TempoScale: A Cloud Workloads Prediction Approach Integrating Short-Term and Long-Term Information

Linfeng Wen[1,2], Minxian Xu[1], Adel N. Toosi[3], Kejiang Ye[1]
1. Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
2. University of Chinese Academy of Sciences, China
3. Faculty of Information Technology, Monash University, Australia
{lf.wen, mx.xu}@siat.ac.cn, adel.n.toosi@monash.edu, kj.ye@siat.ac.cn

***Abstract*—Cloud native solutions are widely applied in various fields, placing higher demands on the efficient management and utilization of resource platforms. To achieve the efficiency, load forecasting and elastic scaling have become crucial technologies for dynamically adjusting cloud resources to meet user demands and minimizing resource waste. However, existing prediction-based methods lack comprehensive analysis and integration of load characteristics across different time scales. For instance, long-term trend analysis helps reveal long-term changes in load and resource demand, thereby supporting proactive resource allocation over longer periods, while short-term volatility analysis can examine short-term fluctuations in load and resource demand, providing support for real-time scheduling and rapid response. In response to this, our research introduces TempoScale, which aims to enhance the comprehensive understanding of temporal variations in cloud workloads, enabling more intelligent and adaptive decision-making for elastic scaling.**

TempoScale utilizes the Complete Ensemble Empirical Mode Decomposition with Adaptive Noise algorithm to decompose time-series load data into multiple Intrinsic Mode Functions (IMF) and a Residual Component (RC). First, we integrate the IMF, which represents both long-term trends and short-term fluctuations, into the time series prediction model to obtain intermediate results. Then, these intermediate results, along with the RC, are transferred into a fully connected layer to obtain the final result. Finally, this result is fed into the resource management system based on Kubernetes for resource scaling. Our proposed approach can reduce the Mean Square Error by 5.80% to 30.43% compared to the baselines, and reduce the average response time by 5.58% to 31.15%. The results demonstrate the effectiveness of our proposed method in reducing violations of service-level objectives and providing better performance in terms of resource utilization.

***Index Terms*—cloud native, load prediction, auto-scaling, deep learning, mode decomposition, transformer**

I. Introduction

With the rise of cloud native and microservices architecture, containerization technology has emerged as a crucial innovation, profoundly altering the landscape of software development and deployment [1]. Among these technologies, Kubernetes (K8s), as an open-source container orchestration platform, has provided a robust framework for automating the deployment, scaling, and operation of application containers, thereby significantly enhancing efficiency [2]. Nowadays, K8s has been widely adopted by mainstream companies, including Amazon, Google, and Microsoft.

However, as more and more enterprises adopt containerized microservices architectures, and application scenarios continue to evolve, limitations of the reactive strategy employed by the default resource scheduler in K8s have become apparent [3]. For instance, under highly variable workloads, its applicability is limited, leading to resource waste and a decrease in service quality. Consequently, some enterprises are gradually shifting towards predictive scaling methods. Predictive scaling not only focuses on current loads but also involves analyzing historical data, trends, and predictive models to forecast future loads and make adjustments based on this analysis. The advantage of this approach lies in proactively allocating resources, avoiding the need for reactive measures when loads increase suddenly. Currently, elastic scaling based on load forecasting has become a crucial technology for effectively adjusting cloud resources in dynamic environments to meet user needs and minimize resource waste.

Applying a prediction-based strategy in a production environment still faces several challenges: Firstly, the dynamics and uncertainty of system and network workloads render traditional static models inadequate when confronted with complex and rapidly changing workloads [4], [5]. Additionally, the diversity and heterogeneity of real-world workloads further complicate predictions, making it difficult for a singular approach to adapt to various scenarios and environments [6]. Lastly, existing methods lack the ability to extract features across different time sequences, resulting in a lack of comprehensive understanding of the workloads. The inherent dynamism of these clusters, the variability of workloads, and the inherent limitations of the method itself, presents challenges that demand innovative solutions [7]. Therefore, this research attempts to address these challenges by exploring and extracting features such as CPU utilization at different time

This work is supported by the National Natural Science Foundation of China (No. 62102408), Shenzhen Industrial Application Projects of undertaking the National key R & D Program of China (No. CJGJZD20210408091600002), Shenzhen Science and Technology Program (No. RCBS20210609104609044), and Chinese Academy of Sciences President's International Fellowship Initiative (Grant. 2023VTB0005).

scales, proposing a comprehensive load prediction method that integrates both long-term and short-term time series information.

In terms of long-term load forecasting, we have adopted advanced time series and machine learning methods to process and extract features from historical load data, establishing an accurate long-term load forecasting model. Simultaneously, to better capture instantaneous fluctuations in the system, we construct an effective short-term load prediction model through meticulous data sampling and feature extraction. To integrate the information from long-term and short-term load forecasting, we utilize the Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) [8] to decompose the load time series data. This step generates multiple Intrinsic Mode Functions (IMFs) and a Residual Component (RC), representing load changes at different time scales. By integrating these IMFs and RC through a fully connected layer, we obtain more comprehensive and detailed load prediction results, enhancing the system's robustness.

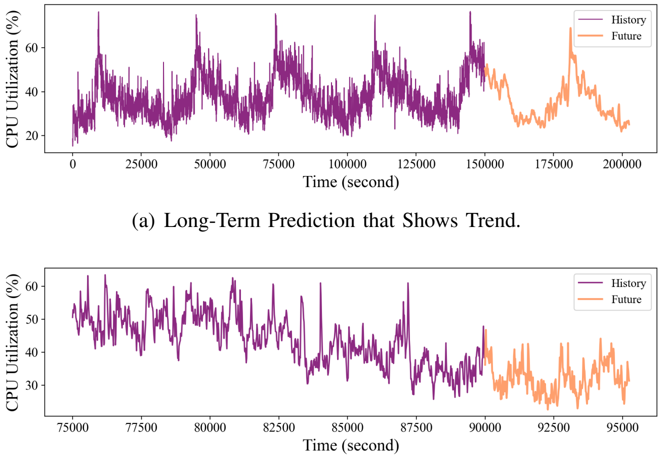
Finally, by developing elastic scaling decision rules based on integrated load prediction, we make cloud computing platforms more adaptable and able to flexibly adjust resource allocation according to real-time load conditions. This method not only improves the performance and stability of the system but also effectively reduces resource costs and promotes the sustainable development of cloud computing across various scenarios.

The **main contributions** of this work are:

- We utilize CEEMDAN to divide the load into three modes: IMF of long-term trend, IMF of short-term fluctuation, and RC. This approach helps capture information across both long-term and short-term time series by separating features.
- We present TempoScale, a cloud workload prediction method that integrates short-term fluctuations and long-term trends. This approach enhances a comprehensive understanding of temporal variations in loads, enabling more intelligent and adaptive decision-making for elastic scaling.
- We evaluate the effectiveness and availability of several baselines through realistic traces and testbed. The results demonstrate the effectiveness of our proposed method in reducing violations of service level objectives (SLO) and improving performance in terms of resource utilization.

