



Research on Elastic Service Scaling Methods for Generative AI Applications Under Cloud-native Architectures

Hongyan Qu

King Graduate School, Monroe University, New Rochelle, NY 10801, USA.

How to cite this paper: Hongyan Qu. (2025) Research on Elastic Service Scaling Methods for Generative AI Applications Under Cloud-native Architectures. *Advances in Computer and Communication*, 6(4), 224-229.
DOI: 10.26855/acc.2025.10.012

Received: August 31, 2025
Accepted: September 30, 2025
Published: October 28, 2025

***Corresponding author:** Hongyan Qu, King Graduate School, Monroe University, New Rochelle, NY 10801, USA.

Abstract

This study focuses on the operational characteristics of generative AI applications under cloud-native architectures, analyzing their computational intensity and fluctuating load patterns, and exploring the adaptability and necessity of elastic service scaling. By reviewing the theoretical foundations, the research further investigates key techniques such as auto-scaling and resource scheduling, constructing an elastic scaling framework suited for cloud-native environments. Combined with load-aware dynamic scaling methods and incorporating containerization, microservices, and service mesh mechanisms, feasible strategic pathways are proposed. Experiments conducted in a representative environment evaluate and compare different scaling schemes, showing that the proposed strategies enhance resource utilization and system stability. The findings indicate that integrating cloud-native architectures with generative AI applications offers significant advantages and practical value in elastic service scaling.

Keywords

Cloud-native architecture; Generative AI; Elastic services; Auto-scaling

Generative AI applications demonstrate unprecedented potential in language processing, image generation, and multimodal interaction, with extremely high demands on computational resources and system responsiveness. Traditional service scaling approaches struggle to accommodate their highly variable loads and intensive computational requirements, often resulting in resource wastage or performance bottlenecks. Cloud-native architecture, centered on containerization, microservices, and service mesh, provides a new technical pathway for efficient management and flexible allocation of computational resources. Against this backdrop, exploring elastic service scaling methods suitable for generative AI applications can improve resource utilization, enhance system stability and sustainability, and facilitate the deployment and optimization of these applications in production environments.

1. Generative AI Applications and Cloud-native Architecture Overview

Generative AI applications leverage deep learning models and large-scale data training to demonstrate strong creative capabilities in text generation, image synthesis, and speech modeling. During operation, these applications typically exhibit high computational intensity, significant load fluctuations, and heavy resource consumption. The enormous model parameters and complex inference processes impose stringent requirements on computing power and storage, while also demanding high real-time performance and stability. Unlike traditional intelligent applications, generative AI displays pronounced interactivity and diversity, increasing the reliance on platform flexibility and elasticity.

Cloud-native architecture provides a pathway to meet these demands, emphasizing cloud-oriented design, lightweight operation through containerization, system decomposition via microservices, and communication and governance through service mesh. This architecture enhances portability and scalability, enabling rapid resource allocation and dynamic scaling according to business needs. The integration of generative AI with cloud-native architectures allows for more efficient responses to sudden computational surges or traffic spikes, improving resource utilization, maintaining system stability, and facilitating large-scale deployment.

2. Theoretical Foundations and Key Technologies of Elastic Service Scaling

2.1 Theoretical Foundations of Elastic Service Scaling

The theoretical foundation of elastic service scaling arises from distributed systems' pursuit of dynamic adaptability, focusing on automatically adjusting resources according to load fluctuations to maintain a balance between high concurrency and low demand. Generative AI training and inference often require hundreds of GBs of GPU memory and high network bandwidth, while maintaining the same resources under low load would lead to inefficiency [1]. Research suggests that scaling mechanisms should trigger when CPU utilization exceeds a certain threshold or GPU usage remains high, and scale down when load decreases to maintain energy efficiency. Elastic service theory considers not only performance but also availability, compatibility, and security, for example, maintaining stable thermal conditions during node expansion to avoid hardware reliability issues.

2.2 Auto-scaling Technologies for Elastic Service Expansion

Auto-scaling technologies dynamically support generative AI applications by continuously monitoring system metrics and executing scaling actions. Common metrics include CPU utilization, GPU memory usage, memory bandwidth, and request latency. When system load or latency exceeds defined thresholds, scaling is triggered to maintain service quality. The key parameters and performance indicators of auto-scaling technologies are summarized in Table 1. In practice, commonly used approaches include rule-based threshold control and prediction-based scheduling algorithms. Rule-based control is suitable for relatively stable loads, while prediction-based methods leverage historical data and forecasting models to provision resources in advance of traffic spikes, ensuring smooth scaling operations and system stability.

