Hybrid Auto-Scaling of Microservices Based on Load Prediction (Postprint)

Authors: Song Chenghao, Jiang Lingyun

Date: 2022-04-07T15:01:57+00:00

Abstract

Given that edge clouds lack more powerful computational processing capabilities compared to central clouds, they are susceptible to unnecessary scaling jitter or insufficient resource processing capability when handling dynamic workloads. This paper therefore conducts experimental evaluations on microservice applications using two synthetic and two real-world workloads in a real edge cloud environment, and proposes a hybrid autoscaling method based on workload prediction (Predictively Horizontal and Vertical Pod Autoscaling, Pre-HVPA). The method first employs machine learning to predict workload data features and obtain final workload prediction results, which are then utilized for hybrid horizontal and vertical autoscaling. Simulation results demonstrate that autoscaling based on this approach can reduce scaling jitter and container usage, making it suitable for microservice applications in edge cloud environments.

Full Text

Preamble

Vol. 39 No. 8
Application Research of Computers
ChinaXiv Partner Journal

Hybrid Autoscaling of Microservices Based on Workload Prediction

Song Chenghao, Jiang Lingyun†
(College of Telecommunications & Information Engineering, Nanjing University of Posts & Telecommunications, Nanjing 210003, China)

Abstract: Since edge clouds lack the powerful computing capabilities of central clouds, they are prone to meaningless scaling jitter or insufficient resource processing capacity when handling dynamic workloads. This paper experimentally evaluates microservice applications in a real edge cloud environment using

two synthetic and two real-world workloads, and proposes a hybrid autoscaling method based on workload prediction (Predictively Horizontal and Vertical Pod Autoscaling, Pre-HVPA). The method first employs machine learning to predict workload data characteristics and obtains final workload predictions. It then leverages these predictions for hybrid horizontal and vertical autoscaling. Simulation results demonstrate that this approach reduces scaling jitter and container usage, making it suitable for microservice applications in edge cloud environments.

Keywords: edge computing; microservices; workload prediction; hybrid autoscaling

0 Introduction

Over the past few years, the convergence of cloud computing and the Internet of Things has positioned Multi-access Edge Computing (MEC) and cloud-native autoscaling technologies for microservice architectures as key research directions. Edge cloud computing provides micro data center (MDC) facilities closer to users and applications, helping to overcome latency issues. Microservices represent an emerging architectural paradigm that decomposes applications into multiple fine-grained distributed components that interact to serve complex business scenarios. Today, many industrial applications, particularly edge cloud and IoT-based services, adopt microservice architectures. While edge cloud platforms benefit from microservice architectures, they also face significant challenges, most notably their limited resource processing capacity compared to central clouds that use microservice architectures. Resource request processing is typically handled through microservice autoscaling, making effective application autoscaling under dynamically changing workloads a primary challenge for reducing edge cloud resource consumption.

Currently, numerous tools support microservice autoscaling, with Kubernetes (k8s) being the most popular container orchestration platform. Kubernetes enables adaptation through Horizontal Pod Autoscaling (HPA) and Vertical Pod Autoscaling (VPA). HPA adjusts the number of application instances (e.g., Pods—the smallest scheduling unit in k8s), while VPA modifies the computing resources allocated to each instance. However, most existing solutions consider only horizontal or vertical scaling in isolation, both of which require substantial resources. Horizontal autoscaling, in particular, achieves high availability by replicating microservices across machines, but this greedy approach consumes significant resources during scaling and is only suitable for hardware-abundant environments—not the resource-constrained, fine-grained edge cloud platforms.

Furthermore, Kubernetes' s container-level autoscaling employs threshold-based policies that generate considerable scaling jitter under dynamic workloads, incurring additional overhead. Although recent research has improved upon Kubernetes' s strategy using workload prediction methods, achieving performance

gains over threshold-based approaches, these solutions remain unsuitable for low-capacity edge clouds and fail to adequately demonstrate scaling effectiveness under minor load fluctuations. Consequently, developing a fine-grained, low-resource autoscaling strategy for edge cloud platforms represents a critical research challenge.

