IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 19 April 2024, accepted 28 April 2024, date of publication 3 May 2024, date of current version 14 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3396652

RESEARCH ARTICLE

Efficient Container Scheduling With Hybrid Deep Learning Model for Improved Service Reliability in Cloud Computing

JUEUN JEON[1], SIHYUN PARK[2], BYEONGHUI JEONG[3], AND YOUNG-SIK JEONG[1], (Member, IEEE)
[1]Department of AI·SW Convergence, Dongguk University, Seoul 04620, Republic of Korea
[2]Department of Multimedia Engineering, Dongguk University, Seoul 04620, Republic of Korea
[3]Department of Computer Science and Artificial Intelligence, Dongguk University, Seoul 04620, Republic of Korea

Corresponding author: Young-Sik Jeong (ysjeong@dongguk.edu)

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00047, Development of microservices development/operation platform technology that supports application service operation intelligence).

ABSTRACT In a cloud computing environment, the container scheduling technique ensures reliability for containerized applications by selecting nodes that satisfy various resource requirements and then deploying containers. If the initial resources of a container are over-allocated, resources may be wasted, or other containers that are waiting in a scheduling queue may not be allocated. However, if resources are under-allocated, service disruptions may occur due to node overbooking, and service reliability cannot be ensured. Therefore, in this study, a forecasted resource-evaluating scheduler (FoRES) is proposed as a container scheduling technique that ensures resource efficiency and service reliability. FoRES predicts future CPU and memory usage by using a time-series decomposition-based hybrid forecasting (DeHyFo) model that combines multiple linear regressions with the LightTS model. FoRES then calculates the optimal scheduling decisions that minimize idle resources and node overload by applying an efficient resource utilization (SERU) scoring function to the predicted resource usage. Evaluating the performance of FoRES based on various scenarios improved resource efficiency and service reliability by up to 2.07 and 2.32 times, respectively, compared with existing scheduling techniques, even if the initial resources of the container were inefficiently allocated.

INDEX TERMS Cloud computing, container scheduling, deep learning, resource efficiency, service reliability, time-series forecasting.

I. INTRODUCTION

Many IT organizations have recently implemented container-based cloud computing environments to improve the scalability and flexibility of their cloud services. Containers are lighter and have lower overheads than virtual machines (VMs) because they share the host's operating system (OS), unlike VMs that require their own OS and a hypervisor [1], [2]. To efficiently and automatically manage the rapidly growing number of containers, various container orchestration platforms (COP) have emerged [3], [4].

The COP container scheduling technique selects nodes by considering the resource allocation of the container and the requirements of the user [5], [6]. To ensure service reliability, the COP then selects the node that best satisfies the scheduling policy among the selected nodes, and it deploys the container. However, existing container scheduling techniques perform scheduling inefficiently owing to the inadequate resource allocation of the initially set container [7]. If the initial resources of a container are over-allocated, some resources are left idle due to the allocation of unneeded

The associate editor coordinating the review of this manuscript and approving it for publication was Hang Shen.

resources, and some containers in the scheduling queue are not deployed [8], [9]. However, if a container's resources are under-allocated, resource interference and overbooking will occur because multiple containers are deployed on a single node, which complicates the interdependencies of the containers [10].

Various studies on container scheduling utilize artificial intelligence (AI) to predict future workloads in order to maximize resource utilization by appropriately allocating initial resources [11], [12]. Although these studies minimize idle resources by utilizing predicted resource usage, they fail to accurately predict certain workloads that include irregular noise. Such prediction errors lead to improper resource allocation, resulting in overload.

In this study, a forecasted resource-evaluating scheduler (FoRES) is proposed to reduce resource waste and service disruptions that are caused by inadequate resource allocation. Using a time-series decomposition-based hybrid forecasting (DeHyFo) model, FoRES predicts future CPU and memory consumption by analyzing the historical data for containers gathered during service operations. Furthermore, FoRES calculates future resource usage from the predicted resource usage and selects nodes that ensure reliable service. Finally, FoRES selects a node that ensures resource efficiency by using the efficient utilization (SERU) function to score the selected nodes, and then it deploys the containers. The main contributions of FoRES can be summarized as follows:

- FoRES predicts workloads with high accuracy that consist of various patterns using the DeHyFo model, which specializes in time-series data prediction.
- FoRES reduces resource waste and overload by scoring predicted resource usage through the SERU function and making optimal scheduling decisions.
- FoRES improves resource efficiency and service reliability compared to existing scheduling techniques, despite inadequate resource allocation in containers.

The remaining sections of this paper are organized as follows. Existing studies on container scheduling techniques are described in Section II. Section III describes the scheme of FoRES. Section IV describes the architecture and implementation of FoRES in a container-based cloud computing environment. In Section V, we evaluate the performance of FoRES on a variety of datasets. Section VI summarizes our findings on FoRES and suggests future research directions.

II. RELATED WORKS

Several studies have been conducted on container scheduling using heuristic and meta-heuristic approaches, as well as AI, with the intent to deploy containers efficiently in a cloud computing environment. Practical container scheduling can be done using a heuristic algorithm with a multi-objective function. However, as they rely on certain rules, calculating the optimal scheduling decision within a limited time is difficult. Meta-heuristic-based container scheduling techniques perform container scheduling that achieves multiple objectives by using metaheuristic algorithms that are intended for optimization problems. However, the operational complexity increases when parameters are continuously and manually adjusted to calculate optimal scheduling decisions. However, AI-based container scheduling techniques deploy containers more efficiently than do existing container scheduling techniques by using AI models that are primarily specialized for predictive analysis and optimization problems. This section describes the latest AI-based container scheduling techniques.

Yang and Chen [11] proposed a container scheduling strategy (hereafter referred to as GLP-S) based on a predictive model that consisted of a gray model (GM) and a long short-term memory (LSTM) model to reduce the resource fragmentation that occurs when container scheduling is performed. GLP-S predicted future resource usage by analyzing the patterns of past usage with GM(1,1) and LSTM models and then rescheduled to minimize idle resources. GLP-S reduced resource fragmentation by up to 44% compared with the kube-scheduler provided by Kubernetes.

Liu et al. [12] proposed the CNCUP, which reduces power consumption while ensuring a service-level agreement (SLA) for green cloud computing. The CNCUP predicted the resource utilization of physical machines (PMs) using a linear regression-based prediction model, and then it performed container scheduling and migration. In comparison with existing scheduling algorithms, CNCUP reduced power consumption, SLA violations, and the number of container migrations.

Mehta et al. [13] proposed WattsApp to minimize power cap violations in servers. WattsApp predicted future power consumption by analyzing historical CPU and memory usage, the number of network requests, and disk I/O using a linear regression-based neural network (NN). WattsApp performed migration and resource deallocation using the predicted power consumption during each container scheduling cycle. WattsApp improved power efficiency and cluster safety over existing power capping techniques.

Tuli et al. [14] proposed a holistic resource management technique for energy-efficient cloud computing using AI (HUNTER) to mitigate energy waste in a hybrid public-private cloud consisting of heterogeneous resources. HUNTER predicted the quality of service (QoS) score by analyzing the host temperature record and the correlation between scheduling decisions and tasks using a gated graph convolution network (GGCN)-based surrogate model. HUNTER performed scheduling based on the predicted QoS score, and incurs lower energy costs than do existing schedulers.