Table 1. Key Parameters and Performance Metrics of Auto-Scaling Technology

Metrics	Parameters/Thresholds	Performance Metrics
CPU Utilization Trigger Threshold	75%-80%	Scaling Response Time <10 seconds
GPU Memory Usage Trigger Threshold	85%-90%	Scaling Completion Time 8-12 seconds
Memory Bandwidth Monitoring Range	60-95%	Avoid Memory Bottlenecks
Request Latency Trigger Threshold	>200ms	Average Latency Control <250 ms
Scaling Type	Threshold Rule / Prediction Model	Smooth Resource Transition to Prevent Fluctuations
Multi-Tenant Security	Data Isolation / Access Control	No Cross-Task Interference During Scaling
Scaling Frequency	0-6 times per minute	System Load Stability and Energy Efficiency Balance

During application, auto-scaling must not only meet performance requirements but also address security and compatibility. In multi-tenant environments, it is essential to ensure data isolation and access control between different tasks, while preventing cross-task interference or resource conflicts during scaling operations. Additionally, auto-scaling can be integrated with energy management, automatically reducing redundant containers or nodes under low load to save energy, and leveraging logging and monitoring strategies to quickly respond to anomalies, ensuring long-term, efficient, and stable system operation.

2.3 Resource Scheduling and Optimization Technologies for Elastic Service Scaling

Resource scheduling and optimization technologies focus on the efficient allocation of computing, storage, and network resources across multiple nodes and clusters. Generative AI applications often handle continuous data streams ranging from 10 to 100 kbps during inference, and training may involve the transfer of hundreds of terabytes of parameters, placing high demands on network bandwidth and storage I/O performance [2]. To avoid performance bottlenecks at hotspot nodes, scheduling systems typically distribute tasks across multiple nodes using load-balancing strategies, while employing GPU passthrough and RDMA technologies to reduce latency, ensuring that critical data is transmitted within 0.5 ms.

Optimization techniques also include energy efficiency management and reliability assurance. For example, scheduling systems must maintain a stable power supply within a current fluctuation range of 100-5000 A to prevent node failures. In data center environments with temperatures ranging from -20°C to 50°C, dynamic cooling strategies are employed to ensure long-term hardware operation. Scheduling systems can also leverage container priority and resource reservation policies to dynamically adjust task allocation order and bandwidth assignment, enhancing overall utilization, reducing latency fluctuations, and achieving a comprehensive balance of performance, stability, and hardware lifespan [3].

3. Design and Implementation of Elastic Scaling for Generative AI under Cloud-Native Architecture

3.1 Architecture Design of the Elastic Service Scaling Framework

Constructing an elastic service scaling framework for generative AI in a cloud-native environment requires tight integration of resource management, load monitoring, and policy execution. The architecture typically consists of monitoring, scheduling, and execution modules. The monitoring module collects CPU, GPU, and network status at a sampling frequency of 100-500 Hz. The scheduling module calculates and generates scaling strategies based on these metrics, while the execution module allocates resources through container orchestration tools within 2-3 seconds [4]. The framework must ensure high availability; if a single node fails within a current fluctuation range of 100-5000 A, the system can automatically switch within 10-12 seconds to maintain overall service stability. This architecture balances performance, reliability, and security throughout the scaling process.

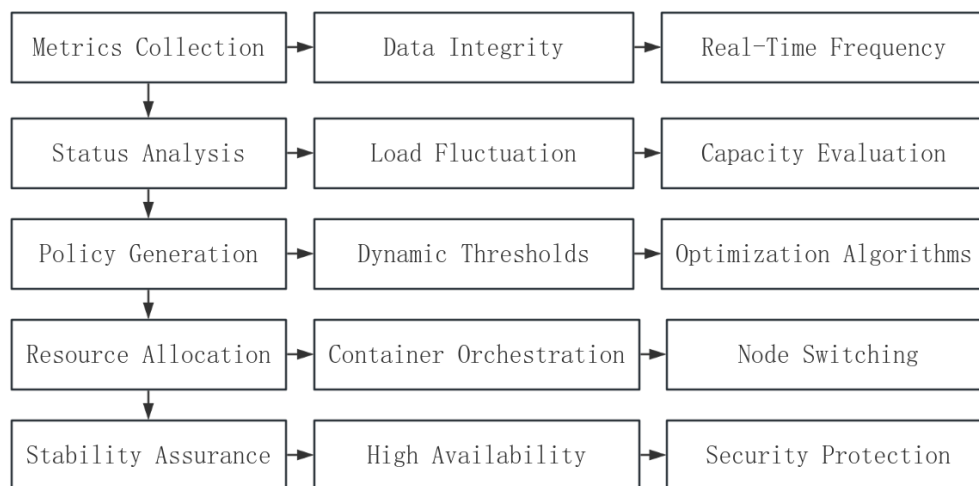


Figure 1. Elastic Scaling Framework Workflow.