1 Related Work

Researchers have proposed various microservice autoscaling strategies and algorithms. Threshold-based policies, as implemented in Kubernetes, remain the most popular approach. However, determining optimal thresholds is challenging, requiring extensive parameter tuning at the container level. While Kubernetes has added support for vertical autoscaling, it remains experimental at release time, with limited implementation details available.

To address Kubernetes' s limitations, some studies have proposed novel algorithms. One approach uses Q-learning to adaptively adjust thresholds without user configuration while meeting SLA targets, though it performs poorly with rapidly changing workloads. Other work employs artificial neural networks and resource optimization for cost-effective microservice autoscaling in cloud infrastructure. Window prediction methods have been applied to fog computing microservices to improve SLO response times. While ARIMA models enable short-term load prediction improvements over threshold-based methods, they rely solely on horizontal scaling—an inefficient, coarse-grained approach unsuitable for edge clouds. These improvements focus on threshold-based drawbacks without considering scaling granularity or dimensions.

Most existing strategies remain exclusively horizontal or vertical. Reinforcement learning has been used to improve QoS and response times in Kubernetes horizontal scaling, but tested with disaster management systems that generate excessive workloads unsuitable for edge applications. Another framework monitors container resource utilization for autoscaling but employs only single-dimensional scaling. Only recent work has combined both dimensions: one study achieved fine-grained hybrid autoscaling using reinforcement learning to optimize response time, while another utilized IBM' s MAPE-K principle for vertical scaling, demonstrating that fine-grained vertical scaling can reduce container usage while horizontal scaling handles workload peaks. CloudVAMP provides memory oversubscription for VMs (vertical scaling) but lacks support for diverse workload types, limiting fine-grained resource management.

Unlike prior work, this paper focuses specifically on resource consumption during microservice autoscaling at edge nodes. Building upon previous research, we improve upon predictive load forecasting by implementing hybrid autoscaling to reduce both Pod replica counts and scaling jitter.

2.1 Deployment Framework

In the edge cloud environment, we first reconfigure modules and decompose dependencies between central and edge cloud platforms to containerize cloud components. We then install Kubernetes worker nodes on both central and edge cloud nodes. After configuring infrastructure orchestration services, Kubernetes clusters, and edge nodes, we define cloud-edge connection components to enable information control between central cloud data and edge computing nodes. Figure 1 illustrates the Kubernetes deployment across central and edge clouds.

Figure 1. Edge cloud deployment framework

2.2 Kubernetes Autoscaling Scheme

Kubernetes, the most popular container solution, provides service discovery, resource scheduling, monitoring, application deployment, and autoscaling. Its microservice autoscaling solution uses threshold-based horizontal autoscaling, which suffers from two problems: difficulty in determining optimal thresholds, and coarse-grained scaling that fails to optimize individual Pod utilization. The algorithm scales when CPU or memory utilization exceeds upper thresholds or falls below lower limits, often leaving resources underutilized across multiple Pods.

Algorithm 1: Kubernetes Horizontal Autoscaling 1) Input: Total application resource requests, scaling threshold, Pod resources
2) Output: Number of scaled Pods
3) Set default tolerance $\tau = 0.1$
4) Calculate resource utilization: Utilization = Request / (NumPods × Resource per Pod)
5) If Utilization > Target + τ, then NumPods = NumPods + 1
6) If Utilization < Target - τ, then NumPods = NumPods - 1
7) Return final scaling result

This algorithm adjusts Pod replicas based on current resource utilization. When average utilization across all replicas exceeds a target percentage threshold, the system scales out; when below threshold, it scales in. Resource utilization is calculated as total requests divided by allocated resources. We collect resource request counts every 5 seconds to compute average utilization. This strategy serves as our baseline for comparison, with its architecture shown in Figure 2.

Figure 2. Kubernetes horizontal pod autoscaling schematic

2.3 Limitations of Predictive and Reactive Autoscaling

Existing literature typically focuses on predictive methods to reduce jitter from Kubernetes's reactive scaling or optimizes response time violations through modified scaling policies. For instance, the predictive algorithm we compare against (named HPA in Chapter 4) uses ARIMA for short-term prediction, inputting

resource utilization and outputting predicted scaling values through time series modeling. However, this approach only predicts incoming workload, and while reducing jitter, it leads to higher Pod usage through proactive scaling.

