Intelligent and Agile Control of Edge Resources for Latency-Sensitive IoT Services

VED P. KAFLE, (Senior Member, IEEE),
AND ABU HENA AL MUKTADIR, (Senior Member, IEEE)
National Institute of Information and Communications Technology, Tokyo 184-8795, Japan

Corresponding author: Ved P. Kafle (kafle@nict.go.jp)

ABSTRACT This paper presents an intelligent and agile resource control scheme for a latency-sensitive virtual network function (VNF) of Internet of things directory service (IoT-DS) deployed in a virtualized edge cloud whose computational and networking resources can be adjusted dynamically. The objective of the proposed scheme is to adjust resources dynamically such that the IoT-DS function can resolve IoT queries and provide IoT records within a bounded delay for latency-sensitive services such as automated driving, despite fluctuations in workloads. The proposed scheme leverages multiple regression models for resource demand prediction and dynamic adjustment. These models are trained offline before their deployment with a large training dataset collected from the system operating with simulated workloads. After the deployment, they are updated regularly by online retraining for more accurate performances. We aim to optimize resource allocation to satisfy both the target performance in terms of service latency and resource utilization. The results obtained from an experimental system implementation of the IoT-DS function in Docker containers show that the dynamic adjustment of CPU resources by the proposed scheme with supervised offline training reduces the CPU resource demand by 21.9% and the number of lookup latency requirement violations by 58.2% in comparison with a threshold rule-based conventional algorithm. Moreover, the proposed scheme can offer an agile control of CPU resources within a 1 s interval, which is five times faster than those reported in previous studies. The addition of unsupervised online retraining further reduces CPU resource requirements by 52% and lookup latency requirement violation cases by 62.5% compared with when no adjustments are performed.

INDEX TERMS Intelligent network control, IoT, virtual network function, agile network management, machine learning.

I. INTRODUCTION New network systems such as the fifth generation (5G) mobile networks are configured on network function virtualization (NFV) infrastructures with software-defined networking (SDN) control mechanisms. Software-controllable NFV infrastructures allow the deployment and operation of virtual network functions (VNFs) in virtual machines and containers, whose virtualized computational (i.e., CPU and memory), storage (i.e., magnetic or solid-state disk space), and networking (i.e., bandwidth) resources can be dynamically adjusted to satisfy the changing demand of VNFs. The resource requirement of a VNF increases with the workload. If the required amount of resources is not allocated to the VNF by dynamic adjustment, its task may not be able to be completed within the desirable latency of the service offered by the VNF.

VNFs placed in a distant cloud computing infrastructure may not be able to satisfy the service latency requirements because of the large communication latency incurred by the distance between end-users and the cloud computing infrastructure. To reduce the communication latency of cloud computing facilities, edge clouds or micro-datacenters [1] are deployed in the proximity of end-users. Edge clouds have relatively fewer resources, but can satisfy the low latency requirements of latency-sensitive applications, such as telemedicine, augmented reality, multiplayer online games, and network-controlled automated driving. Software-defined machine-to-machine communications are considered beneficial for cost reduction, fine granularity...
resource allocation, and end-to-end quality of service guarantee [2]. To optimally utilize edge resources and satisfy quality of service (QoS) requirements, a mechanism for the dynamic and fine-grained adjustment of computational, storage and networking resources allocated to VNFs is required. This necessity becomes stringent when multiple services deployed in an edge cloud compete for the available limited resources. Therefore, we propose a scheme for the intelligent and agile control of resources for latency-sensitive IoT service functions deployed in edge clouds.

In the cloud infrastructure, virtual machines (VMs) and container platforms are used to virtualize the available computational and networking physical resources. VNFs are installed and operated in virtualized resources, which can be elastically scaled up or down in response to the continuously changing resource demand as the workload fluctuates, with the objectives of improving resource utilization and satisfying the QoS. Elastic virtual resources can be scaled dynamically via two approaches: horizontally and vertically [3]. In horizontal scaling, the number of VMs or containers allocated to a service is increased or decreased dynamically according to the workload. In vertical scaling, the resources allocated to a single VM or container are adjusted slightly, without increasing the number of VMs and containers. Horizontal scaling is considered as a coarse-grained resource management approach that may often waste resources owing to the underutilization of some VMs or containers. Therefore, for the optimal utilization of limited resources of edge clouds, vertical scaling methods have been attempted before executing the horizontal scaling method [3], [4].

This paper presents a dynamic resource adjustment scheme utilizing the vertical scaling of resources allocated to containerized VNFs of latency-sensitive applications. Here, the latency sensitivity implies that the VNF has an upper bound on its task execution latency, beyond which users may perceive the service as unacceptable or useless. For example, if an automated driving control system does not complete its task within a specified latency threshold, users will perceive the service as useless.

We used a machine learning (ML)-based dynamic resource adjustment scheme employing both offline training and online retraining, in which three target performance requirements are satisfied simultaneously: i) service latency, ii) resource utilization, and iii) completion of performance data collection, resource adjustment decision making, and decision execution within a time interval of 1 s. In prior studies, using the threshold rule-based algorithms, the dynamic resource adjustment of containerized VNFs was executed in realtime [5]–[7]. However, the collection of data related with resource utilization and performance in these studies required a longer time than our target value. For example, 30, 5, and 4 s were required in the studies presented in [1], [5], and [6], respectively. To the best of our knowledge, no prior ML-based work has simultaneously addressed the three aforementioned requirements. The preliminary version of this study, which employed the offline training of ML models, was published in [10].

