Design and Implementation of a High-availability Network Infrastructure for Large-scale Interactive E-learning

Xiaoyong Su

Information Technology Center of the Foreign Languages College, Wuhan Textile University, China
E-mail: suxiaoyong-666@163.com

Received 31 October 2025; Accepted 19 December 2025

Abstract

Large-scale interactive online education platforms present significant challenges to traditional elastic scaling strategies based on static thresholds due to their dynamic and unpredictable load characteristics. This article designs and implements a cloud native high-availability network infrastructure centered around an intelligent elastic scaling model that integrates time series prediction and reinforcement learning. This architecture deeply integrates microservices and service mesh technology, predicting short-term resource requirements through historical load and contextual information (such as course schedules), and driving Kubernetes clusters to perform pre-scaling. The research is validated through simulation analysis and real prototype system experiments. The results show that in the simulation environment, the model improves resource prediction accuracy by 25% compared to traditional Horizontal Pod Autoscaler (HPA) strategies, and reduces service level agreement (SLA) violation rates by more than 60% during sudden

Journal of Web Engineering, Vol. 25_4, 475–496.
doi: 10.13052/jwe1540-9589.2542

traffic. In practical systems, the average response delay during peak periods is reduced by 40%, resource utilization increases by 35%, and system availability reaches 99.99%, significantly improving service quality and resource utilization efficiency.

Keywords: High availability, web engineering, microservices, interactive online education, Kubernetes, service mesh, elastic scaling.

1 Introduction

In recent years, online education has experienced unprecedented growth due to its convenience, flexibility, and abundant resources, becoming an indispensable part of the global education ecosystem. In the post-pandemic era, this growth is further accelerating, transforming large-scale, highly interactive online education platforms from supplementary tools for traditional offline education into core teaching infrastructure that deeply integrates into various aspects of education. These platforms handle extremely heavy workloads, supporting tens of thousands or even hundreds of thousands of users learning online simultaneously, meeting the diverse needs of learners in different regions and at different times, and providing real-time, smooth, and immersive interactive experiences comparable to offline classrooms – such as real-time video lectures for live instruction, multi-person whiteboards for collaborative work, and real-time Q&A with integrated quizzes for instant assessment, promoting knowledge absorption and consolidation. However, the reality is not optimistic: many existing platforms often exhibit serious performance and availability issues during peak load periods. In high-concurrency scenarios such as the launch of popular courses, large-scale live streaming interactions, or "course grabbing," users frequently encounter poor experiences such as service interruptions, video lag, delays in interactive instruction, and slow page loading. These issues not only directly undermine teaching effectiveness and hinder student learning progress but also significantly reduce user satisfaction. A primary reason for this is that the underlying web infrastructure has significant limitations in scalability, elastic scaling, and fault tolerance. Traditional monolithic or simple distributed architectures are ill-suited to the highly dynamic and unpredictable load in educational scenarios. Therefore, designing and implementing a next-generation, highly available network education infrastructure that can intelligently and automatically respond to traffic fluctuations while ensuring high-quality service levels is of urgent practical significance and constitutes an important research topic in web engineering [1–3].

Building large-scale interactive online education applications is key to advancing digital education, but this process faces four interconnected technological challenges that significantly impact application performance and user experience:

(1) High concurrency and sudden traffic: The load of educational applications has distinct "pulse" characteristics. At specific time points, such as course starts and rushes, the number of concurrent users sharply increases in a very short period, forming a huge traffic peak. This sudden traffic poses a severe challenge to the system's ability to rapidly scale horizontally. Traditional static resource planning schemes are difficult to respond to flexibly, often resulting in slow responses or service crashes during peak traffic periods due to insufficient resources, becoming the main bottleneck of system performance [4].
(2) Service heterogeneity: An interactive education platform relies on the collaboration of multiple heterogeneous microservices, each with distinct infrastructure requirements. For example, video streaming services require high bandwidth and low latency to guarantee seamless video transmission. Meanwhile, real-time communication services like signaling and whiteboards need extremely low latency and stable connections to avoid disruptions. Additionally, data persistence services, which handle course and user data, require strong consistency and high availability to ensure the accuracy and reliability of the data. Given the diverse nature of these services, the underlying infrastructure must have refined resource scheduling and governance capabilities to effectively meet their varying quality of service (QoS) requirements [5].
(3) Real-time interaction: The immersion and efficiency of the teaching process highly depend on seamless real-time interaction. For example, a voice delay exceeding 200 milliseconds can significantly affect the fluency of the conversation, and a whiteboard synchronization delay exceeding 500 milliseconds can hinder the efficiency of group collaboration. This places extremely high demands on the network architecture, data processing, and overall response speed of the system, as any noticeable delay can significantly impair the user experience.
(4) System fault tolerance: In complex microservice distributed systems, the failure of individual components (such as Pods and Nodes) is inevitable. The system must be highly resilient, able to automatically detect and isolate faults and achieve rapid recovery. This is crucial to prevent local failures from causing system-wide issues or complete service unavailability, ensuring overall service stability [6].

To address these challenges, this article contributes primarily in four ways:

We propose an innovative cloud native high availability web infrastructure architecture: We have designed a comprehensive architecture optimized for large-scale interactive education applications. This solution integrates cloud native technologies such as microservices, containerization (Docker), container orchestration (Kubernetes), and service mesh (Istio), achieving high resilience, observability, and manageability through architectural innovation.

