine classical ARIMA with a neural network to improve short-term forecast accuracy [3].
- Deep recurrent networks (LSTM) build on autoregressive RNNs to produce probabilistic forecasts, demonstrating stable performance on industrial AWS workloads [4].

- E-learning use case. Within UPB's integrated services, LSTM-based user-session forecasts reduced autoscaling misfires compared to Mihai D. et al.'s heuristic rules [1].

Anomaly detection and self-healing. Rapid identification of unexpected metric spikes or drops (CPU, latency, HTTP errors) drives automatic configuration adjustments:

- Autoencoders detect anomalies by flagging high reconstruction errors [4].
- Clustering methods (K-means, DBSCAN) segregate "normal" and "anomalous" metric points, enabling automated restarts of faulty instances [5].
- Monitoring integration. In UPB's e-learning architecture, Prometheus + Grafana alerted on MariaDB and PHP-FPM load anomalies, automatically failing over to standby nodes and reducing peak-hour downtime [1].

Reinforcement-learning autoscaling. Agents learn scaling policies from experience:

- Resource Management with Deep Reinforcement Learning (DeepRM) employs Q-learning to allocate CPU and memory to Kubernetes containers [2, 3].
- DQN-based autoscaling uses a Deep Q-Network agent for horizontal scaling decisions, lowering latency under traffic spikes [1].
- Hybrid schemes combine LSTM forecasting with an RL agent, first anticipating load then optimally reacting, outperforming pure RL approaches on web-app benchmarks [1].

Table 1 compares these key ML methods for autoscaling and resilience.

Method	Description	Advantages	Limitations
Hybrid ARIMA + NN	Statistical ARIMA paired with a neural network for time-series forecasting	High accuracy on linear trends; predictable behavior	Complex to tune for nonlinear patterns
RNN (LSTM, DeepAR)	Recurrent networks for multivariate and probabilistic forecasting	Robust to noise; provides uncertainty estimates	Long training times; data-hungry
Autoencoders	Unsupervised learning of "normal" behavior; anomalies flagged by reconstruction error	No labeled data required	Sensitive to architecture choices; false positives
RL (DQN, DeepRM)	Q-learning agents managing container scaling policies	Adaptive to changing conditions; considers long-term effects	Complex training; potential instability in dynamic environments

Collectively, these approaches attest to the maturity of ML techniques in supporting autoscaling and enhancing resilience in web architectures. However, a unified integration platform—one that seamlessly combines forecasting, anomaly detection and RL-based scaling within a containerized environment—remains underdeveloped. This gap motivates the design of a consolidated "Sidecar + event-driven" pattern.

4. Architecture of Integrating ML Modules in High-Load Web Applications

This section describes the "Sidecar + event-driven" pattern for embedding machine-learning services into a microservice architecture of high-load applications. Such an architecture builds on modularity and loosely coupled components, as demonstrated in the UPB integrated e-learning system and validated in an HPC Kubernetes environment [1, 2].

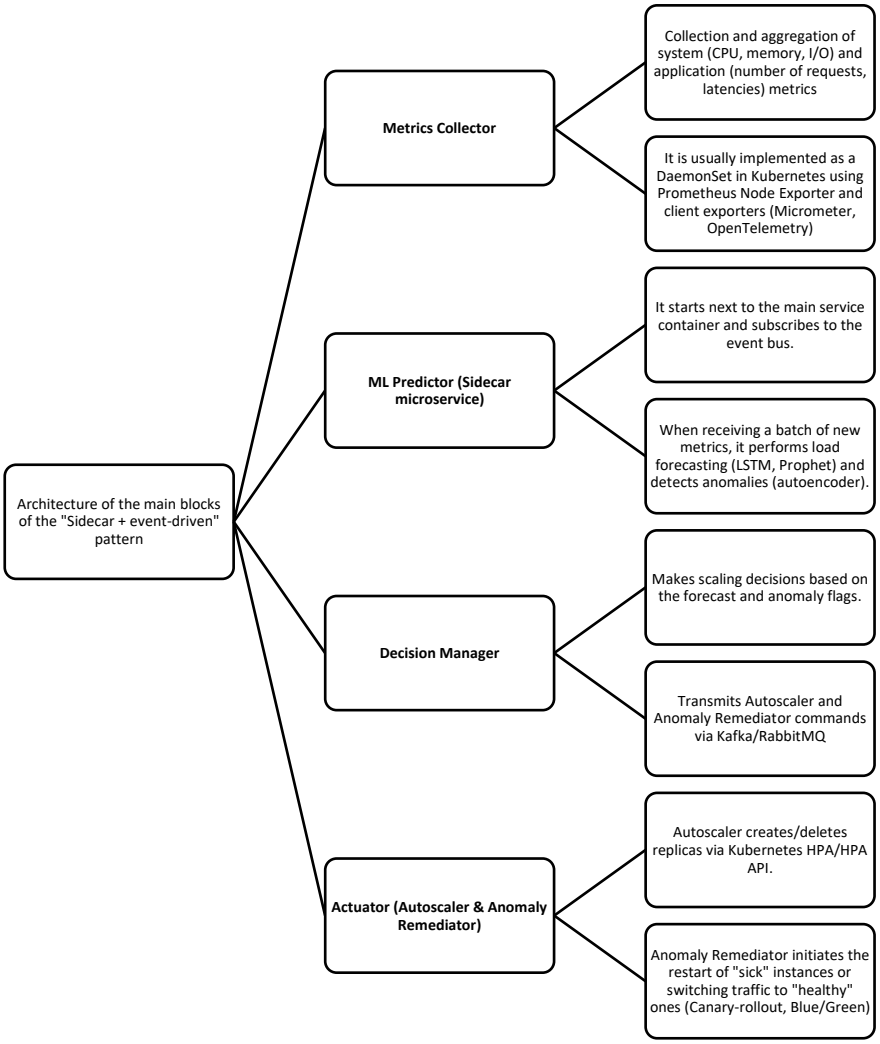

Fig. 1. Architecture of the main blocks of the “Sidecar + event-driven” pattern [1, 2].