The reactive hybrid autoscaling algorithm we compare (named HVPA in Chapter 4) uses reinforcement learning for fine-grained microservice scaling through a 9-step transition model that introduces vertical autoscaling. While more precise, this fine-grained approach targets minimal scaling quantities, causing frequent Pod scaling operations under fluctuating loads and generating unnecessary jitter —also detrimental to edge cloud resource utilization.

Note that the Kubernetes baseline requires optimization for both jitter and Pod count, while existing predictive and reactive hybrid approaches each optimize only one metric. To improve both simultaneously, we employ machine learning to predict characteristics across synthetic and real workloads, proposing a hybrid autoscaling algorithm that combines prediction with hybrid scaling. This provides proactive scaling for rapidly changing loads to reduce jitter while prioritizing vertical scaling through min/max replica calculations to utilize fine-grained per-Pod resources more effectively, thereby reducing Pod usage during slow-changing loads and achieving overall resource reduction.

3 Predictive Hybrid Autoscaling Algorithm (Pre-HVPA)

To address excessive Pod resource consumption and scaling jitter, we propose a hybrid autoscaling method using machine learning-based workload prediction. Figure 3 illustrates our approach: first, we generate growth and periodic workloads for edge-deployed microservices while collecting CPU-intensive and I/O memory-intensive load data. Next, we partition this data to train machine learning models for workload prediction. Finally, we integrate predicted results into our edge cloud Kubernetes platform' s hybrid autoscaling module to determine required Pod quantity changes and scaling frequency.

Figure 3. Block diagram of predicting hybrid scaling method

3.1 Deploying Edge Applications

We deploy microservice applications on edge cloud nodes (see Section 4.1 experimental environment) that use Support Vector Regression (SVR)—a supervised machine learning algorithm—to predict temperature from historical sensor logs. This benchmark application simulates CPU-intensive and memory-intensive workloads, representing typical microservice use cases such as data prediction, search algorithms, digital content conversion, and data compression tasks for IoT industrial applications.

3.2 Importing Workloads

To evaluate our predictive hybrid autoscaling model, we use two synthetic and two real-world workloads. The synthetic workloads include growth and periodic

patterns to simulate conventional load variations. Growth workloads occur when most users begin accessing services, while periodic workloads are common in real applications with regular cyclical patterns.

For growth workloads, we initialize a fixed request count M(0) and increase it at a fixed growth rate q per minute, calculated as:

$$M(t) = M(0) + \Delta t \times q + r_c \tag{1}$$

where r_c (0,1) is a random factor.

For periodic workloads, we use a sinusoidal function with period λ, amplitude α, and initial bias factor β:

$$U(t) = U(0) + \alpha \times \sin\left(\frac{2\pi t}{\lambda}\right) + r_c \tag{2}$$

The real-world workloads utilize the publicly available Alibaba Cloud Tianchi 2020 hybrid cloud workload trace, including CPU-intensive and I/O memory-intensive patterns.

3.3 Machine Learning Module for Load Data Training

We train five linear regression models on the four workload datasets, comparing their Mean Squared Error (MSE) to evaluate prediction accuracy:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3}$$

where m is the sample count, y_i is the true value, and ŷ_i is the predicted value.

The compared models include Linear Regression (LR), Elastic Net (EN), Polynomial Regression (PR), XGBoost (XGB), and Decision Tree Regression (DTR). After 10 tests averaging MSE values, Bayesian Ridge (BR) demonstrates improvements of 88.5%, 85.9%, 83.8%, 63.2%, and 38.1% over LR, EN, PR, DTR, and XGB respectively. To enhance hybrid autoscaling accuracy, we select BR as our final machine learning algorithm.

Table 1. Prediction error (MSE) of dataset

Method	MSE
...	...

3.4 Hybrid Autoscaling Algorithm

The trained model and predicted workload data feed into our hybrid autoscaling module. The primary objective is maintaining high availability with low

resource consumption, enabling dynamic resource scaling at high utilization rates across edge nodes. While Kubernetes-based horizontal scaling is effective, it consumes excessive resources by replicating entire Pod instances. Vertical scaling offers better resource efficiency for specific scenarios by allocating more resources to individual Pods. Our hybrid approach differs from coarse-grained horizontal scaling by precisely determining microservice requirements, incorporating fine-grained vertical adjustments while retaining horizontal capabilities for state consistency.