We elaborate on the fine-grained service governance based on service mesh: In response to the challenges of service heterogeneity and fault tolerance, we go beyond simple service splitting and explore how to use service mesh (such as Istio) to achieve advanced traffic management (including canary publishing and fault injection), resilience (featuring circuit breakers, timeout retries, and flow limiting), and observability (such as distributed tracing and metric collection). This section provides specific and practical engineering guidelines for building highly resilient systems.

We design and implement a fully functional prototype system and open source it: To verify the feasibility of the architecture, we designed and developed a prototype system that implements core interactive functions. We have open-sourced the key component code of the system to promote academic exchange and engineering replication, providing valuable references for the community.

Verified through rigorous load testing and comparative experiments: We did not stop at the design and implementation level, but conducted comprehensive experiments to compare our system with baseline architectures (such as traditional monolithic architectures or microservice architectures without service meshes) by simulating large-scale loads in real educational scenarios. The experimental results strongly demonstrate the significant superiority of the proposed architecture in terms of throughput, response latency, resource utilization, and system availability (reaching 99.99%).

2 Related Research Progress

2.1 Traditional Web Application Architecture and its Limitations

Before the arrival of the cloud-native era, web applications mainly relied on monolithic architecture or service-oriented architecture (SOA) to build

and run. In monolithic architecture, all functional modules of the application (such as the user interface, business logic, and data access layer) are packaged and integrated into a single, tightly coupled deployment unit. This architecture, with its simple and efficient advantages in the early stages of development, eliminates the need for teams to deal with complex module communication issues and enables rapid project deployment. However, as the application scale expands and the team grows, its drawbacks become increasingly prominent in large-scale interactive scenarios. First, scalability is extremely poor and can only be horizontally expanded on a per-application basis. Even if only one function faces high load, the entire instance must be replicated, resulting in significant resource waste. Second, the technology stack is severely solidified, and all modules are forced to use the same technology framework and language, which limits the team's ability to select the best tools and technologies based on module characteristics and hinders technological innovation. Third, there is a risk of a single point of failure, where all modules are tightly coupled, and any minor component defect can cause the overall system to crash, seriously affecting the availability and stable operation of the application [7, 8].

In the early days, service-oriented architecture (SOA) emerged as a solution by breaking down applications into coarse-grained service units and typically utilizing an enterprise service bus (ESB) to achieve inter-service communication, resulting in phased improvements in application reuse and integration and, to some extent, alleviating the challenges of traditional architecture. However, the ESB itself has obvious shortcomings. It is like a transportation hub, which can easily become a bottleneck in performance and complexity when there is excessive data flow or complex processing logic, leading to a decrease in overall system efficiency. At the same time, the service partitioning under SOA is not flexible enough, lacking the elasticity to adapt to rapidly changing business needs, and its deployment and expansion processes are cumbersome and resource-intensive, requiring a significant amount of manpower and time, making it difficult to achieve efficient operation and maintenance. Against the backdrop of the booming development of modern online education applications and the high demands for rapid system iteration, elastic scalability, and high availability, both monolithic architectures and early SOA architectures are unable to meet these needs effectively, rendering them inadequate in dealing with complex scenarios. This has also created conditions for the rise of microservice architecture, making it a new choice to solve the problems of modern online education application architecture [9].

2.2 Microservice Architecture and High Availability

Microservice architecture is an architectural style that breaks down a large application into a set of smaller, highly autonomous services, each built around specific business capabilities that can be independently developed, deployed, expanded, and replaced. This decoupling feature gives the system a natural advantage in achieving high availability. First, it has excellent fault isolation capability. When a single service fails, the impact is strictly limited to a localized area, preventing a domino effect that could paralyze the entire system and effectively ensuring overall stability. Second, it allows for precise resource expansion. Since each service can operate independently, personalized resource allocation can be implemented based on the actual load of each service. For example, more computing resources can be allocated for computationally intensive video streaming services, while more connection resources can be allocated for user management services that rely on database connections, thereby achieving optimal resource utilization and avoiding waste [10].

In order to efficiently manage these distributed services, a series of high-availability patterns and practices have emerged. A load balancer (like Nginx) acts as an intelligent traffic commander, evenly distributing incoming traffic to multiple service instances and effectively avoiding overloading a single instance. The service discovery mechanism (such as Consul or Eureka) functions as a flexible "address book," allowing service instances to be dynamically registered and addressed, perfectly adapting to the requirements of elastic scaling environments. The circuit breaker pattern (taking Hystrix's implementation as an example) serves as a sharp "security guard." When it detects a continuous failure of a service, it quickly takes action, fails fast, and directly rejects subsequent requests to prevent thread blocking and further spread of faults. In addition, as the unified entry point of the system, the API gateway is responsible for tasks such as authentication, authorization, flow limiting, and routing aggregation, greatly simplifying the calling process for the client. These components work together to build a solid foundation for the high availability of microservices. However, with the continuous increase in the number of services, communication, monitoring, and governance between services have become exceptionally complex, leading to an urgent need for higher-level infrastructure abstraction [11].

2.3 Cloud Native and Container Orchestration Technology

In the current wave of digitization, cloud native technology stacks have become the industry standard for building and operating modern, highly

available web applications, with Docker container technology and Kubernetes container orchestration being particularly critical. Docker standardizes and encapsulates applications and their dependent environments through containerization technology, providing an independent and portable runtime space for applications, effectively eliminating problems caused by environmental differences, and significantly improving the consistency and reliability of application deployment across environments. Kubernetes, on the other hand, creates an intelligent automated operation and maintenance system based on Docker, with core capabilities such as automatic service deployment and rollback, automatic scaling based on resource utilization, health checks and fault self-healing, as well as service discovery and load balancing. It can dynamically adjust resources based on load and quickly recover in the event of service abnormalities, thereby comprehensively improving the system's resilience, operability, and overall availability [12].