II. Motivation and Feasibility

Existing load forecasting methods often focus on either the short-term fluctuations of the load or the long-term trends, lacking an integrated model that combines both short-term volatility and long-term trends. As shown in Fig. 1, utilizing long-term trend analysis can reveal prolonged changes in load and resource requirements, such as weekly or daily periodic variations, supporting the proactive allocation of resources over an extended period. On the other hand, employing short-term volatility analysis allows for the examination of short-term fluctuations in load and resource demands, including peak and off-peak loads, as well as sudden spikes, supporting real-time scheduling and rapid response.



(a) Long-Term Prediction that Shows Trend.

(b) Short-Term Prediction that Shows Instantaneous Fluctuations.

Fig. 1. The Focus Differs Between Long-Term and Short-Term Predictions in Time Series Forecasting.

The integration of short-term volatility and long-term trends in the study of load variations in large-scale systems can further enhance the precision of models characterizing resource requirements. This, in turn, provides robust support for the performance optimization of elastic scheduling.

To illustrate the limitations of approaches focusing solely on either short-term or long-term aspects, we conducted preliminary forecasting experiments to calculate the Mean Square Error (MSE) using experimental data from the Alibaba Cluster[1], which provides real production cluster traces. We selected two representative models for long-term and short-term time series prediction, namely Informer [9] and efficient supervised learning-based Deep Neural Network (esDNN [10]). Informer, based on the Transformer architecture, demonstrating significant improvements in long-term predictive performance compared to the original Transformer. On the other hand, esDNN, based on Gated Recurrent Unit (GRU) [11], is an algorithm used for short-term cloud load prediction. It adapts to workload variations by updating GRU control gates, overcoming limitations such as gradient vanishing and exploding. For different forecast horizons, we employed esDNN and Informer respectively for time series forecasting of loads. Each experiment was repeated 10 times, and the experimental results are presented in Table I. The collected results have all been subjected to inverse normalization.

Fig. 2 illustrates that in short-term time series prediction, particularly when the predicted time series length is less than 8 points (each point having a 30-second interval), Informer performs noticeably worse than esDNN. On the other hand, in long-term time series prediction, when the predicted time series length exceeds 8 points, Informer outperforms esDNN significantly. Moreover, as the time series length increases or

[1] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018

TABLE I
PERFORMANCE COMPARISON OF SHORT-TERM (ESDNN) AND LONG-TERM (INFORMER) PREDICTION.

History:Future	esDNN (MSE)	Informer (MSE)	Informer is X% better than esDNN
3:1	**3.65**	4.12	-12.88%
6:2	**5.18**	5.70	-10.04%
12:4	**8.07**	8.68	-7.56%
24:8	13.06	**13.05**	0.08%
48:16	19.61	**18.87**	3.77%
96:32	28.11	**26.69**	5.05%
192:64	37.53	**33.71**	10.18%

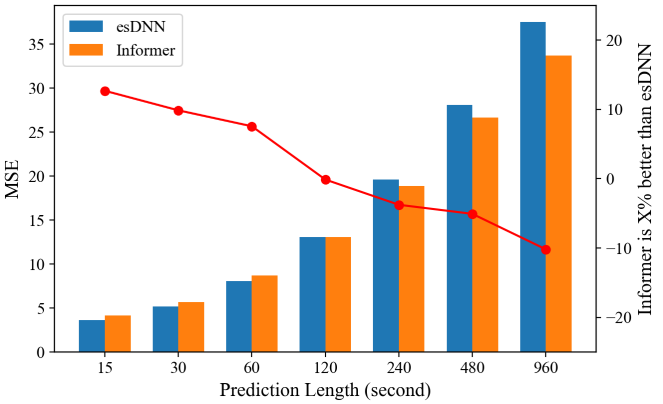


Fig. 2. Performance Comparison of Short-Term (esDNN) and Long-Term (Informer) Prediction.

decreases, the performance gap becomes more significant. This insight suggests the possibility of performance optimization through the development of an integrated algorithm that combines long-term and short-term time series prediction.

III. RELATED WORK

Many researchers have extensively investigated workload forecasting, and these studies can be categorized into three classes: 1) machine learning-based models, 2) neural network-based models, and 3) attention mechanism-based models.

A. *Traditional Machine learning-based models*

Numerous studies have been dedicated to leveraging traditional machine learning models to enhance the accuracy and efficiency of cloud workload prediction.

Prassanna et al. [12] proposed a new virtual machine consolidation technique, NMT-FOLS, which employs an Adaptive Regressive Holt-Winters Workload Predictor to identify the workload state and utilizes the prediction results to allocate user-requested tasks to the optimal VM. Xie et al. [13] proposed a hybrid model of ARIMA and triple exponential smoothing, which accurately predicts both linear and nonlinear relationships in the container resource load sequence. The weighting values of the two different models are chosen based on the sum of squares of their prediction errors over a period of time. Biswas et al. [14] proposed a new Linear Regression model to predict future CPU utilization, which is described by a straight line and a mean point. The proposed algorithm reduces energy consumption and SLA violation rates in cloud data centers. Kholidy [15] developed a novel Swarm Intelligence-Based Prediction Approach, which utilizes Particle Swarm Optimization to select optimal features from the dataset and estimate parameters for the prediction algorithms. Righi et al. [16] proposed a proactive elasticity model named Proliot. The contribution of Proliot lies in its utilization of a mathematical formalism employing ARIMA and Weighted Moving Average for predicting the behavior of Internet of Things load, enabling anticipation of scaling operations.

B. *Neural network-based models*

Traditional machine learning methods perform poorly when dealing with complex features of cloud workloads. To more effectively capture these features, researchers have turned to more advanced learning methods, with deep learning being particularly noteworthy [17], especially excelling in handling large-scale, high-dimensional data, and nonlinear relationships.

Dogani et al. [4] proposed an innovative approach utilizing Bidirectional GRU and Discrete Wavelet Transformation to enhance the accuracy of host workload prediction. Xu et al. [10] proposed an esDNN algorithm for cloud workload prediction, which adapts to workload variations by updating the gates of the GRU, overcoming the limitations of gradient disappearance and explosion. Ruan et al. [18] proposed a deep learning-based workload prediction method named CrystalLP that utilize Long Short-Term Memory (LSTM) networks. Ouhame et al. [19] proposed a Convolutional Neural Network (CNN) and LSTM model for predicting multivariate workloads. It first analyzes the input data using the vector autoregression method to filter the linear correlations among multivariate data. Then, it calculates the residual data and inputs it into the CNN layer to extract complex features of each virtual machine usage component.

C. *Attention mechanism-based models*

Attention mechanism is a new research field base on neural networks in recent years, achieving significant success [20]. The core mechanism of attentional mechanisms involves focusing resources on key components of time series input while filtering out irrelevant information [9]. Therefore, attention mechanisms can enhance the model's capability to capture the dynamic changes in time series data, thereby enhancing the performance of time series analysis and prediction tasks.

Zhou et al. [9] designed an efficient Transformer model for Long Sequence Time Series Forecasting named Informer. It employs the ProbSparse self-attention mechanism, self-attention distilling by halving cascading layer input, and a generative-style decoder, significantly improving the inference speed for long sequence predictions. Zerveas et al. [21] introduced a novel framework for multivariate time series representation learning based on the Transformer encoder

TABLE II
RELATED WORK.