Carvalho and Macedo [15] proposed a quality of experience (QoE)-aware container scheduler to solve the problem of the reduced user QoE that results from the interdependencies between multiple applications. The proposed scheduler analyzed the Spearman correlation between the QoE and various metrics extracted by the application to extract metrics that had a high impact on QoE. In addition, it predicted future QoE by analyzing the patterns of the selected metrics using LSTM and gated recurrent unit (GRU) models. The proposed scheduler ensured the service-level objective (SLO) and improved the QoE by calculating the scheduling decisions that account for the predicted QoE.

Calheiros et al. [16] proposed proactive dynamic resource provisioning approach to maintain the QoS and manage costs effectively. The proposed model predicted future workload using autoregressive integrated moving average (ARIMA), and then it determined the number of VMs to allocate based on the performance of running VMs and predicted workload. It achieved a prediction accuracy of up to 91%, and improved resource utilization efficiency while minimizing the impact on QoS, which was reflected in user response time.

Bi et al. [17] proposed VAMBiG to predict large-scale workload including noise with high accuracy. This hybrid prediction model extracted low-frequency and high-frequency feature from the nonlinear time-series data using variational mode decomposition, and then it filtered the noise by using adaptive Savitzky-Golay filter. VAMBiG analyzed long-term dependencies and features in the time-series data using a hybrid deep learning model, which combined bidirectional LSTM (Bi-LSTM), grid LSTM, and attention mechanism, predicting workload, CPU usage, RAM usage. VAMBiG's prediction results for workload, CPU, and RAM showed an average R^2 of 0.946, 0.9405, and 0.914, respectively, demonstrating superior performance compared to other RNN-based deep learning models.

Xie et al. [7] proposed a hybrid model to optimize the scheduling of resource usage through container resource load prediction. The proposed model accurately predicted the linear and nonlinear relationship of container resource load sequence by removing the random fluctuation of container resource load sequence through ARIMA model and smoothing the container resource load sequence through triple exponential smoothing. It improved the prediction accuracy by up to 203.72% with less time overhead compared to existing prediction models.

The proposed FoRES predicts future resource usage by analyzing the main patterns and bursts of workload separately using a prediction model that specializes in the field of time-series forecasting (TSF) to increase the accuracy of predictions for future workloads. Based on the prediction results, FoRES calculates the optimal scheduling decision by considering the future load of each container. FoRES also reduces idle resources relative to existing container scheduling techniques, and it ensures container runtimes.

III. SCHEME OF FoRES

In this study, FoRES is proposed for determining the optimal scheduling decision based on predicted resource usage to improve resource efficiency and service reliability in a container-based cloud computing environment. Figure 1 shows the overall scheme of FoRES.

FoRES consists of three steps for efficient container scheduling. First, the metrics provider step extracts the resource usage metrics for the container, and then the future resource forecaster step predicts the future resource usage. Finally, the optimal decision scheduler step filters the nodes based on the predicted resource usage and the node-specific resource usage, and then it deploys a container to the optimal node. Table 1 describes all the symbols used in this paper.

TABLE 1. Summary of symbols used in constructing FoRES.

Symbol	Description
C_{us}, C_{time}	Resource usage and operation time of container
Tr, Re	Trend and remainder component of C_{us}
N_{ut}, N_{cap}	Resource utilization and resource capacity of a node
L_{node}, L_{filter}, L_{const}	List of all nodes, filtered nodes, and containers with constrained scheduling
R_{ut}	Future resource utilization of a node when a container is deployed
ℓ, u	Minimum and maximum thresholds of resource utilization
n, m	Minimum and maximum scores of the SERU function

A. METRICS PROVIDER

Before we performed the metrics provider step, the container's system metrics were extracted using the container-monitoring tool, and they were then stored in the historical database. Subsequently, in the metric provider step of FoRES, when a container enters the scheduling queue, the historical resource consumption of that container is extracted from the historical database. Here, FoRES extracts only the system metrics CPU and memory, which are the basic requirements for creating a container.

B. FUTURE RESOURCE FORECASTER

To accurately predict future resource usage based on past resource usage data collected by containers, the future resource forecaster step consists of a preprocessing step and a DeHyFo model. First, the preprocessing step vectorizes and normalizes the extracted resource usage metrics from the metrics provider step. FoRES then inputs the generated multivariate time-series data into the DeHyFo model, which consists of a moving average smoothing technique, multiple linear regression layers, and a LightTS layer [18] to analyze and learn the features of the time-series data. Figure 2 shows the overall architecture of the DeHyFo model built into FoRES.

The DeHyFo model first derives the Tr from the C_{us} using a moving average smoothing technique, as shown in (1), to identify unique patterns in diverse and complex resource usage. In addition, to consider bursts in the workload, the DeHyFo model uses the calculated Tr to derive the Re from C_{us}, as shown in (2). Here, w denotes the window size of moving average smoothing.

$$Tr_t = \frac{1}{w} \times \left(\sum_{i=1}^{w} C_{us}^{t+i-\lceil \frac{w}{2} \rceil} \right) \tag{1}$$