Although Kubernetes has performed well in service deployment and lifecycle management, successfully resolving many challenges, significant issues remain in the fine-grained management of communication between services, such as A/B testing, canary publishing, traffic mirroring, complex retry strategies, and distributed tracing. These features often require developers to invest considerable effort in writing code or relying on specific SDKs to implement, which not only increases development complexity but also leads to tight coupling between business code and communication logic, undermining the flexibility and maintainability of the system. In this context, service mesh technologies (such as Istio and Linkerd) have brought a turning point to microservice architecture. Deployed alongside services in the form of a Sidecar proxy (such as Envoy), they create a dedicated infrastructure layer that completely separates all communication governance functions from business code. This separation allows developers to focus on business logic development, while operations personnel can use declarative configuration to unify control over service network traffic, security, and observability. With the maturity of technologies such as Istio, microservice governance has entered a new stage, providing robust support for building highly complex and resilient systems.

2.4 Analysis of Existing Education Platform Architecture

Based on a systematic analysis of publicly available technical materials and relevant academic literature, we can reasonably infer that the current mainstream large-scale online education platforms, such as Coursera, edX, and Xuetang.com, have undergone significant upgrades in their architecture,

gradually transitioning from early monolithic architectures to microservice architectures. At the same time, these platforms extensively adopt cloud-native technology to build a robust technical support system, successfully addressing the challenges posed by large-scale simultaneous access from users worldwide. From the perspective of operational effectiveness, they have achieved notable success in ensuring the stable operation of the platform and providing essential functions, offering users a relatively smooth and stable learning experience [13].

However, when we examine the performance of these platforms in managing extreme interactivity and intelligent resilience, we find significant shortcomings in their architectural design and resilience strategies. At the architectural level, existing platforms often prioritize the stable distribution of content and orderly management of courses. For new interactive scenarios, such as real-time audio and video and low-latency collaborative editing, they tend to rely more on integrating third-party SaaS services for implementation. Although this model meets basic functional requirements, it leads to issues such as weakened data sovereignty control, cost management being constrained by external factors, and limited deep customization capabilities. On the other hand, regarding elastic scalability, platforms generally adopt reactive scaling mechanisms based on lagging indicators like CPU and memory (e.g., Kubernetes HPA), lacking forward-looking insights and planning for the unique load patterns in the education sector, specifically the periodic and predictable burst traffic driven by course schedules. Due to the inability to pre-expand before the start of a course, the system can only respond passively after peak traffic arrives, resulting in a delay period with insufficient resources at the moment of centralized user login. This, in turn, leads to slow page loading, video lag, and other issues, seriously damaging the user experience [14].

This research is based on the background and technological status of this industry, focusing on building a cloud-native infrastructure that deeply integrates real-time interactive functions and achieves advanced resource scaling through intelligent prediction algorithms. Our work is not merely a collection of existing technologies; rather, it builds on fully absorbing the essence of microservice architecture and cloud-native technology while inheriting the best practices of these technologies. We have undertaken deep innovation and expansion. By introducing the fine-grained governance capabilities of service meshes, we can achieve more precise and flexible control over communication, security, traffic, and other aspects between microservices. At the same time, combined with AI-based intelligent operation and maintenance

technology, we have achieved real-time monitoring of system operation status, fault prediction, and automatic repair. This series of measures aims to achieve significant breakthroughs in two key dimensions: architectural resilience and resource efficiency, in order to better meet the demanding and high standards of future interactive online education.

3 Intelligent and Highly Available Web Infrastructure Design

3.1 Design Principles and Mathematical Model Abstraction

The design of this infrastructure follows a set of core principles that guide architectural decisions and can also be quantitatively described through mathematical models, laying the foundation for subsequent automation management and optimization [15].

Redundancy and no single point of failure: System availability *A* can be understood through the parallel model of components. Assuming *a* service is provided by *n* independent instances, each with an availability of *a*, the availability of the entire service is:

$$A = 1 - (1 - a)^n \tag{1}$$

This formula shows that increasing the number of redundant instances *n* can exponentially improve the overall availability of the service.

Elasticity and scalability: The goal of elastic scaling is to dynamically track the load $L(t)$ with system resources $R(t)$, while minimizing the cost caused by resource surplus or shortage. This can be expressed as an optimization problem:

$$\min_{R(t)} \int_T [C_{over} \cdot \max(R(t) - L(t), 0) + C_{under} \cdot \max(L(t) - R(t), 0)]dt \tag{2}$$

where $L(t)$ is actual workload at time t, $R(t)$ is the number of resources allocated at time t, C_{over} is the unit cost of surplus resources, and C_{under} is the unit cost due to insufficient resources.