Work	Types of Prediction Methods			Prediction Length		Resource Scaling in Cloud
	Machine Learning	Neural Network	Attention Based	Single	Multiple	
Dogani et al. [4]		✓	✓	✓		
Zhou et al. [9]			✓		✓	
Xu et al. [10]		✓		✓		✓
Prassanna al. [12]	✓			✓		✓
Xie et al. [13]	✓				✓	✓
Biswas et al. [14]	✓			✓		✓
Kholidy [15]	✓				✓	
Righi et al. [16]	✓			✓		
Ruan et al. [18]		✓		✓		
Ouhame et al. [19]		✓			✓	
Zerveas et al. [21]			✓		✓	
Wang et al. [22]			✓		✓	
Wu et al. [23]			✓		✓	
This paper		✓	✓		✓	✓

architecture. The framework incorporates an unsupervised pretraining scheme, which offers substantial performance benefits over fully supervised learning in downstream tasks. Wang et al. [22] proposed a novel method for time series prediction, leveraging the Transformer with a multiscale CNN. It consists of multiscale extraction and multidimensional fusion frameworks. Wu et al. [23] designed Autoformer as a novel decomposition architecture with an Auto-Correlation mechanism, breaking the preprocessing convention of time series decomposition and transforming it into a fundamental building block of deep models. This design empowers Autoformer with progressive decomposition capabilities for complex time series.

D. Critical analysis

We summarize and compare the related work in Table II. Models based on traditional machine learning are mostly effective for workloads with clear patterns, but the high variability and non-linearity of modern cloud workloads make these models less effective [24]. Neural network-based models may face issues like gradient vanishing or exploding when dealing with long sequences, especially in tasks that require considering long-term dependencies. This can make it challenging for the model to capture and learn effective information over extended time intervals. Transformer-based models, leveraging attention mechanisms, excel in long-time sequence prediction tasks, significantly improving performance. However, they also come with drawbacks, such as larger parameter sizes, complex tuning processes, and higher resource costs, leading to increased usage expenses.

Therefore, this study proposes a predictive algorithm that integrates both long-term and short-term temporal features, enabling better capture of dependencies in time series data. Utilizing long-term trend analysis to reveal the extended variations in load and resource demands supports proactive resource allocation over more extended periods. Additionally, employing short-term volatility analysis examines the short-term variations in load and resource demands, facilitating real-time scheduling and rapid responsiveness. The main difference between our work and others is that we focus more on studying the characteristics of time series data across different dimensions. We employ different types of models to process them, enabling us to leverage strengths effectively.

IV. TEMPOSCALE: A RESOURCE SCHEDULER INTEGRATING SHORT-TERM AND LONG-TERM INFORMATION

In order to address the inherent dynamics of clusters and the variability of workloads, we propose an innovative solution in this work called TempoScale. The architecture of TempoScale is illustrated in Fig. 3, the module ① represents a server cluster, we have implemented a prototype system and deployed a resource scheduler and a resource monitor[2,3], enabling real-time monitoring and resource control of the server cluster, the module ② illustrates the TempoScale algorithm, which involves three steps: 1) preprocessing of data and decomposition of IMFs, 2) processing intermediate results using different models, and 3) obtaining the final results through a Multilayer Perceptron (MLP). In the following sections, we will focus on providing a detailed description of these steps.

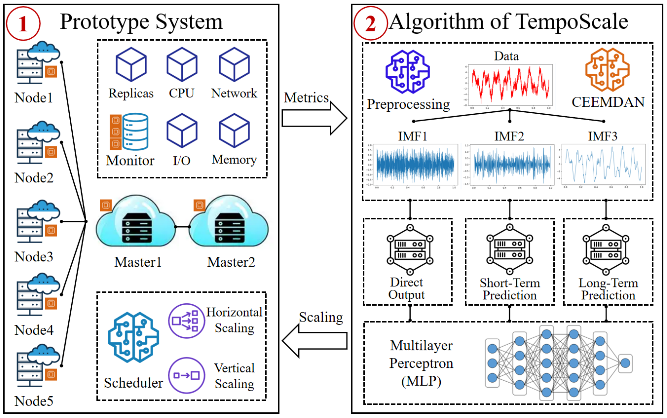


Fig. 3. The Architecture of TempoScale.

A. Preprocessing of data and decomposition of IMFs

TempoScale processes raw data exported from the monitoring system of a cloud cluster ,the monitoring system of the cloud cluster records major resource usage such as CPU, memory, disk, and network. Among these resources, CPU is considered as the most crucial and dominant resource in the computer system, and we primarily focus on the usage of CPU. Initially, TempoScale removes rows containing empty and anomalous data as they can negatively impact predictive data. Subsequently, TempoScale calculates the average value for each parameter with the same timestamp, represented as a time-ordered sequence $X(x_1, x_2, ..., x_t)$ with constant time intervals. Then, it normalizes the data to enhance the model's convergence speed and prediction accuracy. TempoScale utilizes Z-score for this purpose. Z-score normalization assumes

[2] https://prometheus.io/
[3] https://github.com/kubernetes-sigs/metrics-server

that the data approximates a normal distribution. This normalization method helps eliminate scale differences between different features. Z is calculated using Eq. (1):

$$Z = \frac{(X-\mu)}{\sigma} = \frac{X - \frac{1}{n}\sum_{i=1}^{n} X_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \frac{1}{n}\sum_{i=1}^{n} X_i)^2}}, \tag{1}$$

where X represents the original data, n is the number of data points, μ is the mean of the original dataset, σ is the standard deviation of the dataset, and Z is the standardized data value.

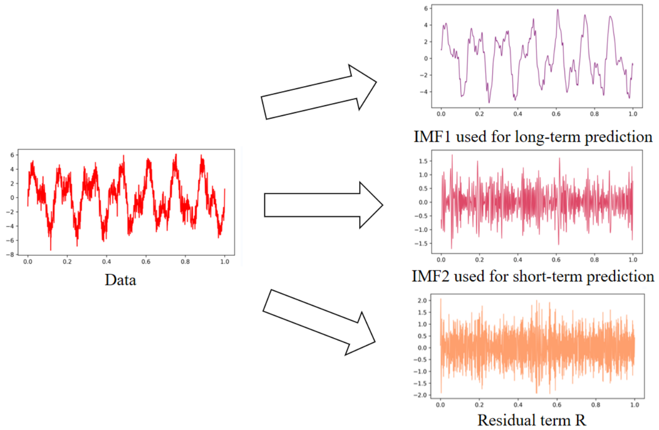


Fig. 4. Modal Decomposition for Feature Extraction.

After preprocessing the data, it is necessary to separately extract the long-term and short-term features of the time-series data for subsequent load forecasting. As shown in Fig. 4, modal decomposition decomposes complex signals into IMFs, enhancing the clarity of signal analysis and demonstrating excellent localization in the time-frequency domain. This facilitates a more accurate exploration of the local characteristics of signals. TempoScale leverages the inherent advantages of modal decomposition in time-series feature extraction, which naturally divides preprocessed data into two IMFs and a RC. The two IMFs represent the long-term and short-term features of the time-series data, respectively, and are then used as inputs for subsequent models. As illustrated in Algorithm 1, the algorithm mainly consists of five steps: **Initializing Parameters**, **Ensemble Generation**, **Ensemble Mean**, **Adaptive Noise**, and **Sifting Process**. First, parameters are initialized to set up various necessary parameters for the algorithm (lines 1-7). Then, multiple ensemble trial instances are generated by Empirical Mode Decomposition (EMD) to evaluate the algorithm's robustness under different data variations (lines 8-13). Next, the ensemble means and adaptive noise for each IMF are computed to enhance the accuracy of extracting the actual signal features (lines 14-21). Finally, the sifting process iteratively extracts IMFs and separates the residual signal, achieving the decomposition of the original signal (lines 22-28).