$$Re_t = C_{us}^t - Tr_t \tag{2}$$

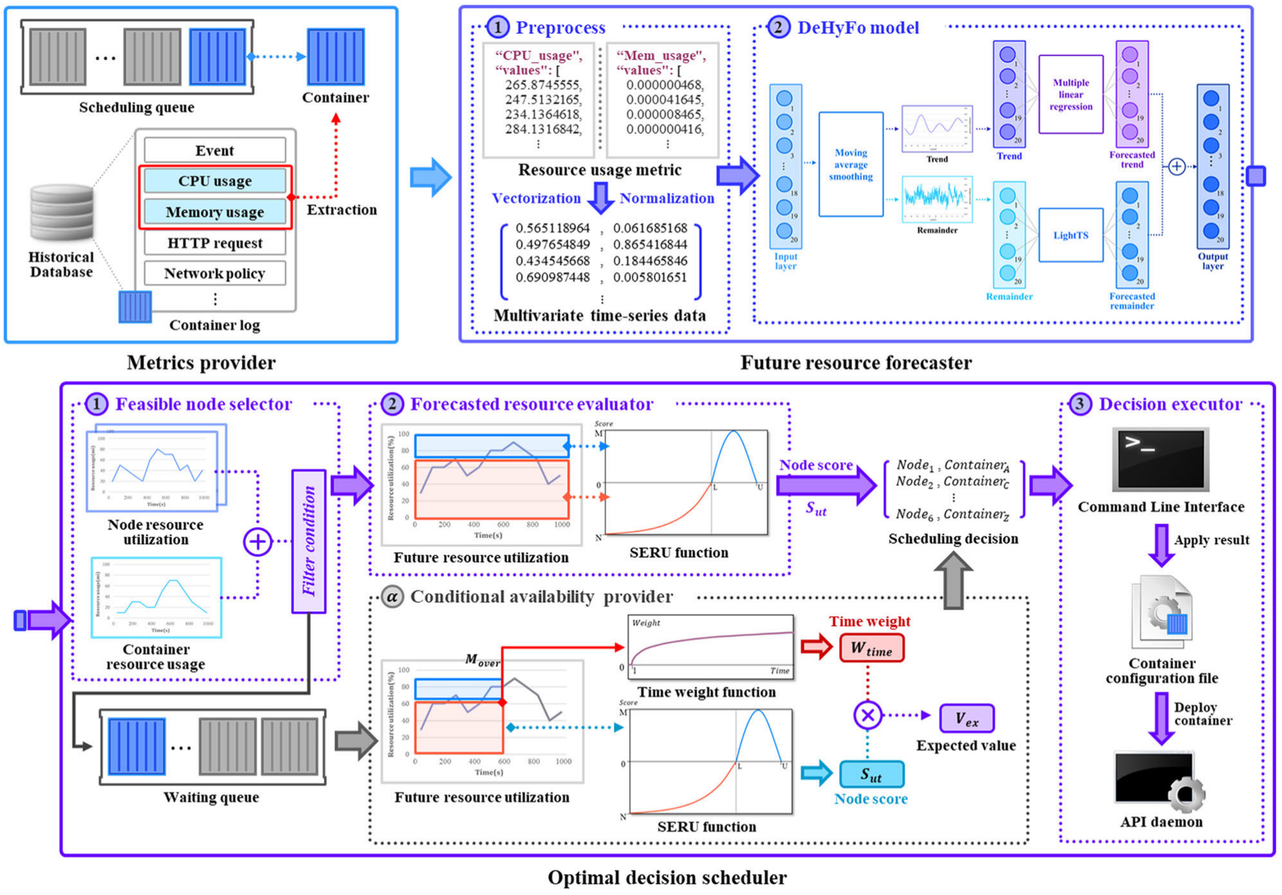


FIGURE 1. Overall scheme of forecasted resource-evaluating scheduler (FoRES).

Tr has a monotonic structure because it consists of linear features with non-linear noise, and the outliers are removed using a moving-average smoothing technique. However, *Re* has a complex structure that contains irregular noise and outliers. Therefore, to effectively analyze the patterns of resource usage, FoRES learns the two decomposed components in different layers. *Tr* first uses multiple linear regression layers that are specialized in linear pattern analysis to learn the main patterns in the time-series data. For *Re*, the LightTS layer, which extracts temporal information from complex time-series data that has irregular features, is used to effectively analyze and learn workload bursts.

Existing deep learning models that are specialized in time-series data analysis, such as recurrent neural networks (RNN), graph neural networks (GNN), and transformers, have a complex structure, and this reduces their efficiency because of the high computational cost and loss of temporal information [19]. In contrast, the LightTS model consists of a simple multi-layer perceptron (MLP) structure, which predicts *Re* with better efficiency than do existing deep learning models.

The LightTS layer of the DeHyFo model consists of feature extraction and prediction components, each of which uses an information exchange block (IEBlock) to extract temporal features for *Re*. At this point, the IEBlock sequentially performs temporal projection, channel projection, and output projection procedures, which consist of a linear transformation structure. First, the temporal projection shares *Re* temporal dimension weights to analyze the features at sequential time points. The channel projection then shares the channel dimensions of *Re* to analyze the features for different temporal resolutions. Finally, the output projection extracts the temporal features for *Re* by mapping the analyzed temporal features to the final output length.

First, the feature extraction part of the LightTS layer divides *Re* into two subsequences, short-term and long-term, so that the long-term and short-term patterns of *Re* can be considered separately. To analyze each subsequence independently without considering its interdependency, the short-term sequence extracts short-term patterns through continuous sampling, whereas the long-term sequence extracts long-term patterns through interval sampling. In the prediction phase, the temporal features for the long- and short-term time points, which were extracted during each sampling process, are concatenated and analyzed to predict the future *Re*.

Finally, the DeHyFo model of FoRES calculates the future resource consumption by adding the predicted *Tr* and *Re*

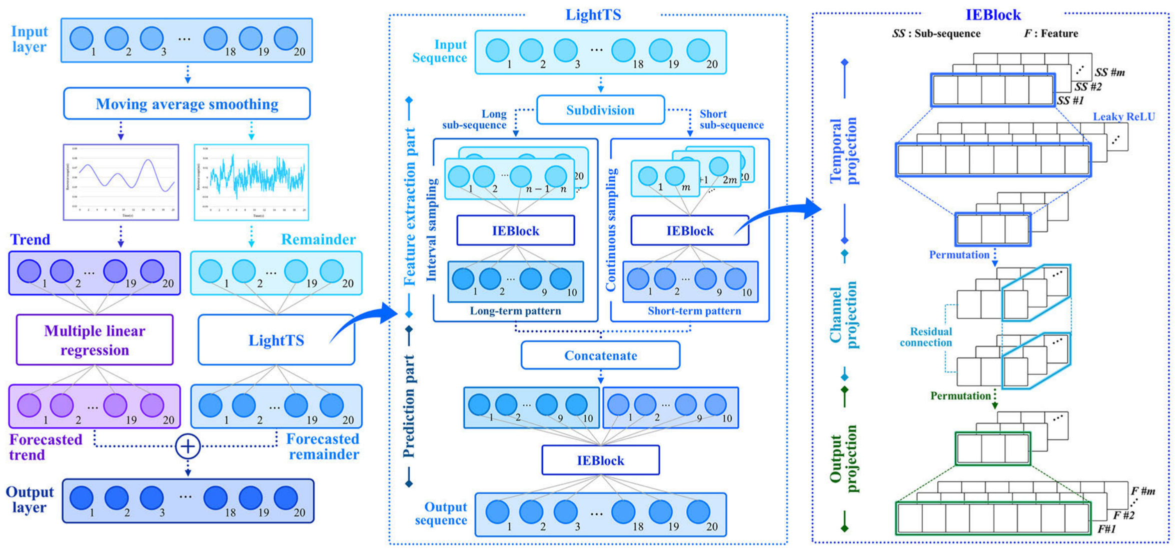


FIGURE 2. Architecture of time-series decomposition-based hybrid forecasting (DeHyFo) model.

values through the multiple linear regression layers and LightTS layers.

C. OPTIMAL DECISION SCHEDULER

The optimal decision scheduler step is composed of a feasible node selector, a forecasted resource evaluator, a conditional availability provider, and a decision executor for calculating the optimal scheduling decision. First, the feasible node selector filters the nodes that ensure service availability in the cluster by selecting the nodes whose N_{ut} is less than $\mathcal{u}$ after applying the C_{us} to be deployed, as shown in (3). At this point, FoRES records the containers that are related to the unfiltered node in L_{const} to execute the containers that are not deployed due to the burst workload and prediction error caused by the future resource forecaster step.

$$L_{filter} = \left\{ N \in L_{node} \mid \left(\frac{C_{us}}{N_{cap}} \times 100 + N_{ut} \right) < \mathcal{u} \right\} \quad (3)$$

The forecasted resource evaluator selects the node that ensures resource efficiency among the selected nodes by calculating the score based on R^t_{ut}. For this purpose, FoRES calculates the score of a node at each predicted time point t from the SERU function based on the ℓ and $\mathcal{u}$, as shown in (4).