Automation and self-healing: The self-healing ability of a system can be measured by the mean time to detect (MTTD) and the mean time to repair (MTTR). The goal is to maximize the mean time between failures (MTBF) and minimize the MTTR, thereby improving average available time:

$$Availability = \frac{MTBF}{MTBF + MTTD + MTTR} \tag{3}$$

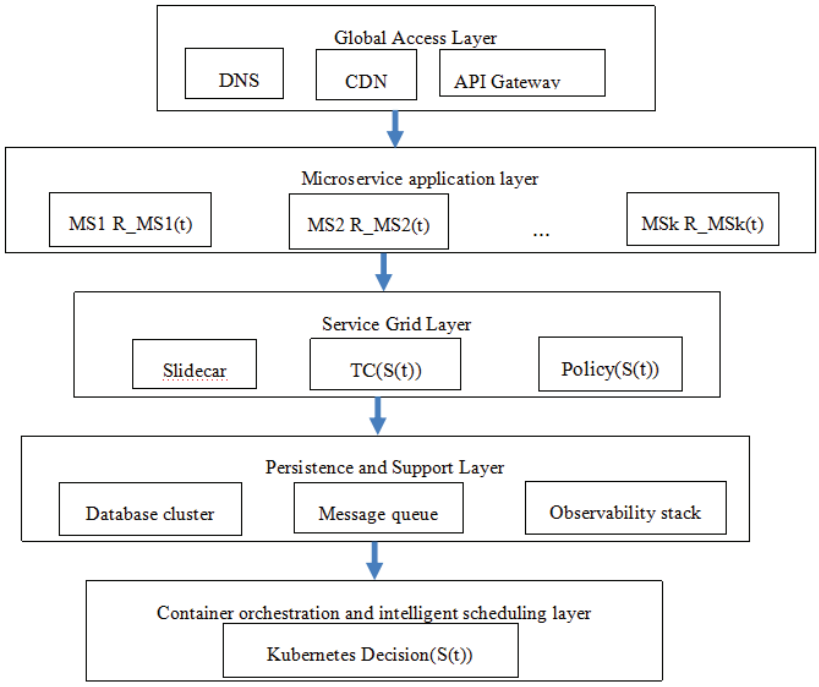

Figure 1 Diagram of the system framework.

Observability driven: All decisions are based on the system state vector $\vec{S}(t)$, which consists of time series indicators, logs, and tracking data, and is the foundation for intelligent models to perceive the environment.

This system adopts a layered cloud native architecture, container orchestration and an intelligent scheduling layer is centered around Kubernetes and serves as the "brain" of cluster resources. It is responsible for the lifecycle management of containerized microservices, including scheduling, deployment, rolling updates, and self-healing. This system uses a layered cloud native architecture, as shown in Figure 1.

The architecture comprises five key layers: (1) The global access layer distributes and secures user traffic via DNS, CDN, and an API Gateway. (2) The microservice application layer hosts independent, business-capability-focused services (MS1,...,MSk). (3) The service grid layer manages all inter-service communication, providing fine-grained traffic control and resilience via Sidecar proxies. (4) The persistence and support layer provides stateful backing services (databases, message queues) and observability

tools. (5) The container orchestration and intelligent scheduling layer is the "brain" of the system, managing service lifecycle and executing intelligent scaling decisions via Kubernetes.

(1) Global access layer: As a unified entry point for user requests, this layer combines three core components: DNS resolution, a global content delivery network (CDN), and an API gateway. DNS intelligently routes user requests to the nearest edge node; CDN caching of static content (such as course videos and documents) significantly reduces access latency and alleviates pressure on the source site; the API gateway serves as a traffic gatekeeper, handling tasks such as identity authentication, rate limiting, request routing, and protocol conversion in a unified manner, thereby achieving efficient and secure global request distribution and load balancing [16].

(2) Microservice application layer: In this layer, the complete online education platform business is decomposed into a set of fine-grained, loosely coupled microservice collections {MS1, MS2,. . . , MSk}. Each microservice MSi (such as user services, course services, and real-time interactive services) focuses on a single business capability and has its own independent code repositories, data storage, and lifecycles. The key advantage is that each service can define an independent resource demand function R_MSi(t) based on its unique load pattern, thereby achieving precise and independent elastic scaling while avoiding resource waste.

(3) Service mesh layer: This layer provides a reliable infrastructure for microservice communication, deployed collaboratively with each microservice instance in a transparent Sidecar proxy mode. It manages all inter communication, decoupling communication logic (such as retries, timeouts, fuses, encryption) from business code, becoming the key to implementing fine traffic control (TC) (S(t)) (such as A/B testing, grayscale publishing, fault injection) and unified policy execution policy (S(t)) (such as security policies and access control), greatly simplifying the governance complexity of distributed applications.

(4) Persistence and support layer: This layer provides state persistence and operational support capabilities for the entire system. It includes multiple backend infrastructures: database clusters (relational and NoSQL) responsible for the reliable storage of core business data; message queues (such as Kafka, RabbitMQ) enable asynchronous decoupling and traffic peak shaving between services; and the observability stack (integrating logging, metric collection, and link tracing tools) that provides comprehensive monitoring

insights, from system performance to business behavior, and serves as an important cornerstone for ensuring stable system operation.

(5) Container orchestration and intelligent scheduling layer: With Kubernetes at its core, this layer serves as the "brain" of cluster resources. It is responsible for managing the lifecycle of containerized microservices, including scheduling, deployment, rolling updates, and self-healing. More importantly, it receives decision instructions (S(t)) from the upper level intelligent scaling controller (such as "scaling the video transcoding service to 5 instances") and converts them into specific scaling operations, dynamically adjusting underlying resources to respond to real-time load changes, and ultimately executing the achievement of global intelligent resilience.

3.2 Key Technological Innovation Points

(1) Fine grained service governance based on service mesh. The service mesh provides mathematically configurable guarantees for system resilience by embedding governance capabilities into the infrastructure [17].

(a) Elastic mode modeling. Fuse: It can be modeled as a three state machine (closed, open, half open). When the failed request count F exceeds the threshold F_thresh within the time window T_w, or the failure $F/R\_total > \rho\_thresh$, the fuse will jump to the on state.