Algorithm 1 Performing Long-Term and Short-Term Feature Segmentation on the Data Within TempoScale.

```
Require: Signal $x(t)$
Ensure: Set of IMFs $\{c_i(t)\}$ and RC $r(t)$
 1: Initialize parameters:
 2: $N$ - Number of ensemble trials
 3: $M$ - Number of sifting iterations
 4: $T$ - Signal length
 5: $t$ - Time index
 6: $c_i(t)$ - Initial IMF estimate
 7: $r(t)$ - RC
 8: $\alpha$ - Sifting parameter
 9: Ensemble Generation:
10: for $i = 1$ to $N$ do
11:     Generate white noise series $w_i(t)$
12:     Add white noise to the signal: $y_i(t) = x(t) + w_i(t)$
13:     Perform EMD on $y_i(t)$ to obtain IMF set: $\{c_{i,1}(t), c_{i,2}(t)\}$
14: end for
15: Ensemble Mean:
16: for $k = 1$ to $2$ do
17:     Compute ensemble mean of each IMF: $\bar{c}_k(t) = \frac{1}{N}\sum_{i=1}^{N} c_{i,k}(t)$
18: end for
19: Adaptive Noise:
20: for $k = 1$ to $2$ do
21:     Compute adaptive noise for each IMF: $a_k(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(c_{i,k}(t) - \bar{c}_k(t))^2}$
22: end for
23: Sifting Process:
24: for $m = 1$ to $M$ do
25:     Extract the RC: $r(t) = x(t) - \sum_{k=1}^{2} c_k(t)$
26:     for $k = 1$ to $2$ do
27:         Sift the IMF: $\tilde{c}_k(t) = c_k(t) + \alpha a_k(t)$
28:     end for
29: end for
```

B. Processing intermediate results using different model

After modal decomposition, two IMFs representing long-term trends and short-term fluctuations will be fed into the prediction module composed of a model based on Transformer and GRU architectures (feasibility has been validated in Section II).

1) Short-term prediction model: The GRU is a type of RNN architecture designed for capturing dependencies and patterns in time series data. The gating mechanisms in GRU allow the network to selectively update and memorize information in the hidden state, enabling it to focus on relevant information while avoiding the long-term dependency issues that can lead to vanishing or exploding gradients. This makes GRU effective at capturing short-term patterns in time series data.

Assuming at time step t, given input x_t, the previous hidden state h_{t-1}, and the parameters W of the GRU, we can compute the update gate z_t, the reset gate r_t, and the candidate hidden state $\tilde{h}_t$ at the current time step [11]:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]), \tag{2}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]), \tag{3}$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t]), \tag{4}$$

where σ is the sigmoid function, $\odot$ denotes element-wise multiplication, $[\cdot]$ indicates matrix multiplication, and tanh represents the hyperbolic tangent function, which is a type of nonlinear activation function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{5}$$

Then, based on the update gate z_t and the candidate hidden state $\tilde{h}_t$, we can compute the current hidden state h_t as follows:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \tag{6}$$

where the update gate z_t controls the balance between the past hidden state h_{t-1} and the new candidate hidden state $\tilde{h}_t$. If z_t is close to 1, the model retains more of the past information; if z_t is close to 0, the model relies more on the new information. The reset gate r_t controls the influence of the past hidden state in computing the candidate hidden state $\tilde{h}_t$.

Through this gating mechanism, GRU effectively captures long-term dependencies and performs well in handling sequential data. Therefore, we adopt the algorithm based on GRU [10] as our short-term prediction model in TempoScale.

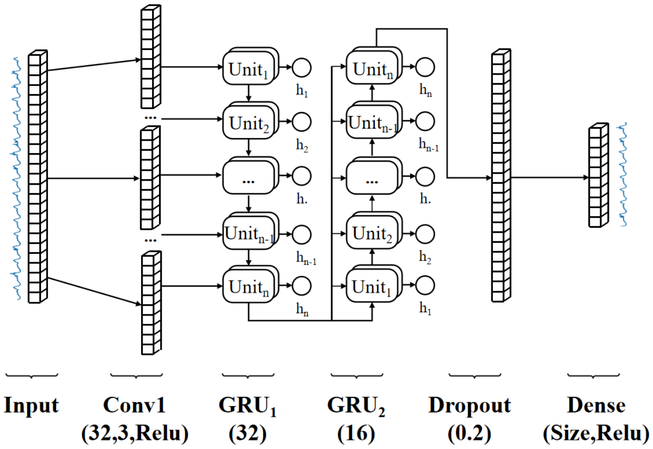


Fig. 5. The Network Structure of Short-term Prediction Model in TempoScale.

The short-term forecasting algorithm used in TempoScale is depicted in Fig. 5. First, the data is fed into a 1D CNN, which can extract features from time series data and model the short-term correlations between the time series data and subsequent trends [25]. The convolved data is then input into a two-layer GRU network, and finally, the activation function ReLU, regularization, and dense layers are applied to generate the final output. This comprehensive approach ensures a refined and well-optimized prediction based on the extracted features and short-term correlations captured during the earlier stages of processing.

2) Long-term prediction model: Methods based on attention mechanisms typically perform exceptionally well in addressing long time series prediction problems because attention mechanisms enable the model to better focus on different parts of the time series, thereby capturing long-term dependencies more effectively. In self-attention mechanisms, each element in a sequence interacts with every other element to compute weights. Specifically, for each attention head i, the attention score matrix Attention_i can be calculated using Eq. (7):

$$\text{Attention}_i = \text{softmax}\left(\frac{\mathbf{Q}_i \cdot \mathbf{K}_i^T}{\sqrt{d_k}}\right) \cdot \mathbf{V}_i. \tag{7}$$

Here, $\mathbf{Q}_i$, $\mathbf{K}_i$, and $\mathbf{V}_i$ are the query, key, and value matrices for the i-th head, and d_k is the dimension of the key vectors. The attention mechanism calculates attention weights to determine the contribution of each element. This way, the model can dynamically adjust weights based on the specific content of the input sequence, better capturing long-term dependencies and important patterns in the sequence.