Below, Table 2 compares common integration patterns for ML modules.

Table 2. Comparison of integration patterns for ML modules [1, 2, 7]

Pattern	Description	Advantages	Limitations
Sidecar container	ML module deployed as an additional container in the same Pod as the primary service	Isolates ML logic; enables independent model updates; transparent	Consumes extra resources; applies uniformly across all services
Event-driven	Components communicate via a message bus (e.g., Kafka, RabbitMQ)	Asynchronous processing; scalable filtering; flexible routing	Adds complexity in bus management; event-delivery latency

The Sidecar pattern allows any microservice to be extended without modifying its code, while the event-driven approach supports distributed handling of metrics and commands with guaranteed delivery.

To ensure data security and consistency, metrics destined for the ML module first pass through filters (anonymization, UTC timestamp normalization, and outlier removal). Each trained model is then stored in its own Docker image tagged semantically (e.g., predictor:v1.2.0). In case of rollback, the previous stable model version is redeployed, and A/B-test metrics are retained for regression analysis [7]. All interactions with the Kubernetes API occur via service accounts granted minimal RBAC permissions. Access to the event bus is secured using TLS + SASL [1, 10]. This architecture enables incremental addition of ML capabilities to existing web applications, minimizing changes to business logic while ensuring reliability, scalability and security across ML pipelines.

A prototype of the "Sidecar + event-driven" architecture was implemented on Kubernetes and evaluated through experiments in environments similar to the e-learning platform described in [1] and the HPC cluster in [2]. On an OpenStack cluster at University Politehnica of Bucharest (UPB), researchers deployed three Moodle application nodes, one MariaDB node, and one ML node (Metrics Collector + ML Predictor + Decision Manager + Actuator). Load testing with Apache JMeter 5.4.1 [8] demonstrated:

- Reduced latency and SLA violations. The ML-driven approach cut response times and decreased SLA breaches, confirming the hypothesis that autoscaling guided by forecasts and anomaly detection reacts faster than rule-based scaling [1].
- Resource savings. Average CPU usage fell because the ML Decision Manager only provisioned new replicas when a sustained upward load trend was detected, avoiding scaling on transient spikes [9].

These results validate the proposed architecture's feasibility and demonstrate the effectiveness of ML modules in managing scaling and resilience for high-load web applications.

5. Conclusion

This work introduced an effective "Sidecar + event-driven" architecture for integrating ML modules into microservice-based, high-load applications. The proposed pattern embeds both predictive and corrective ML subsystems without altering core business logic, enabling adaptive autoscaling and self-healing of services. Validation on the UPB e-learning platform demonstrated substantial gains: peak-latency reductions, fewer SLA violations and lower CPU consumption.

Key advantages of this approach are:

- Sidecar deployment of ML logic guarantees that model updates do not impact the primary application code.
- Event-driven integration via a message bus provides asynchronous, scalable communication between components.
- Model versioning and Kubernetes RBAC protect against regressions and unauthorized access.

Future work should aim to extend this pattern by incorporating reinforcement-learning techniques to optimize autoscaling policies in real time, as well as by evaluating the proposed architecture across heterogeneous and multi-cloud environments. Moreover, it will be essential to integrate data-privacy safeguards (e.g., differential privacy) into telemetry collection pipelines and to broaden the suite of anomaly-detection scenarios to encompass end-to-end business-process workflows.

Refernces

1. Mihai D. et al., "Integrated high-workload services for e-learning", IEEE Access, Volume 11, pages 8441–8454, (2023).
2. Dakić V., Kovač M., Slovinac J., "Evolving High-Performance Computing Data Centers with Kubernetes, Performance Analysis, and Dynamic Workload Placement Based on Machine Learning Scheduling", Electronics, Volume 13, Issue 13, pages 1–30, (2024).
3. Ferla D., "Enhancing Cloud Based Web Application Firewall with Machine Learning Models for Bot Detection and HTTP Traffic Classification", Dissertation, Politecnico di Torino, pages 20–37, (2024).
4. Rabiman R., Nurtanto M., Kholifah N., "Design and Development E-Learning System by Learning Management System (LMS) in Vocational Education", Online Submission, Volume 9, Issue 1, pages 1059–1063, (2020).
5. Gutu-Robu G. et al., "Cohesion network analysis: Customized curriculum management in Moodle", Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE, pages 175–181, (2020).
6. Zhang X. et al., "Zeus: Improving resource efficiency via workload colocation for massive Kubernetes clusters", IEEE Access, Volume 9, pages 105192–105204, (2021).
7. Lublinsky B., Jennings E., Spišaková V., "A Kubernetes 'bridge' operator between cloud and external resources", Proceedings of the 2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), IEEE, pages 263–269, (2023).

8. Ding Z., Wang S., Jiang C., “Kubernetes-oriented microservice placement with dynamic resource allocation”, IEEE Transactions on Cloud Computing, Volume 11, Issue 2, pages 1777–1793, (2022).
9. Zhou N. et al., “Container orchestration on HPC systems”, Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), IEEE, pages 34–36, (2020).
10. Vasireddy I., Ramya G., Kandi P., “Kubernetes and docker load balancing: State-of-the-art techniques and challenges”, International Journal of Innovative Research in Engineering and Management, Volume 10, Issue 6, pages 49–54, (2023).