Our algorithm uses different scaling metrics per workload type: CPU utilization for CPU-intensive loads and memory utilization for memory-intensive loads, quantified uniformly through resource request counts. Algorithm 2 describes the CPU-intensive case.

Algorithm 2: Hybrid Autoscaling Algorithm - Input: Predicted load resources, Pod resource requirements, target utilization
- Output: Number of scaled Pods, scaling operation count
- For observed resource usage, compare current utilization with target to determine vertical scaling feasibility
- If total allocation equals target utilization: maintain current scaling strategy with horizontal scaling up to max replicas
- If total allocation exceeds target: first vertically scale each Pod up, then horizontally scale out
- If total allocation is below target: first horizontally scale in, then vertically scale each Pod down (minimum 1 Pod)
- Record final Pod replica count

When predicted loads enter the scaling module, the algorithm first compares total allocated resources against target utilization to prioritize horizontal or vertical scaling. When resources exceed targets, vertical scaling is prioritized to increase per-Pod utilization, reducing horizontal scaling needs while meeting requests. When requests decrease, horizontal scaling occurs first to reclaim Pods, followed by vertical scaling to minimize Pod usage. Predictive inputs enable proactive scaling based on future resource requests, avoiding unnecessary scaling operations and reducing jitter for smoother Pod count transitions.

4.1 Experimental Environment

We test our hybrid autoscaling model using a Kubernetes cluster deployed in an edge cloud environment consisting of four physical machines running compatible Docker and Kubernetes versions. Detailed configurations are shown in Table 2.

Table 2. Equipment information of MEC

Configuration	Details
OS	CentOS 7.8
Memory	4 GB

Configuration	Details
CPU	i5 4-core
...	...

4.2 Load Prediction Results Analysis

We employ 10-fold cross-validation, randomly partitioning all workload data into training (90%) and test (10%) sets. This method provides optimal error estimation as demonstrated through extensive experiments.

Figure 4 presents 120-minute prediction results for four workload types. CPU-intensive and I/O memory-intensive workloads show higher request density than the synthetic loads. Growth and periodic workloads exhibit superior prediction accuracy because machine learning methods struggle with abrupt bursts in intensive workloads. Nevertheless, our approach achieves relatively good results even for these challenging cases, enabling more accurate autoscaling when better-predicted loads feed into our hybrid module.

Figure 4. Actual and forecasted load experiment evaluation

4.3 Autoscaling Results Analysis

This section compares our Pre-HVPA method against Kubernetes threshold-based horizontal autoscaling, prediction-based horizontal autoscaling (HPA), and threshold-based reactive hybrid autoscaling (HVPA) across incremental, periodic, CPU-intensive, and memory-intensive workloads.

4.3.1 CPU-Intensive Load Results Kubernetes baseline uses threshold-based horizontal autoscaling (reactive). When any Pod' s resource utilization exceeds the custom scaling threshold (typically 75%), the system adds Pod replicas; when all Pods fall below the reduction threshold (typically 50%), it removes replicas. While this restores application performance, Figure 5(a) shows that for rapidly changing CPU-intensive workloads, it generates excessive scaling jitter and Pod usage overhead.

Our Pre-HVPA method predicts CPU-intensive load characteristics, better adapting to dynamic changes through fine-grained hybrid autoscaling. Figure 5(b) shows that prediction-based HPA improves jitter but increases Pod count. Our approach reduces both jitter and Pod usage, particularly during 20-50 and 70-80 minute intervals where it uses fewer Pods than horizontal scaling alone.

Figure 5(c) compares against reactive HVPA, which minimizes scaling quantity without considering jitter. Our predictive hybrid approach uses similar Pod counts during 0-40 minutes while reducing 2 meaningless scaling operations, and proactively scales before the 60-minute load spike, reducing 2 unnecessary expansions while using minimal Pods for smoother, more stable scaling.