$$\mathrm{SERU}\left(R^t_{ut}\right) = \begin{cases} (1-\mathcal{n})^{\frac{R^t_{ut}}{\ell}} + \mathcal{n} - 1, & \text{if } 0 < R^t_{ut} \leq \ell \\ \dfrac{-4\mathcal{m}}{(\mathcal{u}-\ell)^2}\left(R^t_{ut}-\ell\right)\left(R^t_{ut}-\mathcal{u}\right), & \text{if } \ell < R^t_{ut} \leq \mathcal{u} \end{cases} \quad (4)$$

In the case of $0 < R^t_{ut} \leq \ell$, where resource efficiency is low, the SERU function uses an exponential function to make negative scores as resource utilization decreases. Specifically, the function is adjusted so that the minimum score is $\mathcal{n}$ when R^t_{ut} is 0, and the maximum score is 0 when R^t_{ut} reaches ℓ. This condition applies when $\mathcal{n} < 0$. On the other hand, for $\ell < R^t_{ut} \leq \mathcal{u}$, which falls within the target range of resource utilization, the SERU function uses a quadratic function to make positive scores based on resource efficiency. It ensures the highest score, $\mathcal{m}$, when $R^t_{ut} = (\mathcal{u} + \ell)/2$, aligning with the optimal resource utilization threshold. As R^t_{ut} approaches ℓ, the resource efficiency decreases, whereas as it approaches $\mathcal{u}$, the risk of overloading due to potential prediction errors increases. Accordingly, FoRES configures the SERU function as a concave function, which decreases the score for the node as it approaches the two thresholds.

FoRES calculates the scores for each node using the SERU function, and then selects the node with the highest accumulated score S_{ut} as the scheduling target, as shown in Algorithm 1. Subsequently, FoRES requests the creation of a pod, which contains a container, on the selected node using SendCom() Here, T denotes the target node of scheduling decision.

In addition, FoRES acts as a conditional availability provider to provide a minimum life cycle to the container recorded in L_{const}, as shown in Algorithm 2. First, to prevent overload, FoRES calculates R_{ut} at each time point and identifies M_{over}, which is the time point before R^t_{ut} becomes higher than $\mathcal{u}$. If R^t_{ut} is below u, FoRES calculates the resource efficiency S_{ut} for each container using the SERU function and updates M_{over} to the current time point. If R^t_{ut} exceeds $\mathcal{u}$, FoRES uses the natural logarithm to calculate the time weight W_{time} of M_{over} to give higher priority to containers with longer life cycles. Subsequently, FoRES uses W_{time} and S_{ut} to calculate the scheduling expectancy V_{ex} for all container-node pairs. Finally, FoRES selects the container-node pair with the highest V_{ex} and uses SendCom() to

Algorithm 1 Forecasted Resource Evaluator

```
Input: Container $C$, $L_{filter}$
Output: Scheduling decision command
Initialize: Dictionary of scores calculated by scoring function $D_{score} \leftarrow \{\}$
1:  for $N$ in $L_{filter}$ do
2:      $S_{ut} \leftarrow 0$
3:      for $t$ in range(0, $C_{time}$) do
4:          $R^t_{ut} \leftarrow (C^t_{us}/N_{cap} \times 100 + N^t_{ut})$
5:          $S_{ut} \leftarrow S_{ut}+$ SERU($R^t_{ut}$)
6:      endfor
7:      $D_{score}[N] \leftarrow S_{ut}$
8:  endfor
9:  $T \leftarrow \max(D_{score}, key \leftarrow D_{score}.get)$
10: SendCom([$T$, $C$])
```

Algorithm 2 Conditional Availability Provider

```
Input: $L_{const}$, $L_{node}$
Output: Additional scheduling decision command
Initialize: Dictionary of expected values of container deployment $D_{value} \leftarrow \{\}$
1:  for $C$ in $L_{const}$ do
2:      for $N$ in $L_{node}$ do
3:          $S_{ut} \leftarrow 0$
4:          $M_{over} \leftarrow 0$
5:          for $t$ in range (0, $C_{time}$) do
6:              $R^t_{ut} \leftarrow (C^t_{us}/N_{cap} \times 100 + N^t_{ut})$
7:              if $R^t_{ut} < u$ then
8:                  $S_{ut} \leftarrow S_{ut}+$SERU($R^t_{ut}$)
9:                  $M_{over} \leftarrow t$
10:             else
11:                 break
12:             endif
13:         endfor
14:         if $M_{over} > 0$ then
15:             $W_{time} \leftarrow \log_e(M_{over})$
16:             $V_{ex} \leftarrow W_{time} \times S_{ut}$
17:             $D_{value}[(N, C)] \leftarrow V_{ex}$
18:         endif
19:     endfor
20: endfor
21: $T$, $C_{max} \leftarrow \max(D_{value}, key \leftarrow D_{value}.get)$
22: SendCom([$T$, $C_{max}$])
```

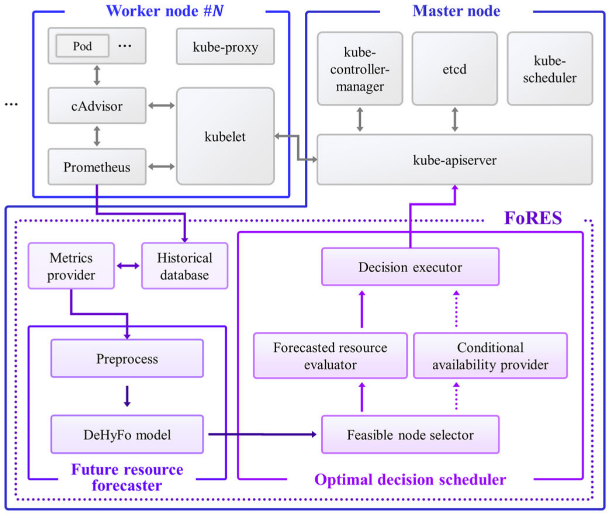


FIGURE 3. Overall architecture of the Kubernetes cluster with FoRES.

request the deployment of the container on the corresponding node.

To deploy the container to the nodes selected by the forecasted resource evaluator and the conditional availability provider, the decision executor modifies the container configuration file before deploying the container inside the cluster through the API daemon.

IV. FoRES IMPLEMENTATION

To ensure an optimized container deployment in a cloud computing environment, FoRES was added to the Kubernetes cluster. The Kubernetes cluster consists of one master node and six worker nodes, each with four CPU cores and 12 GB of memory, and was built using Kubernetes v1.23.1. Figure 3 shows the overall architecture of Kubernetes with the FoRES proposed in this study.

The Kubernetes cluster consists of one or more master nodes and worker nodes to manage the containers. First, the master node manages the pods in the cluster through a control plane that includes the kube-apiserver, the kube-scheduler, the kube-controller-manager, and etcd. A pod is a group of containers that includes one or more containers. The kube-apiserver controls requests from clients and components that use the Kubernetes API. The kube-scheduler selects the nodes where the pods are deployed according to various conditions and policies, such as resources and anti-affinity. The kube-controller-manager uses various controllers to manage the state of a cluster. The etcd is a distributed key-value repository that stores all the configuration data in the cluster.