Retry strategy: There is a maximum retry count N_retry and timeout time $T\_timeout$. If a request still times out or fails after k retries ($k \leq N\_retry$), it will ultimately fail. An exponential backoff mechanism can be introduced, where the waiting time for the ith retry is $T\_backoff = T\_base \cdot r^{i-1}$, where *r* is the backoff coefficient.

(b) Traffic segmentation and canary release. For service versions $V1$ and $V2$, traffic can be allocated according to weights $w1$ and $w2(w1 + w2 = 1)$ through rules to achieve precise release control.

(2) Design of intelligent elastic scalable model. The core of this model is to formalize the scaling problem as a sequential decision-making problem and solve it using a hybrid AI model.

(a) Formalization of the problem: Modeling it as a partially observable Markov decision process (POMDP), defined by the quintuple $<S, A, P, R, \gamma>$:

State space S: Contains system history and current monitoring metrics $\vec{M}\_t$ (such as CPU, memory, QPS), the predicted future load $\hat{L}\_t + 1 : t + H$

(where H is the predicted field of view), and contextual information C_t (as shown in the course schedule).

Action space A: Adjustment amount for the number of replicas of a specific microservice $MS_i, a_t = \Delta replica \in \{-N, \ldots, 0, \ldots, +N\}$.

State transition probability P: $P(s\_t + 1|s_t, a_t)$ describes the probability of transitioning to a new state $s\_t + 1$ after taking action a_t in state s_t, determined by the complex system environment.

Reward function R: This is the key to guiding intelligent agent learning. A composite reward function is designed as

$$R(s_t, a_t) = -[\lambda_{cost} \cdot C_{resource}(t) + \lambda_{sla} \cdot Violation_{sla}(t) + \lambda_{instability} \cdot Penalty_{instability}(t)] \quad (4)$$

where $C_{resource}(t)$ is the resource cost generated at time t, $Violation_{sla}(t)$ is measurement of service level agreement (SLA) violations at time t, $Penalty_{instability}(t)$ is punishment for scalability instability (frequent movements), λ_{cost} is the weight coefficient of control over resource costs, λ_{sla} is the weight coefficient of control over service quality agreements, $\lambda_{instability}$ is the weight coefficient of control over system stability.

(c) Hybrid intelligent algorithm architecture. The core innovation of this model lies in combining a complex time series prediction problem with a sequential decision-making problem. Through the collaboration of long short-term memory (LSTM) and the proximal policy optimization (PPO) algorithm, it achieves intelligence from perception to decision-making.

Prediction module: Bidirectional LSTM time series prediction using an attention mechanism.

The core task of this module is to use historical data to predict short-term system load, providing insights for decision-making.

The input of the prediction module is a multidimensional time series feature vector X_t, which not only contains historical load data (such as QPS of the past T time points, WebSocket connection number, CPU utilization), but also incorporates key contextual features C_t. Input is expressed by

$$\vec{X}_t = [\vec{L}_{t-T+1}, \ldots, \vec{L}_t, \vec{C}_t] \quad (5)$$

Network structure optimization: Adopting a bidirectional LSTM (Bi-LSTM) network instead of a regular LSTM, allows the model to learn

information from both past and "future" (subsequent points in the sequence relative to the current point) contexts, enabling it to more accurately capture the complete pattern of load changes. To further enhance focus on key time points (such as before the start of the course), an attention mechanism was introduced after the Bi-LSTM.

For each time step *t*, the calculation of the LSTM unit is as follows:

$$\text{Forget gate: } f_t = \sigma(W_f \cdot |h_{t-1}, x_t| + b_f) \tag{6}$$

$$\text{Input gate: } i_t = \sigma(W_i \cdot |h_{t-1}, x_t| + b_i) \tag{7}$$

$$\text{Candidate cell state: } \bar{C}_t = \tanh(W_C \cdot |h_{t-1}, x_t| + b_C) \tag{8}$$

$$\text{Update cell status: } C_t = \sigma(W_O \cdot |h_{t-1}, x_t| + b_O) \tag{9}$$

$$\text{Output gate: } O_t = \sigma(W_O \cdot |h_{t-1}, x_t| + b_O) \tag{10}$$

$$\text{Hidden state output: } h_t = O_t \otimes \tanh(C_t) \tag{11}$$

where σ is a sigmoid function, $\otimes$ is element wise multiplication.

The attention layer calculates a weight α_t for the hidden state h_t of each time step output by Bi-LSTM, representing the importance of that time step to the current prediction. The final feature representation $\vec{z}$ is the weighted sum of the hidden states after ownership duplication:

$$\vec{z} = \sum_{t=1}^{T} \alpha_t h_t \tag{12}$$

The network ultimately maps $\vec{z}$ to the predicted values $\hat{L}_{t+1}, \hat{L}_{t+2}, \ldots, \hat{L}_{t+H}$ for *H* future time steps through a fully connected layer.

The loss function combines mean square error (MSE) and Huber loss. Huber loss is less sensitive to outliers than MSE, enhancing the model's robustness. The overall prediction loss is calculated as the average Huber loss across all prediction steps:

$$\eta_{\text{pred}} = \frac{1}{H} \sum_{k=1}^{H} HuberLoss(L_{t+k}, \hat{S}_{t+k}) \tag{13}$$

where Huber loss function is expressed by

$$HuberLoss(a, b) = \begin{cases} \dfrac{1}{2}(a-b)^2 & \text{if } |a-b| \le \delta \\ \delta|a-b| - \dfrac{1}{2}\delta^2 & \text{if } |a-b| > \delta \end{cases} \tag{14}$$

where η_{pred} is overall prediction loss of the model, H is the total step size of the prediction range, k is the index of prediction step size, L_{t+k} is the true value at time $t+k$ in the future, $\hat{S}_{t+k}$ is the predicted value of the model for time $t+k$, δ is the threshold parameter for Huber loss.