However, transformer-based models need to address significant time and resource consumption issues. Many methods have been proposed to improve the performance and speed of attention-based models, reduce memory usage, and make them applicable to a wider range of time series prediction problems. For example, the original self-attention mechanism requires performing full connectivity computations across the entire input sequence, which can lead to prohibitively high computational costs for long sequences. As shown in Eq. (8), ProbSparse self-attention [9] addresses this issue by introducing a sparsity-inducing mechanism, selectively interacting with only a subset of input positions based on probabilities, thereby reducing computational complexity while preserving model performance to some extent. We applied this mechanism to TempoScale, significantly reducing time costs and improving computational efficiency.

$$M\left(\mathbf{q}_i, \mathbf{K}\right) = \ln \sum_{j=1}^{L_K} e^{\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d_k}}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d_k}}. \tag{8}$$

In Eq. (8), $M(\mathbf{q}_i, \mathbf{K})$ represents the function computing the attention score, where $\mathbf{q}_i$ is the query vector and $\mathbf{K}$ is the set of key vectors. The query vector $\mathbf{q}_i$ measures similarity between queries and keys, while $\mathbf{K}$ represents the keys in the attention mechanism. Each $\mathbf{k}_j$ is an element of the key vectors set with a length L_K. The equation aims to compute the attention score efficiently, capturing the relationships between queries and keys.

Self-attention distillation [9] aims to solve this problem by extracting the essential information captured by a large self-attention mechanism into a smaller one, while maintaining or even improving the model's performance. This method is implemented by integrating multiple pooling layers, as shown in Eq. (9):

$$\mathbf{X}_{j+1}^t = \text{MaxPool}\left(\text{ELU}\left(\text{Conv1d}\left(\left[\mathbf{X}_j^t\right]_{\text{AB}}\right)\right)\right). \tag{9}$$

In Equation (9), $\mathbf{X}_{j+1}^{t}$ represents the feature representation of layer $j+1$ at time step t, which is obtained by applying a convolution operation followed by an Exponential Linear Unit (ELU) activation function, and then a max-pooling operation. The max-pooling operation selects the maximum value in each channel of the input tensor based on a specified window size, while the ELU activation function has a non-zero gradient in the negative region to alleviate the vanishing gradient problem. The convolution operation convolves the input tensor with a convolutional kernel to generate an output tensor, applied here to $\left[\mathbf{X}_{j}^{t}\right]_{\mathrm{AB}}$ representing the feature representation of layer j at time step t, where AB denotes a specific slice or subset selection. This equation describes the process of generating the feature representation of the next layer through convolution, activation, and max-pooling operations, where each operation can be controlled by adjusting parameters to influence the feature extraction process of the model. Applying this mechanism to TempoScale enables the solution of prediction problems with longer sequences.

In traditional sequence generation tasks, each output is generated one at a time in a left-to-right manner, which can be slow and computationally expensive. The One Forward Decoder [9] directly generates the entire long sequence without the need for individual generation, thus speeding up the generation process and reducing computational costs.

The aforementioned techniques contribute to enhancing the performance of transformer-based models while reducing time and memory overheads. TempoScale incorporates these techniques into its long-term prediction module to facilitate the execution of long-term forecasts.

C. Obtaining the final results through a MLP

In the end, the output of long-term and short-term prediction models, along with RC calculated in Algorithm 1, is fed into a MLP in TempoScale to obtain a final long-term time series prediction result for scaling. The MLP introduces nonlinear transformations and higher-level feature representations, enabling more flexibility in capturing complex relationships between inputs and enhancing the model's understanding and generalization capabilities. In TempoScale, the MLP's input layer consists of 144 neurons, the output layer has 48 neurons, and there are 4 hidden layers with 192, 240, 240, and 192 neurons, respectively, all with ReLU activation functions.

Subsequently, the results are fed into the resource management system of a cloud cluster for resource auto-scaling. This step involves using the model's output to make resource management decisions, determining whether to allocate additional resources or remove excess resources. The overall goal of this process is to achieve more intelligent and efficient resource allocation, optimizing the performance of the entire system.

V. Performance Evaluations

In this section, we provide a detailed description of the dataset used and the experimental configurations. Additionally, we conducted experiments on the cluster to compare the performance of TempoScale with several state-of-the-art approaches. The results validate that TempoScale can be effectively applied to optimize cloud resource usage.

A. Experimental setup

TempoScale is mainly developed using Python 3.9. Resource scaling is performed every 15 seconds, and load prediction is conducted every 12 minutes. Load prediction utilizes data from the past 48 minutes to forecast the next 12 minutes. As shown in Table I and [9], the longer the predicted length, the greater the potential for improvement. Therefore, we'll take an intermediate value of 48 minutes for the prediction time length, and a prediction length of 12 minutes provides the system with a sufficient resource scheduling time [26]. All performance tests were conducted using a K8s cluster consisting of one master and two worker nodes. The operating system used was CentOS-7, with each node having 4 GB of memory and 4 CPU cores. The workload dataset, microservices demo application, and baseline methods used in the experiments are as follows:

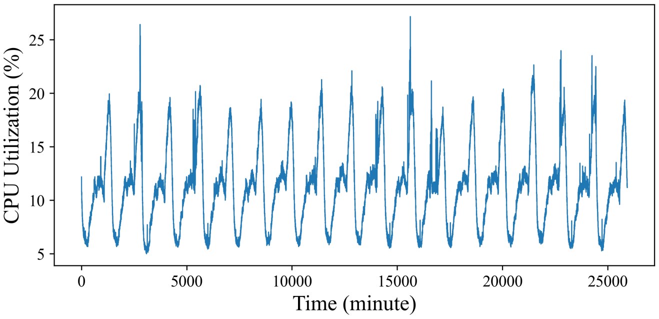


Fig. 6. CPU Utilization of *MS_10489* Over Time.

1) Workload dataset: We used the dataset from the Alibaba Cluster[4], which was collected from Alibaba production clusters consisting of over ten thousand bare-metal nodes over a period of 13 days in 2022. One of the datasets, named *MSResource*, records CPU and memory utilization of over 470,000 containers for more than 28,000 microservices in the same production cluster. It includes attributes such as *timestamp*, *msname*, *msinstanceid*, *nodeid*, *cpu_utilization*, and *memory_utilization*. This dataset accurately represents the workload characteristics of current large-scale cloud clusters. We utilized this dataset as input for workload simulation to evaluate the performance and reliability of applications or systems under various workload conditions. For experimental evaluation, we select a microservice named *MS_10489*, illustrating the variation in its resource utilization rates in Fig. 6.

2) Microservices demo application: Sock Shop[5] is a microservice application commonly used for testing purposes. It is an open-source demo application designed to demonstrate best practices in developing cloud native applications. Sock

[4] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022

[5] https://microservices-demo.github.io/

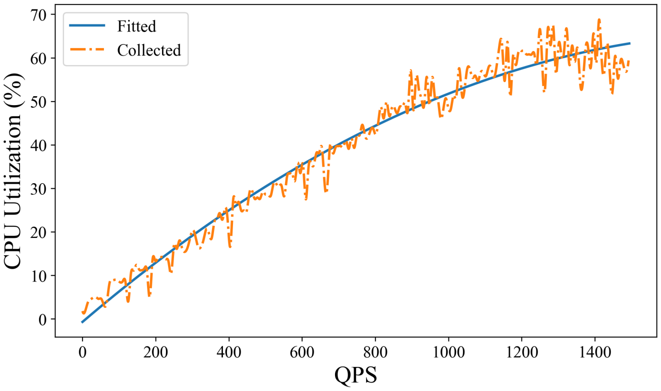


Fig. 7. The Profiling Between QPS and CPU Utilization.