A worker node deploys and runs pods, and it consists of a kubelet, a kube-proxy, container runtime software, and a container monitoring tool. The kubelet, which is the default agent on the worker node, communicates with the kube-apiserver on the master node to manage the pods in the node. The kube-proxy manages pod traffic according to the network policies. Container runtime software manages the life cycle of a container. In this study, we used Docker as the container runtime software. In addition, a container advisor (cAdvisor) and Prometheus were used as container monitoring tools to collect hourly system metrics for the containers and store them in the historical database in FoRES.

The proposed FoRES is intended to efficiently deploy containers, and it is located on the master node. It consists of a metrics provider, a future resource forecaster, and an optimal decision scheduler. First, the metrics provider extracts the CPU and memory metrics from the historical database that contains the operational data for the container to be deployed.

The future resource forecaster preprocesses the extracted resource metrics and then predicts future resource usage using the DeHyFo model. The optimal decision scheduler filters the deployable nodes based on the predicted results using the feasible node selector, and then it selects the optimized nodes using the forecasted resource evaluator. If a node is not selected by the feasible node selector, it selects the node that ensures the minimum life cycle of the container by the conditional availability provider. Finally, FoRES transmits a deployment command to the kube-apiserver of the master node to deploy the container to the node selected by the decision executor. The detailed implementation of FoRES is as follows.

A. METRICS PROVIDER

First, FoRES accessed the historical database that held the system metrics that were collected from cAdvisor and Prometheus on each worker node to extract the CPU and memory usage metrics for the containers in the scheduling queue. The extracted resource usage metrics were recorded at intervals of 15 s.

B. FUTURE RESOURCE FORECASTER

In the future resource forecaster step, FoRES converted the resource usage metrics into vectors and then performed min-max normalization to account for learning errors and to reconstruct the values of the CPU and memory usage vectors in the range of [0,1]. In addition, to train the prediction model with supervised learning, FoRES used the sliding window technique with a w of 1 to generate labels for each time point.

The DeHyFo model, which is a FoRES prediction model, consists of an input layer, a multiple linear regression layer, a LightTS layer, and an output layer. It is used to predict future resource usage with high accuracy. The lengths of the input and output sequences of the DeHyFo model were set to 20 to analyze the pattern of node state change cycles in Kubernetes. The multilinear regression layer consisted of one hidden layer with 40 cells to efficiently analyze the linear data trends. The LightTS layer divided the input sequence into subsequences with lengths of two and ten to accurately analyze the short- and long-term patterns of Re. In addition, the LightTS layer consisted of three IEBlocks, two of which were downsampled to extract the features of Re, and one IEBlock that analyzed long- and short-term patterns to predict future Re. At this point, the IEBlock used Leaky ReLU in temporal projection to prevent the vanishing gradient problem and to perform residual connection in the channel projection to prevent the loss of main feature information.

The DeHyFo model used the smooth-L1 technique as the loss function, which has the advantages of mean absolute error (MAE) and mean squared error (MSE). In addition, the DeHyFo model was set to 100 epochs, a 32 batch size, and a learning rate of 1e-4, and the weights were adjusted with an Adam optimizer to analyze and learn the resource usage patterns [20].

TABLE 2. Number of data for workload trace data in Alibaba2018, Alibaba2021, and Google2019 datasets.

No. data	Alibaba2018	Alibaba2021	Google2019
Total	45,850 (7 d)	2,880 (12 h)	178,540 (30 d)
Train	36,680	2,304	142,832
Validation	4,585	288	17,854
Test	4,585	288	17,854

C. OPTIMAL DECISION SCHEDULER

In the optimal decision scheduler step, FoRES set the ℓ and u, to 80% and 90%, respectively, to select nodes that would ensure resource efficiency and service availability [21], [22], [23]. In addition, the n and m, were set to -1 and 1, respectively. To modify the node information in the container configuration file according to the scheduling results, FoRES used yq, which is a command line interface (CLI) tool [24]. Finally, to deploy the modified container in the cluster, FoRES used kubectl to transmit the deployment command to the kube-apiserver.

V. PERFORMANCE EVALUATION

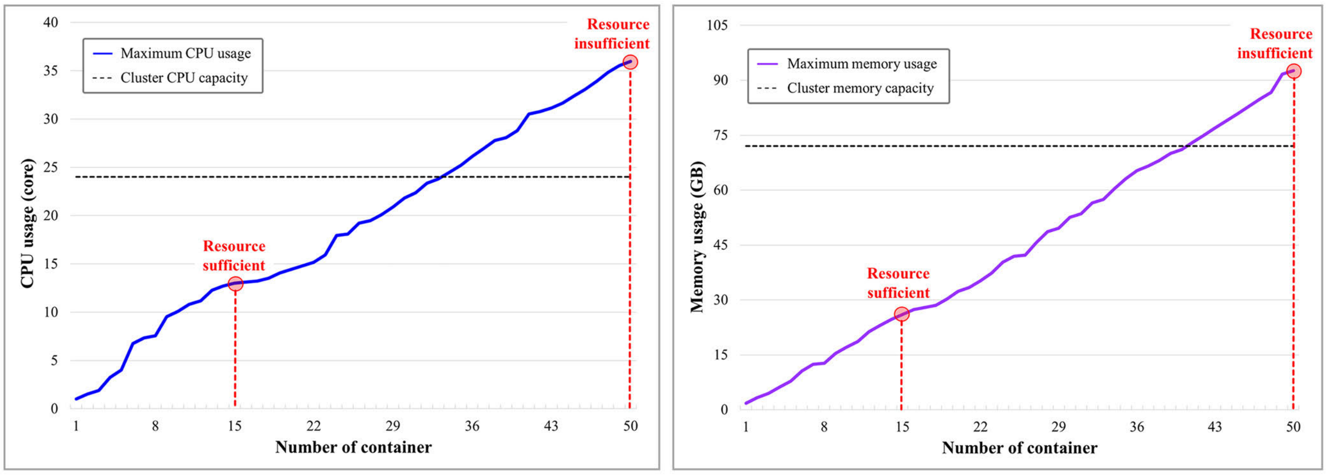
To evaluate the effectiveness of the proposed FoRES, we measured the prediction performance of the DeHyFo model and the scheduling performance of FoRES. To consider the different life cycles of the containers, we used container resource usage data that were recorded over 7 d, 12 h, and 30 d from the Alibaba cluster-trace-v2018 (Alibaba2018) [25], Alibaba cluster-trace-microservices-v2021 (Alibaba2021) [25], and Google Cluster Workload Traces 2019 (Google2019) [26] datasets, respectively. To evaluate the performance of FoRES on various workload patterns, we randomly selected 15 workload trace data from each dataset. In addition, the timestamps of all data were reconstructed to Prometheus' default scraping period of 15 s, and the training, validation, and testing datasets were configured as 80%, 10%, and 10%, respectively, as shown in Table 2.

To verify the prediction performance of the DeHyFo model, we compared its predictive accuracy with that of the GRU [27], Informer [28], Autoformer [29], DLinear [19], and LightTS models used in time-series data analysis. We also used the MAE, root mean squared error (RMSE), mean absolute percentage error (MAPE), and symmetric MAPE (SMAPE) as prediction performance evaluation metrics.