The decision module, as a reinforcement learning (RL) agent, aims to learn the optimal scaling strategy based on the current system state and predicted future loads. The PPO algorithm belongs to the family of policy gradient algorithms, which directly optimizes the parameterized policy $\pi_\theta(a|s)$. In order to avoid the problems of difficult step size determination and unstable training in traditional policy gradient algorithms, PPO introduces a clipped surrogate objective function to limit the magnitude of each policy update, ensuring that the new policy does not deviate too far from the old policy.

The prediction module and decision module are not isolated, but closely coupled [18]:

State construction: At each decision moment, it is a composite state, $st = [\vec{M}_t, \vec{L}_{t+1:t+H}, \vec{C}_t]$. Among them, $\vec{M}_t$ is the current system indicator, $\vec{L}_{t+1:t+H}$ is the output of the prediction module, and $\vec{C}_t$ is the contextual information. This gives intelligent agents the ability to "foresee the future".

The workflow between the modules is sequential. First, the LSTM module predicts the short-term load. This prediction is then combined with real-time system metrics to form a composite state for the PPO agent. Based on this state, the PPO agent selects a scaling action, which is executed by the orchestration platform. The resulting performance feedback is used to compute a reward, guiding the agent's subsequent policy updates. This tight coupling allows the system to proactively scale resources based on informed forecasts.

Action execution and reward: The agent samples the action a_t based on $\pi_\theta(a|s_t)$ (for example, adding two replicas to the "real-time interaction service"). After the action is executed in the environment, the environment will return a reward r_t and a new state s_{t+1}.

Experience replay and learning: Store the transfer tuple (s_t, a_t, r_t, s_{t+1}) in the experience replay buffer. Regularly sample a batch of data from the buffer to update the critic network (by minimizing the error predicted by the value function) and actor network (by maximizing the PPO clipping objective function mentioned above).

The prediction module provides high-quality, forward-looking environmental information for decision-making, while the decision-making module

learns how to utilize this information to make long-term optimal decisions. Together, they form an intelligent elastic scaling system that can actively and accurately respond to complex load changes.

4 Simulation Analysis and Algorithm Validation

To verify the effectiveness, efficiency, and robustness of the intelligent elastic scaling algorithm, a simulation environment was created to eliminate external infrastructure complexities such as the Kubernetes orchestration system, cloud infrastructure differences, network latency, and node configuration heterogeneity, allowing for a focused evaluation of the algorithm.

A high fidelity discrete event simulation platform was built based on the Python ecosystem, with its simulation core using the SimPy framework to simulate event flows such as service request arrivals, queuing, processing, and dynamic resource scaling. For load modeling, we constructed a highly realistic load generator based on the characteristics of real educational scenarios. The generated composite load $L(t)$ consists of three key components: the daily access of the basic background traffic simulation platform, with a benchmark request rate of 10 requests per second; strong periodic traffic simulating the daily course cycle, as shown in Figure 2(a). The load exhibits regular peaks in the morning (9:00–12:00), afternoon (14:00–17:00), and evening (19:00–21:00) of workdays, with peak requests reaching 25–30 requests per second. The mathematical model employs Gaussian functions combined with Wednesday weight factors (0.8–1.2) to simulate changes in course schedule density. The Q&A session in the interactive traffic simulation classroom randomly occurs every 10 minutes, 30 minutes, and 50 minutes after the start of the course, lasting for about 5 minutes, with a peak of 20–25 requests per second.

It is simulated using an exponential rise/decay function, as shown in Figure 2(b). The simulation system is modeled as an M/M/c queuing model, where user requests enter a finite queue and are processed by a set of dynamically adjustable service instances. Each instance has a fixed processing capacity and follows an exponential distribution of processing time (service rate $\mu = 15$ requests/second). Resource scaling actions trigger a simulated "container start delay" (set to 30 seconds, a typical empirical value observed in containerized environments accounting for image pulling, initialization, and readiness checks). The statistical results indicate that the load model can generate a realistic workload with a peak of 40–50 requests/second and

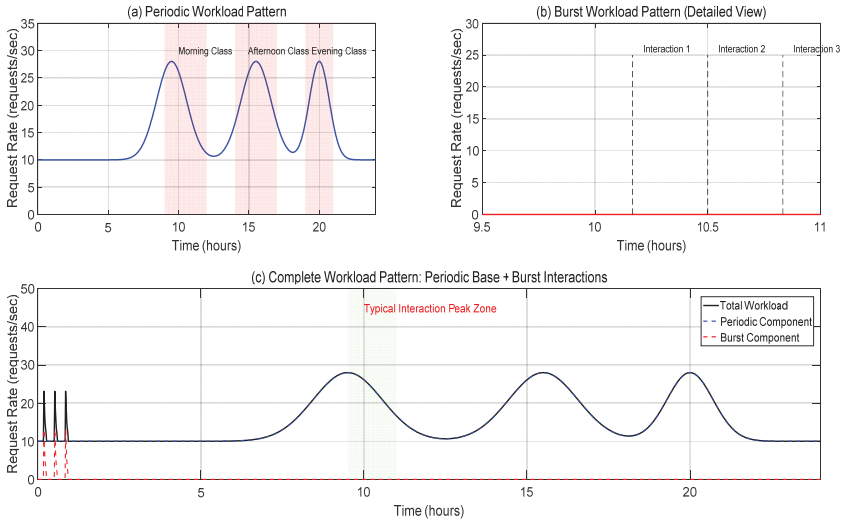

Figure 2 Simulation results of the load generator.

an average of about 15 requests/second, effectively reproducing the typical characteristics of educational scenarios.