Figure 5. CPU intensive load autoscaling results

4.3.2 Memory-Intensive Load Results Memory-intensive workloads have fewer requests with slower changes, making them another representative test case. Figure 6(a) shows Kubernetes produces less jitter and fewer Pods for memory-intensive loads compared to CPU-intensive cases. Nevertheless, our predictive hybrid strategy demonstrates clear advantages, particularly during 80-100 minutes, where proactive prediction reduces 5 meaningless scaling jitters and fine-grained scaling decreases Pod count. Detailed Pod usage is evaluated in Section 4.4.

Figures 6(b) and 6(c) show comparisons with HPA and HVPA, demonstrating that our strategy completes proactive scaling around 60 minutes while using fewer Pods overall.

Figure 6. Memory intensive load autoscaling results

4.3.3 Periodic Load Results Periodic loads'regular patterns enable more accurate prediction and precise resource calculation, facilitating proactive hybrid autoscaling deployment with fewer resources. Figure 7(a) shows that compared to Kubernetes, our strategy enables earlier resource reclamation during 10-30 minutes and uses relatively more Pods during 40-60 minutes for proactive scaling that reduces jitter.

Figures 7(b) and 7(c) compare against prediction-only horizontal and hybrid approaches, showing our method uses fewer Pods than pure prediction-based scaling and generates less jitter than pure hybrid scaling. Growth workload results follow similar patterns and are omitted for brevity.

Figure 7. Periodic load autoscaling results

4.4 System Performance Evaluation

This section evaluates scaling frequency and Pod usage across four workload types. Figure 8 shows that Pre-HVPA generates less scaling jitter than Kubernetes, HPA, and HVPA for all workloads: 40% and 20% improvement over Kubernetes and HVPA for growth workloads; 8.5% over HVPA for periodic; 37.2%, 18.2%, and 32.5% over the three baselines for CPU-intensive; and 18.8%, 7.1%, and 25.7% for memory-intensive.

Figure 9 measures Pod replica usage over 120 minutes, showing clear improvements: 9.4% and 8.4% over Kubernetes and HPA for growth; 9.6%, 11.4%, and 2.1% over all three for periodic; 9.5% and 8.8% over Kubernetes and HPA for CPU-intensive; and 12.4% and 14.7% for memory-intensive.

For growth workloads, our strategy trades 1 additional scaling operation for 9.4% fewer Pods. For periodic loads, it matches Kubernetes' s jitter while reducing Pods by 9.6%. For CPU-intensive workloads, it sacrifices 2.1% more Pods to

eliminate 13 unnecessary scaling operations. For memory-intensive loads, it trades 2.5% more Pods to reduce 9 meaningless jitters.

Overall, our predictive hybrid autoscaling strategy for edge cloud environments achieves superior results in both jitter reduction and Pod usage optimization.

Figure 8. Comparison of scaled times of four schemes

Figure 9. Comparison of number of scaled pod replicas of four schemes

5 Conclusion

Existing microservice autoscaling solutions primarily target central clouds. Achieving resource-efficient autoscaling for edge cloud microservice containers presents significant challenges. This paper proposes a novel algorithm that determines more cost-effective autoscaling strategies for industrial microservice applications deployed at the edge. Compared to current popular solutions, our approach optimizes both scaling frequency and Pod replica count, demonstrating superior performance.

Our method employs supervised machine learning with minimal MSE to predict diverse workload types, feeding predictions into a custom hybrid autoscaling module. Evaluations using two intensive and two synthetic workloads show overall optimization for both metrics. This research represents an initial achievement; future work will test the model with different applications and investigate reducing scaling response time to further decrease user costs.

References

[1] Taibi D, Lenarduzzi V, Pahl C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation [J]. IEEE Cloud Computing, 2017, 4(5): 22-32.

[2] Pan J, Mcelhannon J. Future Edge Cloud and Edge Computing for Internet of Things Applications [J]. IEEE Internet of Things Journal, 2018, 5(1): 439-449.

[3] Horovitz S, Arian Y. Efficient Cloud Auto-Scaling with SLA Objective Using Q-Learning [C]// Proc of IEEE the 6th International Conference on Future Internet of Things and Cloud. Barcelona: IEEE Press, 2018: 130-137.

[4] Dhuraibi Y, Paraiso F, Djarallah N, et al. Autonomic vertical elasticity of Docker containers with ELASTICDOCKER [C]// Proc of IEEE the 10th International Conference on Cloud Computing. Honolulu, HI: IEEE Press, 2017: 472-479.