Table 3 lists the prediction performance evaluation results for each deep learning model. The GRU model did not consider the temporal features of the time-series data, which resulted in low prediction accuracy. However, the Informer and Autoformer models, which are based on Transformer, extracted and analyzed the temporal features of the time-series data, but their prediction performance was poor owing to their insufficient training data. The DLinear model considered temporal features through time-series decomposition but was unable to accurately predict Re that contained irregular noise. The LightTS model efficiently analyzed different temporal resolutions of the time series and predicted future

TABLE 3. Forecasting performance results for DeHyFo model and five other deep learning models.

Dataset	Metric	GRU	Informer	Autoformer	DLinear	LightTS	DeHyFo
Alibaba2018	MAE	0.000938	0.003974	0.001982	0.001240	0.000278	0.000184
	RMSE	0.001322	0.004685	0.002441	0.001623	0.000742	0.000632
	MAPE	5.855562	17.69699	14.28104	5.267718	1.963130	1.401165
	SMAPE	5.654279	17.38176	12.61321	5.060081	1.932038	1.372802
Alibaba2021	MAE	0.035454	0.047225	0.024982	0.009073	0.006020	0.005536
	RMSE	0.036900	0.049513	0.031214	0.011037	0.010191	0.009739
	MAPE	28.20831	22.20939	15.61043	7.948317	4.876077	4.725560
	SMAPE	17.84492	18.51818	16.88249	8.151550	4.695236	4.553116
Google2019	MAE	0.003388	0.005055	0.004205	0.003775	0.001472	0.001351
	RMSE	0.008722	0.009932	0.010856	0.009152	0.006129	0.005882
	MAPE	8.340069	12.99154	10.42671	9.055201	3.479430	3.252311
	SMAPE	7.939521	12.58774	9.566956	8.544638	3.366247	3.274959



(a) Maximum cumulative CPU usage from workload trace data (b) Maximum cumulative memory usage from workload trace data

FIGURE 4. Cumulative resource usage for cluster resource capacity and workload trace data.

resource usage with higher accuracy than other deep learning models. The DeHyFo model offered better prediction performance of future resource usage than the existing LightTS model because it analyzed *Tr* and *Re* separately using multiple linear regression and the LightTS model, both of which are time-series decomposition techniques.

To comprehensively measure the scheduling performance of FoRES, we used several test cases that accounted for the resource capacity of the cluster and the resource allocation of the container. To establish a baseline for the cluster's resource capacity, we first calculated the cumulative value of the maximum resource usage in the workload trace data, as shown in Figure 4. When more than 34 containers were deployed in the workload trace data, the CPU capacity of the cluster was insufficient. However, when 41 or more containers were deployed, the memory capacity of the cluster was insufficient. Therefore, the resource capacity scenario of the cluster was configured to be resource-sufficient when 15 containers were used for the workload trace data, and resource-insufficient when 50 containers were used.

Figure 5 shows example of the resource allocation thresholds for the over, adequate, and under cases that were defined in this study to represent different scenarios of container resource allocation. For the over case, the container request was set to 1.5 times the maximum C_{us}. In the adequate case, the container request was set to the maximum C_{us} that

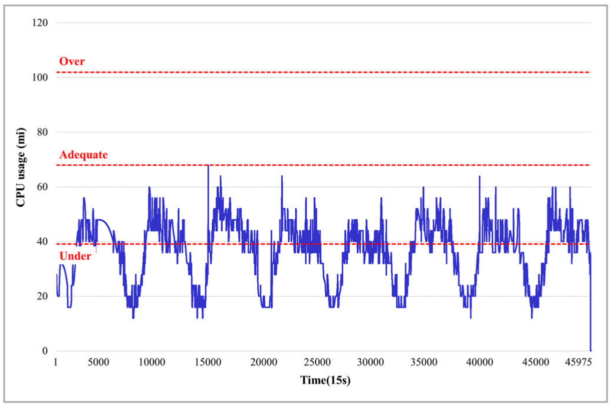


FIGURE 5. Resource request by resource allocation scenarios for CPU usage of c_7844 in Alibaba cluster-trace-v2018.

TABLE 4. **Test cases for the scheduling performance evaluation of FoRES.**

Case	Resource capacity scenarios of cluster		Resource allocation scenarios of container		
	Sufficient	Insufficient	Over	Adequate	Under
1	✓		✓		
2	✓			✓	
3	✓				✓
4		✓	✓		
5		✓		✓	
6		✓			✓

TABLE 5. **Average number of node overloads for each container scheduling technique for the test cases.**

Case	kube-scheduler	Trimaran	GLP-S	FoRES
1	0	0	0	0
2	0	0	3955	0
3	117	1	7905	0
4	0	0	2	2
5	0	0	4141	4
6	774	59	14127	4

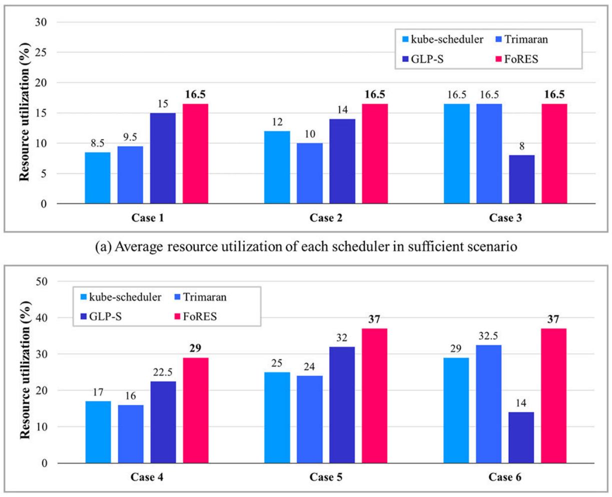
ensured the container runtime. Finally, for the under case, the container request was set to median of C_{us}.

Table 4 shows six test cases, combining two cluster resource capacity scenarios with three container resource allocation scenarios, to evaluate the scheduling performance of FoRES under various resource environments and allocation conditions.

To verify the scheduling performance of FoRES, we compared its resource efficiency, service reliability, and scheduling decision time with three container scheduling techniques that used heuristic algorithms and deep-learning models. The heuristic algorithm-based container scheduling techniques were Kubernetes' kube-scheduler and IBM's Trimaran [30]. Kube-scheduler evaluates all nodes within the Kubernetes cluster and deploys containers on the node with the highest score. Trimaran, extending kube-scheduler, aims for resource balancing. It scores nodes based on resource usage and then selects nodes with resource utilization below a certain threshold to deploy containers. For the deep learning-based container scheduling technique, we selected GLP-S [11]. In addition, this study used the resource consumer image of Kubernetes to create workloads for three types of datasets [31]. Because Kubernetes generally uses pods that consist of one container, we used singleton pods in this study [10], [31].

Figure 6 and Table 5 show the measurements for the average resource utilization of the cluster and the average number of node overloads for the four container scheduling techniques for each test case.

To calculate the average resource utilization (ARU) of a cluster, we used the average resource utilization of the CPU and memory, as shown in (5). Here, Cl_{time} and $Ru_{resource}$ denote the operation time and resource utilization of the cluster, respectively.