Shop simulates an online shopping platform and consists of eight microservices, each serving a specific function such as shopping carts, payments, and inventory.

3) Baseline methods: The three baseline methods used in our experiments are state-of-the-art and representative methods of the three categories discussed in Section III.

1) **ARIMA** [27]: It effectively captures trends and seasonality in time series data. Due to its simplicity and widespread application, ARIMA is often used as a standard for comparing the performance of other time series forecasting models.
2) **esDNN** [10]: This is an optimized method based on GRU. It is designed to be simple, with fewer parameters, easy to train, and computationally efficient. It serves as an ideal benchmark for capturing sequential patterns in various tasks..
3) **Informer** [9]: This is an improved method based on Transformer. Due to its outstanding performance in sequence tasks and the effectiveness of its self-attention mechanism in handling long-term dependencies, it is the preferred benchmark method for many sequence data processing tasks.

B. Profiling

Due to the varying relationship between CPU utilization and Queries Per Second (QPS) on each machine, which depends on factors such as hardware configuration, load characteristics, and the running software system, it is necessary to establish the profile between CPU utilization and QPS before starting the experiment. This helps determine the expected CPU utilization levels at different QPS levels, ensuring the accuracy and comparability of the experimental results. The profiling results are shown in Fig. 7, the experimental results are similar to those of previous work [28].

C. Predictive Evaluation

Table III presents the forecast results of ARIMA, esDNN, Informer, and TempoScale on Alibaba's 2022 trace data, evaluated using the Mean Square Error (MSE), coefficient of determination (R^2), and Mean Absolute Percentage Error (MAPE), The equations are as follows:

TABLE III
COMPARISON OF PREDICTION RESULTS FROM DIFFERENT METHODS.

Method	ARIMA	esDNN	Informer	TempoScale
MSE	0.000099	0.000085	0.000073	**0.000069**
MAPE	0.049752	0.047577	0.048178	**0.044682**
R^2	0.359305	**0.400834**	0.358701	0.365229

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{10}$$

$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100, \tag{11}$$

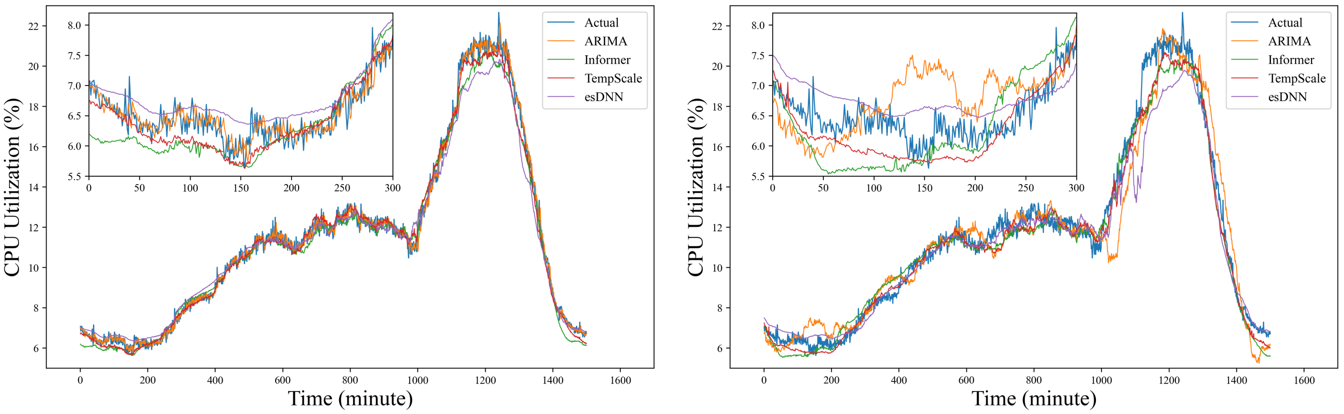
$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{12}$$

where n is the sample size, y_i represents the ith observed value, $\hat{y}_i$ represents the model's predicted value for the ith observation, $\bar{y}$ represents the mean of the observed values.

As shown in Table III, we have highlighted the best value for each metric. Assessment metrics are calculated on a per-prediction-period basis, and the displayed results have all been subjected to inverse normalization. The data demonstrates that our proposed method, TempoScale, outperformed others in terms of both MSE and MAPE. Specifically, in terms of MSE, TempoScale outperforms ARIMA by 30.43%, esDNN by 18.78%, and Informer by 5.80%. In terms of MAPE, TempoScale outperforms ARIMA by 10.19%, esDNN by 6.08%, and Informer by 7.26%. Although it did not yield the highest result in terms of R^2, it remained at a satisfactory level. The above results suggest that TempoScale exhibits higher accuracy and reliability in forecasting Alibaba Cloud workload data.

To investigate the impact of forecast length on accuracy, we selected a subset of forecast results (1500 data points, with each data point representing one minute) for visualization. Fig. 8(a) illustrates the forecast results for the first time slice within the forecasting period, demonstrating the outstanding performance of ARIMA, with its predictions almost perfectly aligning with the actual data. Fig. 8(b) demonstrates the forecast results for the last time slice within the forecasting period. Upon inspection in the zoomed-in figure, it is evident that Informer and esDNN either underestimated or overestimated the actual data, whereas TempoScale was able to predict the actual data more accurately.

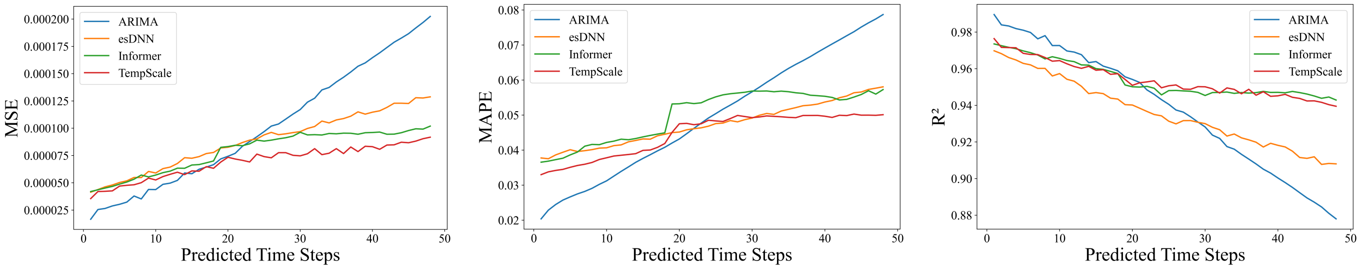
In order to study the impact of time steps on prediction accuracy, we conducted statistical calculations to analyze the variations in performance metrics across different time steps. It is evident from Fig. 9 that the performance of ARIMA deteriorates almost linearly with the increase in the length of the time interval, as indicated by MSE, MAPE, and R^2 in Fig. 9(a), Fig. 9(b), and Fig. 9(c). The reason lies in



(a) The First Time Step of the Prediction Interval.

(b) The Last Time Step of the Prediction Interval.

Fig. 8. Comparison Between Predicted and Actual Values Based on Alibaba Dataset.



(a) MSE.

(b) MAPE.

(c) R^2.