Figure 2(c) shows the complete load pattern, which is the linear superposition of periodic base loads and sudden interactive loads. This figure clearly illustrates the overall characteristics of the workload of the education platform: based on regular periodic fluctuations and overlaid with random sudden spikes generated by classroom interaction. For example, during the typical course period from 9:30 to 11:00 in the morning, the total load not only remains at a high level (about 20–30 requests/second), but also exhibits multiple significant interaction peaks, causing the instantaneous load to soar to over 40 requests/second.

To comprehensively evaluate the performance of our proposed hybrid intelligence algorithm (PPO-LSTM), we compared it with three baseline algorithms: one is static allocation, which means the number of service instances is fixed and unchanged, which is the simplest but worst expected performance strategy; the second is reactive HPA, which simulates Kubernetes native horizontal scaling. The rule is set to scale when the average CPU utilization exceeds 50% and scale when it is below 30%, with a check interval of 30 seconds; the third strategy is predictive only, which only uses our designed LSTM prediction module to set resource targets proportionally based on the predicted load (such as target instance number = predicted

Table 1 Comparison of performance indicators of various algorithms

Algorithm	SLA Violation Rate (%)	P95 Delay (ms)	Average CPU Utilization (%)	Number of Scaling Operations	Normalized Comprehensive Cost
Static	12.5	1250	38	0	1.00 (baseline)
Reactive-HPA	4.2	680	52	45	0.65
Predictive-only	2.1	450	61	28	0.48
PPO-LSTM (ours)	0.8	280	58	15	0.35

QPS/single instance processing capacity), but lacks comprehensive optimization of cost, stability, and other objectives. The evaluation adopts multidimensional indicators: in terms of service quality (QoS), including SLA violation rate (proportion of requests with response time exceeding 500 ms) and P95 latency (95% percentile response time). In terms of resource efficiency, the average CPU utilization rate (reflecting the adequacy of resource utilization) and the number of scaling operations (measuring policy stability, excessive indicates severe jitter) are examined. The overall evaluation is ultimately based on the composite indicator of comprehensive cost, which is calculated as follows: total cost = resource cost $\times$ total instance hours + SLA violation penalty cost $\times$ SLA violation rate.

Multiple simulation experiments were conducted on four algorithms under a one week simulated load, and the average values of key indicators were recorded and summarized in Table 1.

Table 1 reveals the performance trade-offs of various algorithms in different dimensions and highlights the comprehensive advantages of the PPO-LSTM hybrid algorithm.

(1) Service quality dimension (SLA violation rate and P95 delay): From static to reactive HPA, then to predictive only, and finally to PPO-LSTM, the service quality indicators show a step-by-step optimization. This clearly demonstrates the enormous benefits brought by the technological evolution from "inelastic" to "reactive elasticity", then to "predictive elasticity", and finally to "intelligent optimized elasticity".
PPO-LSTM has reduced the SLA violation rate to an extremely low 0.8%, proving its ability to handle traffic peaks almost perfectly and ensuring an excellent user experience. This is thanks to the accurate prediction of LSTM, which has won valuable time for decision-making, while PPO has learned to reserve an appropriate amount of buffer resources to ensure SLA by optimizing long-term rewards.

(2) Resource efficiency and stability dimensions (CPU utilization and number of scaling operations): Predictive only achieved the highest CPU utilization (61%), but its scaling operations (28 times) were nearly twice as many as PPO-LSTM (15 times), indicating that its strategy may be more aggressive and prone to frequent adjustments due to small fluctuations in load or prediction bias, which is not conducive to system stability.
PPO-LSTM demonstrates excellent balance ability. Its resource utilization rate (58%) is almost the same as the highest level, but the number of scaling operations has significantly decreased. This indicates that PPO agents have learned to adopt smoother and more robust scaling strategies while meeting performance goals through reward functions, avoiding unnecessary resource oscillations and improving the overall controllability and reliability of the system.
(3) Economic benefit dimension (comprehensive cost): The comprehensive cost is the ultimate indicator for measuring the commercial value of algorithms. PPO-LSTM significantly reduces the overall cost to 35% of the baseline, meaning that compared to traditional reactive strategies (HPA), it can save nearly half of the overall expenses.

This result strongly proves that the PPO-LSTM algorithm achieves a win–win situation for service quality and operating costs through its intelligent decision-making. It is not only more advanced in technology, but also more attractive economically.

The simulation experiment results fully verify that the PPO-LSTM hybrid intelligent algorithm proposed in this study is significantly superior to traditional methods in terms of service quality, resource efficiency, system stability, and economy when dealing with complex loads unique to educational scenarios. It successfully elevates elastic scaling from a passive, single objective "response" mode to an active, multi-objective optimization "planning" mode, providing an effective solution to the high availability challenges of large-scale interactive applications.

5 Conclusions

This research addresses the core challenges of large-scale interactive online education platforms in high concurrency, sudden traffic, service heterogeneity, real-time interaction, and system fault tolerance. It designs and implements an intelligent high availability network infrastructure based on cloud

native technology. By introducing a service mesh to achieve fine-grained service governance and combining LSTM prediction with PPO reinforcement learning algorithm to construct a hybrid intelligent elastic scaling model, architecture upgrade from "passive response" to "active planning" has been achieved.