(a) Average resource utilization of each scheduler in sufficient scenario

(b) Average resource utilization of each scheduler in insufficient scenario

FIGURE 6. **Average resource utilization of a cluster using a container scheduling technique based on the cluster's resource capacity scenario.**

$$\text{ARU} = \frac{1}{Cl_{time}} \sum_{i=1}^{Cl_{time}} \frac{Ru^{i}_{CPU} + Ru^{i}_{Memory}}{2} \tag{5}$$

In the scenario in which the resource capacity of the cluster was sufficient, as shown in Figure 6(a), the maximum average resource usage of the cluster was measured as 16.5% when 15 containers having different life cycles were deployed using each container scheduling technique. In case 3, because kube-scheduler and Trimaran performed scheduling based on resource requests, low resource usage resulted in the creation of idle resources. GLP-S achieved higher resource utilization than the other heuristic-based container scheduling techniques in cases 1 and 2 because it considered the predicted resource usage and resource requests. However, as shown in Table 5, GLP-S experienced a significantly higher number of overloads than did the other techniques, owing to prediction errors. GLP-S had the lowest resource usage in case 3 because it did not consider scenarios in which resources were under-allocated. In contrast, FoRES achieved resource utilization that was up to 8.5% better than that of the other container scheduling techniques, with a maximum resource utilization of 16.5% for cases 1, 2, and 3, because it calculated scheduling decisions using a SERU function

TABLE 6. Comprehensive scheduling performance evaluation results for each container scheduling technique.

Case	Performance metric	kube-scheduler	Trimaran	GLP-S	FoRES
1	Normalized RC	1.21	1.21	1.22	1.22
	Resource efficiency	7.02	7.84	12.29	13.44
	CSR	0.73	0.73	0.86	1
	Service reliability	1.73	1.73	1.86	2
	Scheduling decision time (s)	0.0005	0.002	1.2	0.8
2	Normalized RC	1.22	1.22	1.22	1.22
	Resource efficiency	9.83	8.19	11.45	13.44
	CSR	0.86	0.86	0.26	1
	Service reliability	1.86	1.86	0.26	2
	Scheduling decision time (s)	0.0003	0.006	1.19	0.8
3	Normalized RC	1.22	1.22	1	1.22
	Resource efficiency	13.44	13.44	8	13.44
	CSR	0.86	0.93	0	1
	Service reliability	0.86	1.3	0	2
	Scheduling decision time (s)	0.0006	0.008	1.24	0.82
4	Normalized RC	1.56	1.78	1.85	2
	Resource efficiency	10.89	8.96	12.16	14.5
	CSR	0.4	0.38	0.5	0.68
	Service reliability	1.4	1.38	0.78	0.81
	Scheduling decision time (s)	0.0005	0.006	1.22	0.86
5	Normalized RC	1.4	1.37	1.69	1.84
	Resource efficiency	17.74	17.48	18.83	20.06
	CSR	0.5	0.48	0.22	0.84
	Service reliability	1.5	1.48	0.22	0.88
	Scheduling decision time (s)	0.0005	0.006	1.26	0.89
6	Normalized RC	1.63	1.69	1.41	1.84
	Resource efficiency	17.78	19.11	9.92	20.06
	CSR	0.54	0.7	0	0.84
	Service reliability	0.54	0.7	0	0.88
	Scheduling decision time (s)	0.0007	0.01	1.27	0.89

that used predicted resource usage. In addition, unlike other container scheduling techniques, FoRES did not cause node overload.

Scenarios with insufficient cluster resource capacity deployed more containers than did scenarios with sufficient resource capacity, which resulted in a higher average resource utilization for each container scheduling technique, as shown in Figure 6(b). In contrast to the kube-scheduler, Trimaran performed scheduling by considering resource balancing. Therefore, its average resource utilization was the lowest in cases 4 and 5. However, in case 6, Trimaran showed a higher average resource utilization than did the kube-scheduler because the kube-scheduler was an under-allocated resource, which resulted in a rapid increase in the number of node overloads. The average resource utilization of GLP-S was higher than that of the heuristic-based container scheduling techniques for cases 4 and 5, which were the same as the scenario having sufficient resource capacity. However, when resources were under-allocated, as in case 6, GLP-S showed the lowest average resource utilization. FoRES efficiently managed resources by improving the average resource utilization by up to 23% compared with other container scheduling techniques, even though some containers experienced overload, to ensure the minimum life cycle of the container.

Table 6 lists the resource cost (RC), resource efficiency, container success rate (CSR), service reliability, and scheduling decision time for the test cases using the container scheduling techniques. The scheduling decision time denotes the average amount of time required to calculate the scheduling decision per container.

In this study, we measured resource efficiency using (7), which takes as input the RC calculated in (6) and the ARU. The resource price $\mathcal{P}$ uses the cost policy from Amazon Web Services (AWS), and the value of RC is normalized to the range of [1], [2], and [32]. Here, C_{sche} denotes the list of scheduled containers, whereas $t_{resource}$ and $\psi_{resource}$ denote the resource utilization time and the amount of allocated resource, respectively.

$$\mathrm{RC} = \frac{1}{C_{sche}} \sum_{i=1}^{C_{sche}} \mathcal{P}_{CPU} \times t^{i}_{cpu} \times \psi^{i}_{CPU} + \mathcal{P}_{memory} \times t^{i}_{memory} \times \psi^{i}_{memory} \tag{6}$$

$$\text{Resource efficiency} = \frac{\mathrm{ARU}}{\mathrm{Norm(RC)}} \tag{7}$$

Furthermore, in this study, the CSR, which is the rate at which a container operates without downtime, was calculated using (8), and service reliability was determined from the CSR and the number of node overloads O_{num} as shown in (9). Here, C_{total} denotes the total number of containers that requested scheduling and C_{full} denotes the number of containers that run a full life cycle without overload.

$$\mathrm{CSR} = \frac{C_{full}}{C_{total}} \tag{8}$$

$$\text{Service reliability} = e^{-O_{num}} + \mathrm{CSR} \tag{9}$$

By using the predicted resource usage to select nodes that provide high resource utilization, FoRES achieved the highest resource efficiency of all the container scheduling techniques for all cases. However, FoRES experienced a greater number of overloads than did the kube-scheduler or Trimaran in cases 4 and 5, which resulted in lower service reliability, but FoRES achieved the highest SR and provided a better container runtime than did the other container scheduling techniques. The scheduling decision time of FoRES was longer than that of the heuristic-based container scheduler because FoRES predicted resource usage and then calculated the scheduling decision. However, the scheduling decision time of FoRES was generally shorter than that of GLP-S because the DeHyFo model that was built into FoRES was composed of an efficient MLP structure, unlike the AI-based TSF model.

VI. CONCLUSION

In this study we proposed FoRES, which ensures optimized container deployment to run services efficiently and reliably in a cloud computing environment. FoRES effectively analyzed the time-specific patterns of workload using the DeHyFo model, which consisted of time-series decomposition, multiple linear regression, and LightTS, and it predicted future resource usage with higher accuracy than did the existing TSF models. In addition, FoRES selected nodes to ensure service availability and resource efficiency using an optimal decision scheduler, and then it deployed the containers. When we evaluated the performance of FoRES against actual microservice benchmark datasets, we observed that it improved resource utilization by up to 23% compared with existing scheduling techniques, even when resources were under- or over-allocated, and it ensured service reliability by reducing the number of node overloads by up to 14,123 times.