Fig. 9. Comparison of Prediction Performance Under Different Time Steps Based on Alibaba Dataset.

that ARIMA may overly rely on the most recent historical data in time series forecasting tasks, treating this segment as the prediction result while overlooking earlier historical data variations. This behavior could lead to a strong correlation between the prediction results and the most recent historical data segment. Conversely, the degradation rate of performance for esDNN and Informer is relatively slower, this results from that they are able to effectively model long-term dependencies, capturing long-term correlations in time series data through techniques such as self-attention mechanisms, thereby maintaining good performance even with longer time steps. TempoScale combines the advantages of both, thereby exhibits nearly the slowest deterioration in performance, demonstrating its outstanding performance in long-term forecasting.

D. Workload Prediction with Auto-Scaling Evaluation

The benefits of auto-scaling lie in its ability to effectively manage costs and improve system performance and availability. By automatically adjusting resource usage based on actual workload, this technology avoids resource waste and shortages, thus saving costs. Additionally, it ensures system performance during high workloads and reduces resource usage during low workloads, enabling efficient system operation.

Therefore, to further demonstrate the capability of the proposed method and develop an efficient auto-scaling approach, we integrate methods including ARIMA, esDNN, Informer and TempoScale into the prototype system based on K8s developed by us are conducted to evaluated.

We first simulate workload variations in a real cluster environment using Locust[6], based on the results of the profiled data in Section V-B. Then, we utilize the elastic scaling system integrated with workload prediction methods to evaluate the effectiveness of this method in cost management and performance improvement. We focus on vertical scaling of containers, with predictions operating on a 12-minute cycle. Vertical scaling involves adjusting the resource configuration of individual instances, such as increasing the CPU quota or memory limit of containers in a containerized environment. The amount of scaling operations is based on the predictions of CPU usage. Our goal is to enhance system performance by minimizing response time and avoiding SLO violations while keeping the total resource budget constant. Here, the resource budget refers to the cumulative product of resource supply within each time unit, represented as $\int R_t \, dt$, where R_t is the resource provided at time t.

In the experiments, detailed average response time is shown in Fig. 10, while CPU allocation is depicted in Fig. 11, with units in milli-cores (m), this unit of measurement is

[6] https://locust.io/

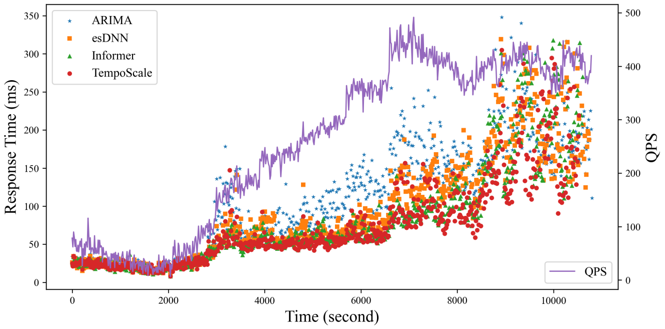


Fig. 10. Comparison of Average Response Times during Runtime.

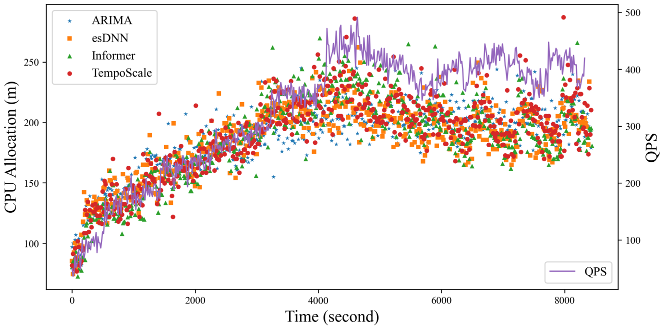


Fig. 11. Comparison of CPU Allocation during Runtime.

commonly used to specify the amount of CPU resources that an application or container can utilize in cloud computing and containerized environments. The figures include CPU allocation, average response time, and workload size for each method during runtime. Additionally, QPS values are plotted on the secondary axis of each figure to facilitate direct comparison. From the figures, it can be observed that during the initial stable workload phase, all four methods maintain system stability with relatively low average response times. However, in the long-term stages, workload spikes and variations lead to varying degrees of response time increases. Notably, TempoScale demonstrates lower performance impact compared to other methods, while ARIMA exhibits the poorest performance. This aligns with the results from the previous section on workload prediction experiments.

Finally, we compare the response time and SLO violation rates as shown in Table IV, CPU budget and usage are measured in m·s, representing the product of time and resource quantity, the experiments are conducted under roughly the same resource budget to compare resource usage. The average response time of the TempoScale method is 76.09 ms, achieving best performance. It outperforms ARIMA by 31.15%, esDNN by 15.81%, and Informer by 5.58%. As for the SLO violation rates, users' acceptance may vary depending on specific application scenarios and business requirements. Generally, most users expect fast response times and high-performance services, so shorter SLOs (e.g., a few hundred milliseconds or shorter) are typically considered ideal. Here, we set two SLO targets, 200 ms and 250 ms, reflecting these expectations. When the SLO is set to 200 ms, the violation rate for TempoScale is 4.58%, which is 6.11% lower than ARIMA, 3.48% lower than esDNN, and 4.03% lower than Informer.

TABLE IV
COMPARISON OF RESPONSE TIME, SLO VIOLATION RATE AND RESOURCE USAGE.

Method	ARIMA	esDNN	Informer	TempoScale
Average Response Time (ms)	110.52	90.38	80.59	**76.09**
99th Percentile Response Time (ms)	378.85	327.82	319.50	**314.91**
Maximum Response Time (ms)	471.07	409.75	399.45	**396.02**
SLO (250 ms) Violation (%)	2.64	2.50	3.47	**1.39**
SLO (200 ms) Violation (%)	10.69	8.06	8.61	**4.58**
CPU Budget (m·s)	119938.00	119285.52	120973.88	119912.67
CPU Usage (m·s)	92466.70	97927.95	103548.11	**108095.69**

Based on the above results, it can be concluded that compared to more primitive forecasting algorithms such as ARIMA, TempoScale can improve performance by over 30%. TempoScale can also improve performance by 5-10% over novel and innovative algorithms such as Informer and esDNN, which have been proposed recently. Moreover, the optimization approach proposed by TempoScale will also contribute to efficiency enhancement for enterprises, promoting the development of various scenarios (e.g. auto-scaling) in cloud.

VI. CONCLUSIONS AND FUTURE WORK

In order to address the inherent dynamics of clusters and the variability of workloads, we have proposed an innovative solution called TempoScale. It is designed to better capture the correlations in time series data, enabling more intelligent and adaptive elastic scaling decisions. TempoScale utilizes long-term trend analysis to reveal the changes in workload and resource demands, supporting proactive resource allocation over extended periods. Additionally, it employs short-term volatility analysis to examine variations in workload and resource demands, facilitating real-time scheduling and rapid responsiveness. We conducted experiments on top of K8s with realistic data from Alibaba, and the results demonstrate the feasibility of our proposed method. Our approach not only enhances system performance and stability but also effectively reduces resource costs, promoting the sustainable development of cloud computing across various industries. However, the framework of TempoScale is built on individual microservices without fully considering the invocation dependency graph [29] and emergency measures in cases of inaccurate predictions. In future work, we plan to address these aspects to enhance TempoScale's capability in handling microservices with complex dependency graphs and improving robustness in special situations, and exploring additional integration possibilities with cloud management platforms.