3.2 Load-aware Dynamic Scaling Method

The key to dynamic scaling lies in real-time sensing of load changes and rapid response. During peak inference, GPU utilization in generative AI can exceed 80%, and latency may reach over 250 ms, requiring the system to complete

scaling actions within 10 seconds to restore performance. When load fluctuates below normal levels during off-peak hours, the number of containers automatically shrinks to reduce energy consumption and maintenance costs. Dynamic scaling not only considers computational resources but also bandwidth and storage I/O; for example, when continuous input reaches 10-20 kbps, network latency must remain within the threshold corresponding to the 0.5 mm/s transmission rate [5].

To quantify scaling decisions, a load index L can be defined as:

$$L = \alpha \cdot U_{GPU} + \beta \cdot U_{CPU} + \gamma \cdot D_{net}$$

where U_{GPU} is GPU utilization, U_{CPU} is CPU utilization, D_{net} is network latency, and α, β, γ are weighting coefficients satisfying $\alpha + \beta + \gamma = 1$. Scaling is triggered when $L > L_{thresh}$ and scaling down occurs when $L < L_{thresh_low}$. Through this load-aware mechanism, the system maintains smooth transitions between different operating states, improving efficiency while ensuring availability.

3.3 Containerization and Microservices in Elastic Scaling

Containerization and microservices are critical supports for cloud-native elastic scaling. Containerization enables rapid deployment and migration of model inference environments, maintaining stable operation even within data center temperature ranges of -20°C to 50°C , while also supporting version management and rollback through automated scripts. Microservices decompose training, inference, and data preprocessing into independent modules, allowing each module to scale independently and be flexibly scheduled across different nodes. For example, in a voice generation service, when input requests surge at remote nodes, the audio processing module can scale independently without affecting the stability of the text generation module [6]. Containers maintain long-term operation under a cooling precision of $\pm 0.3^{\circ}\text{C}$, ensuring compatibility and security among microservices, while unified logging and monitoring mechanisms facilitate efficient cross-module collaboration and fault isolation.

3.4 Service Mesh and Resource Scheduling Optimization Strategies

Service meshes handle communication and traffic governance in elastic scaling, using a unified control plane to implement cross-node resource scheduling and service discovery, while supporting dynamic routing and load balancing strategies. When traffic surges to several thousand requests per second, the mesh can complete traffic distribution in a very short time, preventing overload on individual nodes and ensuring reliable request delivery [7]. Resource scheduling optimization also includes energy efficiency and security management; for instance, in adequately cooled environments, node heat dissipation remains stable to prevent performance degradation due to overheating. During task scheduling, the system ensures isolation and compatibility in multi-tenant scenarios, maintaining stable service quality for generative AI under high loads. Furthermore, the service mesh continuously adjusts resource allocation strategies through real-time monitoring and scheduling feedback, optimizing performance, energy efficiency, and reliability for large-scale generative AI applications [8].

4. Experimental Analysis and Performance Evaluation

4.1 Experimental Environment and Dataset Design

To ensure objective evaluation of elastic scaling strategies, experiments were conducted under high-spec hardware and strict environmental conditions, guaranteeing reliable data and results. The experimental cluster consisted of 32 nodes, each equipped with a 64-core CPU, 512 GB memory, and 8 GPUs with 48 GB VRAM, interconnected via 100 Gbps bandwidth. Monitoring employed a sampling frequency of 100-500 Hz to track CPU, GPU, and network states, ensuring precise scaling triggers. The workload included hundreds of terabytes of text and image generation tasks, with inference request rates maintained at 5000-7000 QPS and batch sizes ranging from 64 to 256. Operating conditions kept the ambient temperature at $20^{\circ}\text{C} \pm 0.3^{\circ}\text{C}$, with airflow of 10-15 m/s to cool the hardware and maintain long-term stability.

Under these conditions, three representative elastic scaling methods were selected to compare performance in terms of trigger timing, resource scheduling, and system stability:

Threshold-based method: Uses CPU or GPU utilization reaching 75-80% as a scaling signal; simple to implement, but may experience 2-3 minutes of response lag during traffic spikes.