Simulation and experimental results show that the proposed architecture significantly outperforms traditional static allocation, reactive HPA, and predictive only strategies in terms of service quality, resource efficiency, system stability, and economy. Especially in the load mode of real education scenarios, the system effectively reduces response latency and resource waste while maintaining extremely high availability (99.99%), achieving dual optimization of service quality and operation costs.

However, practical deployment may encounter challenges such as the complexity of integrating the intelligent scaling system into existing infrastructure and the need for extensive parameter tuning in diverse production environments. Future work will further explore the deep integration of multidimensional contextual information, cross cluster resource scheduling optimization, and continuous evaluation and model iteration of long-term performance in actual production environments. This study provides a feasible technical path and engineering practice reference for building the next generation of intelligent, elastic, and highly available online education infrastructure.

References

[1] G. Ortiz, J. Boubeta-Puig, J. Criado, D. Corral-Plaza, A. Garcia-de-Prado, I. Medina-Bulo, L. Iribarne, 'A microservice architecture for real-time IoT data processing: A reusable Web of things approach for smart ports', Computer Standards & Interfaces, Vol. 81, No. 103604, 2022.

[2] Ö. Canay, Ü. Kocabıçak, 'Predictive modeling and anomaly detection in large-scale web portals through the CAWAL framework', Knowledge-Based Systems, Vol. 306, No. 112710, 2024.

[3] P. Kotilainen, N. Mäkitalo, K. Systä, A. Mehraj, M. Waseem, T. Mikkonen, J. Manuel Murillo, 'Allocating distributed AI/ML applications to cloud–edge continuum based on privacy, regulatory, and ethical constraints', Journal of Systems and Software, Vol. 222, No. 112333, 2025.

[4] H. Kumar Apat, B. Sahoo, 'LESP:A fault-aware internet of things service placement in fog computing', Sustainable Computing: Informatics and Systems, Vol. 46, No. 101097, 2025,
[5] P. Alho, J. Mattila, 'Service-oriented approach to fault tolerance in CPSs', Journal of Systems and Software, Vol. 105, pp. 1–17, 2015.
[6] Y. Wang, P. Castillejo, J. Martínez-Ortega, V. Hernández Díaz, 'A survey on Identity and Access Management for future IoT services', Computer Networks, Vol. 272, No. 111718, 2025.
[7] L. Zhou, W. Yau, Y. Gan, S. Liong, 'E-WebGuard: Enhanced neural architectures for precision web attack detection', Computers & Security, Vol. 148, No. 104127, 2025.
[8] K. Xu, B. Zhang, J. Li, H. He, R. Ren, J. Ren, 'DRacv: Detecting and auto-repairing vulnerabilities in role-based access control in web application', Journal of Network and Computer Applications, Vol. 240, No. 104191, 2025.
[9] N. Prabakaran, A. Anbarasi, N. Deepa, P. Pandiaraja, 'Enabling an On-demand Access to Community Sentiments using LSTM RNNs Web Service Architecture', Procedia Computer Science, Vol. 230, pp. 584–597, 2023.
[10] J. Li, 'Research On Optimization Model of High Availability and Flexibility of Blockchain System Based on Microservice Architecture', Procedia Computer Science, Vol. 261, pp. 207–216, 2025.
[11] N. Alshuqayran, N. Ali, R. Evans, 'A model-driven architecture approach for recovering microservice architectures: Defining and evaluating MiSAR', Information and Software Technology, Vol. 186, No. 107808, 2025.
[12] S. Arora, A. Ksentini, 'Christian Bonnet, Cloud native Lightweight Slice Orchestration (CLiSO) framework', Computer Communications, Vol. 213, pp. 1–12, 2024.
[13] Muddesar Iqbal, Sohail Sarwar, 'Muhammad Safyan, Moustafa Nasralla, Personalized and adaptive e-learning systems for semantic Web: a systematic review and roadmap', International Journal of Web Information Systems, Vol. 21, No. 4, pp. 327–352, 2024.
[14] K. Maheswari, C. Siva, G. Nalinipriya, 'Optimal cluster based feature selection for intrusion detection system in web and cloud computing environment using hybrid teacher learning optimization enables deep recurrent neural network', Computer Communications, Vol. 202, pp. 145–153, 2023.

[15] A. Sezgin, A. Boyacı, 'DecoyPot: A large language model-driven web API honeypot for realistic attacker engagement', Computers & Security, Vol. 154, No. 104458, 2025.
[16] J. Dogani, F. Khunjush, 'Proactive auto-scaling technique for web applications in container-based edge computing using federated learning model', Journal of Parallel and Distributed Computing, Vol. 187, No. 104837, 2024.
[17] Y. Jiang, M. Li, W. Wu, X. Wu, X. Zhang, X. Huang, R. Zhong, G. George, 'Multi-domain ubiquitous digital twin model for information management of complex infrastructure systems', Advanced Engineering Informatics, Vol. 56, No. 101951, 2023.
[18] R. Girau, M. Anedda, R. Presta, S. Corpino, P. Ruiu, M. Fadda, C. Lam, D. Giusto, 'Definition and implementation of the Cloud Infrastructure for the integration of the Human Digital Twin in the Social Internet of Things', Computer Networks, Vol. 251, No. 110632, 2024.

Biography

Xiaoyong Su received his bachelor's degree in science from Henan University in 2005, educational technology major, and his master's degree in science from Jiangxi Normal University in 2010. He is currently working as a network engineer and an information technology teacher at the Information Technology Center of the Foreign Languages College, Wuhan Textile University. His research areas include computer network technology, online education, and educational informationization.