SOFTWARE AVAILABILITY

The codes have been open-sourced to https://github.com/lifwen/TempoScale for research usage.

References

[1] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, “Elasticity in cloud computing: State of the art and research challenges,” *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2018. [Online]. Available: https://doi.org/10.1109/TSC.2017.2711009

[2] E. A. Brewer, “Kubernetes and the path to cloud native,” in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2015, p. 167. [Online]. Available: https://doi.org/10.1145/2806777.2809955

[3] M. Chrysopoulos, I. Konstantinou, and N. Koziris, “Deep reinforcement learning in cloud elasticity through offline learning and return based scaling,” in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, 2023, pp. 13–23. [Online]. Available: https://doi.org/10.1109/CLOUD60044.2023.00012

[4] J. Dogani, F. Khunjush, and M. Seydali, “Host load prediction in cloud computing with discrete wavelet transformation (dwt) and bidirectional gated recurrent unit (bigru) network,” *Computer Communications*, vol. 198, pp. 157–174, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422004479

[5] B. Suleiman, M. M. Fulwala, and A. Zomaya, “A framework for characterizing very large cloud workload traces with unsupervised learning,” in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, 2023, pp. 129–140. [Online]. Available: https://doi.org/10.1109/CLOUD60044.2023.00023

[6] T. Wu, M. Pan, and Y. Yu, “A long-term cloud workload prediction framework for reserved resource allocation,” in *2022 IEEE International Conference on Services Computing (SCC)*, 2022, pp. 134–139. [Online]. Available: https://doi.org/10.1109/SCC55611.2022.00030

[7] J. Bi, H. Ma, H. Yuan, and J. Zhang, “Accurate prediction of workloads and resources with multi-head attention and hybrid lstm for cloud data centers,” *IEEE Transactions on Sustainable Computing*, vol. 8, no. 3, pp. 375–384, 2023. [Online]. Available: https://doi.org/10.1109/TSUSC.2023.3259522

[8] M. A. Colominas, G. Schlotthauer, and M. E. Torres, “Improved complete ensemble emd: A suitable tool for biomedical signal processing,” *Biomedical Signal Processing and Control*, vol. 14, pp. 19–29, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809414000962

[9] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond efficient transformer for long sequence time-series forecasting,” in *AAAI Conference on Artificial Intelligence*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:229156802

[10] M. Xu, C. Song, H. Wu, S. S. Gill, K. Ye, and C. Xu, “Esdnn: Deep neural network based multivariate workload prediction in cloud computing environments,” *ACM Trans. Internet Technol.*, vol. 22, no. 3, aug 2022. [Online]. Available: https://doi.org/10.1145/3524114

[11] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://aclanthology.org/D14-1179

[12] J. Prassanna and N. Venkataraman, “Adaptive regressive holt–winters workload prediction and firefly optimized lottery scheduling for load balancing in cloud,” *Wirel. Netw.*, vol. 27, no. 8, p. 5597–5615, nov 2021. [Online]. Available: https://doi.org/10.1007/s11276-019-02090-8

[13] Y. Xie, M. Jin, Z. Zou, G. Xu, D. Feng, W. Liu, and D. Long, “Real-time prediction of docker container resource load based on a hybrid model of arima and triple exponential smoothing,” *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1386–1401, 2022. [Online]. Available: https://doi.org/10.1109/TCC.2020.2989631

[14] N. K. Biswas, S. Banerjee, U. Biswas, and U. Ghosh, “An approach towards development of new linear regression prediction model for reduced energy consumption and sla violation in the domain of green cloud computing,” *Sustainable Energy Technologies and Assessments*, vol. 45, p. 101087, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213138821000977

[15] H. A. Kholidy, “An intelligent swarm based prediction approach for predicting cloud computing user resource needs,” *Computer Communications*, vol. 151, pp. 133–144, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419303329

[16] R. da Rosa Righi, E. Correa, M. M. Gomes, and C. A. da Costa, “Enhancing performance of iot applications with load prediction and cloud elasticity,” *Future Generation Computer Systems*, vol. 109, pp. 689–701, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17329229

[17] P. T. Yamak, L. Yujian, and P. K. Gadosey, “A comparison between arima, lstm, and gru for time series forecasting,” in *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, ser. ACAI ’19. New York, NY, USA: Association for Computing Machinery, 2020, p. 49–55. [Online]. Available: https://doi.org/10.1145/3377713.3377722

[18] L. Ruan, Y. Bai, S. Li, S. He, and L. Xiao, “Workload time series prediction in storage systems: a deep learning based approach,” *Cluster Computing*, vol. 26, pp. 25–35, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231607863

[19] S. Ouhame, Y. Hadi, and A. Ullah, “An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model,” *Neural Comput. Appl.*, vol. 33, no. 16, p. 10043–10055, aug 2021. [Online]. Available: https://doi.org/10.1007/s00521-021-05770-9

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[21] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, “A transformer-based framework for multivariate time series representation learning,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2114–2124. [Online]. Available: https://doi.org/10.1145/3447548.3467401

[22] Z. Wang and Y. Guan, “Multiscale convolutional neural-based transformer network for time series prediction,” *Signal, Image and Video Processing*, 10 2023. [Online]. Available: https://doi.org/10.1007/s11760-023-02823-5

[23] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting,” in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 22 419–22 430. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/bcc0d400288793e8bdcd7c19a8ac0c2b-Paper.pdf

[24] M. Chen, M. R. Read, P. Arroba, and R. Buyya, “En-beats: A novel ensemble learning-based method for multiple resource predictions in cloud,” in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, 2023, pp. 144–154. [Online]. Available: https://doi.org/10.1109/CLOUD60044.2023.00025

[25] D. Xu, W. Cheng, B. Zong, D. Song, J. Ni, W. Yu, Y. Liu, H. Chen, and X. Zhang, “Tensorized lstm with adaptive shared memory for learning trends in multivariate time series,” in *AAAI Conference on Artificial Intelligence*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:210177610

[26] M. Xu, L. Yang, Y. Wang, C. Gao, L. Wen, G. Xu, L. Zhang, K. Ye, and C. Xu, “Practice of alibaba cloud on elastic resource provisioning for large-scale microservices cluster,” *Software: Practice and Experience*, vol. 54, no. 1, pp. 39–57, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3271

[27] J. Bi, H. Yuan, S. Li, K. Zhang, J. Zhang, and M. Zhou, “Arima-based and multiapplication workload prediction with wavelet decomposition and savitzky–golay filter in clouds,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–12, 2024. [Online]. Available: https://doi.org/10.1109/TSMC.2023.3343925

[28] A. Jindal, V. Podolskiy, and M. Gerndt, “Performance modeling for cloud microservice applications,” in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 25–32. [Online]. Available: https://doi.org/10.1145/3297663.3310309

[29] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, “Characterizing microservice dependency and performance: Alibaba trace analysis,” in *Proceedings of the ACM Symposium on Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2021, p. 412–426. [Online]. Available: https://doi.org/10.1145/3472883.3487003