However, FoRES cannot be applied to clusters in a cold start state because it requires the use of historical container data to predict future resource usage. Therefore, we intend to add an optimization algorithm-based scheduling technique to FoRES that uses the basic information of the containers.

ACKNOWLEDGMENT

(Jueun Jeon and Sihyun Park contributed equally to this work.)

REFERENCES

[1] I. Mavridis and H. Karatza, "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing," *Future Gener. Comput. Syst.*, vol. 94, pp. 674–696, May 2019.

[2] M. Sollfrank, F. Loch, S. Denteneer, and B. Vogel-Heuser, "Evaluating Docker for lightweight virtualization of distributed and time-sensitive applications in industrial automation," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3566–3576, May 2021.

[3] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surveys*, vol. 54, no. 10s, pp. 1–35, Sep. 2022.

[4] C. Carrión, "Kubernetes as a standard container orchestrator—A bibliometric analysis," *J. Grid Comput.*, vol. 20, no. 4, Dec. 2022, Art. no. 42.

[5] I. Ahmad, M. G. AlFailakawi, A. AlMutawa, and L. Alsalman, "Container scheduling techniques: A survey and assessment," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 7, pp. 3934–3947, Jul. 2022.

[6] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surveys*, vol. 55, no. 7, pp. 1–37, Jul. 2023.

[7] 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 Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1386–1401, Apr. 2022.

[8] X. Zhang, L. Li, Y. Wang, E. Chen, and L. Shou, "Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters," *IEEE Access*, vol. 9, pp. 105192–105204, 2021.

[9] W. Yang, X. Liao, D. Dong, and J. Yu, "Exploring job running path to predict runtime on multiple production supercomputers," *J. Parallel Distrib. Comput.*, vol. 175, pp. 109–120, May 2023.

[10] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–24, May 2020.

[11] Y. Yang and L. Chen, "Design of kubernetes scheduling strategy based on LSTM and grey model," in *Proc. IEEE 14th Int. Conf. Intell. Syst. Knowl. Eng. (ISKE)*, Dalian, China, Nov. 2019, pp. 701–707.

[12] J. Liu, S. Wang, A. Zhou, J. Xu, and F. Yang, "SLA-driven container consolidation with usage prediction for green cloud computing," *Frontiers Comput. Sci.*, vol. 14, no. 1, pp. 42–52, Feb. 2020.

[13] H. K. Mehta, P. Harvey, O. Rana, R. Buyya, and B. Varghese, "WattsApp: Power-aware container scheduling," in *Proc. IEEE/ACM 13th Int. Conf. Utility Cloud Comput. (UCC)*, Leicester, U.K., Dec. 2020, pp. 79–90.

[14] S. Tuli, S. S. Gill, M. Xu, P. Garraghan, R. Bahsoon, S. Dustdar, R. Sakellariou, O. Rana, R. Buyya, G. Casale, and N. R. Jennings, "HUNTER: AI based holistic resource management for sustainable cloud computing," *J. Syst. Softw.*, vol. 184, Feb. 2022, Art. no. 111124.

[15] M. Carvalho and D. F. Macedo, "Container scheduling in co-located environments using QoE awareness," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 3, pp. 3247–3260, Feb. 2023.

[16] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015.

[17] 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 Trans. Sustain. Comput.*, vol. 8, no. 3, pp. 375–384, Mar. 2023.

[18] T. Zhang, Y. Zhang, W. Cao, J. Bian, X. Yi, S. Zheng, and J. Li, "Less is more: Fast multivariate time series forecasting with light sampling-oriented MLP structures," 2022, *arXiv:2207.01186*.

[19] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" in *Proc. AAAI*, Washington DC, USA, 2023, pp. 11121–11128.

[20] J. Jeon, B. Jeong, S. Baek, and Y.-S. Jeong, "Hybrid malware detection based on bi-LSTM and SPP-net for smart IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 7, pp. 4830–4837, Jul. 2022.

[21] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, and L. Qi, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.

[22] S.-Y. Hsieh, C.-S. Liu, R. Buyya, and A. Y. Zomaya, "Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers," *J. Parallel Distrib. Comput.*, vol. 139, pp. 99–109, May 2020.

[23] B. Jeong, J. Jeon, and Y. S. Jeong, "Proactive resource autoscaling scheme based on SCINet for high-performance cloud computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 4, pp. 3497–3509, Dec. 2023.

[24] M. Farah. *Yq Command Line Interface*. Accessed: Mar. 27, 2023. [Online]. Available: https://github.com/mikefarah/yq

[25] Alibaba Group. (2021). *Alibaba-Cluster-Trace*. Accessed: Mar. 3, 2023. [Online]. Available: https://github.com/alibaba/clusterdata

[26] (2019). *Google Cluster Workload Traces*. Accessed: Mar. 8, 2023. [Online]. Available: https://github.com/google/cluster-data

[27] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," 2014, *arXiv:1409.1259*.

[28] H. Zhou, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI*, 2021, pp. 11106–11115.

[29] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," in *Proc. NeurIPS*, 2021, pp. 22419–22430.

[30] (2020). *Trimaran*. Accessed: Feb. 2, 2023. [Online]. Available: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/61-Trimaran-real-load-aware-scheduling

[31] B. Jeong, S. Baek, S. Park, J. Jeon, and Y.-S. Jeong, "Stable and efficient resource management using deep neural network on cloud computing," *Neurocomputing*, vol. 521, pp. 99–112, Feb. 2023.

[32] *AWS Fargate Pricing*. Accessed: Apr. 2, 2023. [Online]. Available: https://aws.amazon.com/ko/fargate/pricing/

JUEUN JEON received the B.S., M.S., Ph.D. degrees in multimedia engineering from Dongguk University, Seoul, Republic of Korea, in 2018, 2020, and 2024, respectively.

She is currently a Postdoctoral Researcher with the Department of AI·SW Convergence, Dongguk University, Seoul. Her current research interests include cloud computing and information security for the Internet of Things.

SIHYUN PARK received the B.S. and M.S. degrees in multimedia engineering from Dongguk University, Seoul, Republic of Korea, in 2022 and 2023, respectively.

His current research interests include cloud computing, the Internet of Things, and edge computing.

BYEONGHUI JEONG received the B.S. degree in computer science and engineering from Kongju National University, Cheonan, Republic of Korea, in 2021, and the M.S. degree in multimedia engineering from Dongguk University, Seoul, Republic of Korea, in 2023, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Artificial Intelligence.

His current research interests include cloud computing and information security for the Internet of Things.

YOUNG-SIK JEONG (Member, IEEE) received the B.S. degree in mathematics and the M.S. and Ph.D. degrees in computer science and engineering from Korea University, Seoul, Republic of Korea, in 1987, 1989, and 1993, respectively.

He was a Professor with the Department of Computer Engineering, Wonkwang University, Republic of Korea, from 1993 to 2012. He worked and conducted research with Michigan State University and Wayne State University as a Visiting Professor, in 1997 and 2004, respectively. He is currently with the Department of AI·SW Convergence, Dongguk University, Seoul. His research interests include cloud computing and information security for the Internet of Things.