[5] Wilder B. Cloud Architecture Patterns: Using Microsoft Azure [J]. Oreilly & Assoc Inc, 2012.

[6] Lin K, Altaf U, Jayaputera G, et al. Auto-Scaling a Defence Application across the Cloud Using Docker and Kubernetes [C]// IEEE/ACM International Conference on Utility and Cloud Computing Companion. Zurich: IEEE Press, 2018: 327-334.

[7] Shan Pengrong, Yang Meihong, Zhao Zhigang, et al. Design and Implementation of Elastic Scaling Scheme Based on Kubernetes Cloud Platform [J]. Journal of Computer Engineering, 2021, 47(1): 312-320.

[8] Nitto E, Florio L, Tamburri D. Autonomic decentralized microservices: The Gruapproach and its evaluation [M]// In Microservices: Science and Engineering. Switzerland: Springer Nature, 2020: 209-248.

[9] Prachitmutita I, Aittinonmongkol W, Pojjanasuksakul N, et al. Auto-scaling microservices on IaaS under SLA with cost-effective framework [C]// Proc of the Tenth International Conference on Advanced Computational Intelligence. Xiamen: IEEE, 2018: 583-588.

[10] Nardelli M, Hochreiner C, Schulte S. Elastic Provisioning of Virtual Machines for Container Deployment [C]// Proc of ACM/SPEC International Conference on Performance Engineering. New York: ACM/SPEC, 2017: 5-10.

[11] Khaleq A, Ra I. Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications [J]. IEEE Access, 2021, PP(99): 1-13.

[12] Lombardi F, Muti A, Aniello L, et al. PASCAL: An architecture for proactive auto-scaling of distributed services [J]. Future Generation Computer Systems, 2019, PP(98): 342-361.

[13] Ma Xiaolin. A Container Cloud Elastic Scaling Strategy Based on Load Characteristics Prediction [J]. Journal of Information Security Research, 2019, 5(42): 236-241.

[14] Rossi F, Nardelli M, Cardellini V. Horizontal and Vertical Scaling of Container-Based Applications Using Reinforcement Learning [C]// Proc of IEEE the 12th International Conference on Cloud Computing. Milan: IEEE, 2019: 329-338.

[15] Xuxin Tang, Fan Zhang, Xiu Li, et al. Quantifying cloud elasticity with container-based autoscaling [J]. Future Generation Computer Systems, 2019, 98: 672–681.

[16] Sembiring K, Beyer A. Dynamic resource allocation for cloud-based media processing [C]// Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. New York: ACM Press, 2013: 49-54.

[17] Germán M, Miguel C, Carlos A. Automatic memory-based vertical elasticity and oversubscription on cloud platforms [J]. Future Generation Computer Systems, 2016, 56(1): 1-10.

[18] Olive D J, Linear Regression [M]. Berlin: Springer, 2017.

[19] Zhang Z, Lai Z, Xu Y, Shao L, et al. Discriminative Elastic-Net Regularized Linear Regression [J]. IEEE Transactions on Image Processing, 2017, 26(3): 1466-1481.

[20] Igual L, Seguí S. Introduction to Data Science: Regression Analysis [M]. Switzerland: Springer, 2017: 97-114.

[21] Chen Tianqi, Guestrin C. XGBoost: A Scalable Tree Boosting System [C]// ACM Knowledge Discovery and Data Mining. New York: ACM Press, 2016: 785-794.

[22] Rathore S, Kumar S. A Decision Tree Regression based Approach for the Number of Software Faults Prediction [J]. ACM SIGSOFT Software Engineering Notes, 2016, 41(1): 1-6.

[23] Ali E A, Seleznjev O, Sjstedt-de S, et al. Measuring Cloud Workload Burstiness [C]// IEEE/ACM International Conference on Utility & Cloud Computing. London: IEEE Press, 2014: 566-572.

[24] Ilyushkin A, AHMEDALI E, Herbst N, et al. An Experimental Performance Evaluation of Autoscalers for Complex Workflows [J]. ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 2018, 3(2): 1-32.

Note: Figure translations are in progress. See original paper for figures.