Load prediction method: Combines 10-20 minutes of historical load curves with real-time monitoring data to schedule resources in advance, enabling scaling before peak load reaches 90%, improving responsiveness and stability.

Comprehensive optimization method: Builds on the prediction method by incorporating service mesh and RDMA direct scheduling, reducing cross-node communication latency to below 0.5 ms, while optimizing resource allocation to achieve balanced performance and energy efficiency under high concurrency.

4.2 Performance Evaluation Metrics for Elastic Scaling Strategies

The evaluation framework considers response latency, resource utilization, and system stability. Latency metrics require an average response time below 200 ms and a peak latency under 500 ms under high concurrency. Resource utilization focuses on GPU efficiency within 70-80%; exceeding 90% may lead to overload. Stability is measured by node recovery time and task loss rate, with targets of recovery within 12 seconds and task loss under 0.5% under current fluctuations of 100-5000 A. Compatibility and scalability are also included; container-to-container communication latency must remain below 0.5 ms. These metrics provide a unified reference for comparing the three strategies under the same conditions.

4.3 Comparative Evaluation of Different Strategies

To comprehensively assess the performance of different elastic scaling strategies in generative AI applications, the experiment systematically compared the three methods based on key metrics such as response speed, resource utilization, and communication latency. The results are summarized in Table 2.

Table 2. Comparison of Performance for Different Elastic Scaling Strategies

Metrics	Threshold-Based Method	Load Prediction Method	Comprehensive Optimization Method
Scaling Trigger Delay	2-3 minutes	<10 seconds	<10 seconds
Peak Latency (ms)	450-500	200-250	<200
GPU Utilization (%)	85-95	70-80	70-80
Node Recovery Time (s)	15-18	10-12	10-12
Data Transmission Latency (ms)	2-3	0.5-1	0.5
Energy Efficiency Ratio	Medium	High	High

From the data, the threshold-based method exhibits significant scaling delay, resulting in higher latency during peak periods, GPU overload, longer node recovery times, and the highest data transmission delays. The load prediction method can schedule resources in advance, reducing peak latency to 200-250 ms, maintaining GPU utilization within a reasonable range, and shortening node recovery time. The comprehensive optimization method further refines communication paths and scheduling strategies on top of prediction, stabilizing data transmission latency within 0.5 ms, achieving the fastest system response, highest energy efficiency, and the most balanced overall performance, demonstrating optimal adaptability under complex, high-load conditions.

5. Conclusion

Generative AI applications demonstrate excellent elastic scaling capability under cloud-native architectures. Future development will increasingly rely on intelligent and automated scheduling mechanisms, leveraging higher-precision prediction models and cross-platform resource coordination to achieve millisecond-level response and stability assurance. At the hardware level, deep integration of heterogeneous computing and low-latency interconnects will further enhance system efficiency; at the software level, continuous optimization of service mesh and microservices will

promote more precise scaling strategies. As data volumes continue to grow, green computing and energy efficiency optimization will become critical considerations, with sustainable operation under resource-constrained conditions determining the adoption rate and industrial value of these applications.

References

- [1] Kotama DNI, Funabiki N, Panduman FYY, et al. Implementation of Sensor Input Setup Assistance Service Using Generative AI for SEMAR IoT Application Server Platform. *Information*. 2025;16(2):108.
- [2] Panduman FYY, Husna R, Noprianto, et al. An Application of SEMAR IoT Application Server Platform to Drone-Based Wall Inspection System Using AI Model. *Information*. 2025;16(2):91.
- [3] MinIO and F5 to Enhance AI Workloads with High-Performance Object Storage and Distributed Application Services. *M2 Press-wire*. 2024.
- [4] Skandali D, Magoutas A, Tsourvakas G. Consumer Behaviour on AI Applications for Services: Measuring the Impact of Value-Based Adoption Model on Luxurious AI Resorts' Applications. *Rev Mark Sci*. 2024;22(1):57-85.
- [5] Kumar Y, Lin M, Paredes C, et al. A Comprehensive Review of AI Advancement Using testFAILS and testFAILS-2 for the Pursuit of AGI. *Electronics*. 2024;13(24):4991.
- [6] Sela L, Sowby BR, Salomons E, et al. Making waves: The potential of generative AI in water utility operations. *Water Res*. 2025;272:122935.
- [7] Elastic Releases Its Elastic Cloud Serverless Powered by Search AI Lake. *Manuf Close-Up*. 2024.
- [8] Plevris V, Papazafeiropoulos G. AI in Structural Health Monitoring for Infrastructure Maintenance and Safety. *Infrastructures*. 2024;9(12):225.