The resource adjustment scheme leverages multiple regression models for the prediction of resource demands and the dynamic adjustment. The regression models are trained offline before their deployment using a large dataset collected from the system by operating it in a simulated environment. After deployment, the regression models are updated regularly by retraining them online with a dataset collected from the operating system. The objective is to optimize resource allocation by dynamically adjusting the models within 1 s such that they can simultaneously satisfy the three requirements listed earlier, despite fluctuations in workloads. As a use case of latency-sensitive VNFs, we selected an Internet-of-things directory service (IoT-DS) function [9], which can store millions of name records of IoT devices and provide a fast lookup service with a latency of a few milliseconds.

The experimental results obtained from the implementation of the IoT-DS function show that the dynamic adjustment of CPU resources by the proposed scheme with supervised offline training reduced CPU resource requirements by 21.9% and the number of lookup latency requirement violations by 58.2% compared with a threshold rule-based conventional algorithm [7]. Moreover, this scheme can provide an agile control of CPU scaling within a 1 s interval, which is five times faster than that of a previous ML-based related study [1]. Furthermore, the dynamic CPU resource adjustment results obtained by additionally retraining the models online reduced CPU resource requirements by as much as 52% and the lookup latency requirement violations by 62.5% compared with when no adjustments were performed.

The remainder of this paper is organized as follows. Section II provides a literature review of regression-based resource management approaches. The proposed multiple regression-based resource adjustment scheme is presented in Section III. The experimental setup and training dataset preparation methods are described in Section IV. Section V presents the results of the CPU resource requirement prediction accuracy and dynamic CPU resource adjustment. Finally, Section VI concludes this paper.

II. RELATED WORK

Virtual resource adjustment techniques related to our work can be classified into two categories: 1) Threshold rule-based heuristic, and 2) regression-analysis-based adaptive. We thoroughly reviewed the prior studies that belong to these two categories. Table 1 shows a comparison of this work’s contribution with those of [1], [5]–[7], with regard to the techniques used (i.e., rule-based or ML), time interval between two successive decisions, maximum value of CPU resource utilization threshold, and scaling type (i.e., horizontal or vertical).

Various methods for the dynamic resource adjustment of containerized VNFs using threshold rule-based algorithms are presented in [5]–[7]. In [5], a threshold rule-based autoscaling method is presented for adjusting resources allocated to a web server VNF in an interval of 30 s by injecting
TABLE 1. Comparison of this work with related works.

| Ref. # | Rule/ML | Interval | Max Util. | Scaling  |
|--------|---------|----------|-----------|----------|
| [5]    | Rule    | 30 sec   | 80%       | Horizontal |
| [7]    | Rule    | 1 sec    | 80%       | Vertical  |
| [6]    | Rule    | 4 sec    | 90%       | Vertical  |
| [1]    | ML      | 5 sec    | 70%       | Horizontal |
| This work | ML      | 1 sec    | 90%       | Vertical  |

workloads in different patterns (using the httpperf tool). Another rule-based method for vertical scaling in every 4 s of Docker containers serving Graylog server applications is presented [6]. Similarly, the implementation of threshold rule-based dynamic vertical scaling of Docker containers at every second is presented in [7]. Heuristic algorithms are proposed in [8] to solve the delay minimization problem to minimize the average delay of task offloading in vehicular mobile edge computing environments.

Various methods for dynamic virtual resource management using regression techniques have been studied in [1], [11]–[18]. Among them, the decision tree, k-nearest neighbor, gradient boosting, and linear regression techniques have been employed to predict the latency for service-level agreement violation of CPU-intensive database applications and allocate the appropriate amount of resources [11]. The lasso regression model with regularization was used to predict server workloads in [13]. In [12], autoregressive models for CPU resource demand prediction and allocation were suggested to virtualized servers operating in enterprise datacenters. In [14], linear regression, artificial neural network (ANN) and support vector regression models were used to control the CPU, memory and storage resources of Docker containers for hosting Spark applications. In [15], a random forest (RF) regression model was used to select the best VM instances based on the workload and user goals. Time series regression with linear, support vector, ANN, and locally weighted regression models were suggested in [18] to predict the possible occurrence of heavy workloads (hotspot) and trigger VM migration in advance to avoid service disruptions in the hotspot. In [16], the performances of five regression models (linear, logistic, decision tree, bagging, and Gaussian process) were compared in the prediction of CPU demand patterns of software tasks executing on a self-driving vehicle. In [1], Gaussian process-based online workload and latency prediction model was presented to adjust the number of CPU cores of Docker containers hosting latency-sensitive applications. Similarly, the performance of linear and k-nearest neighbor regression models to predict future CPU and memory utilization, and accordingly, make decisions on energy-aware VM consolidation were reported in [17].

Both offline [1], [11]–[18], and online [19] ML training approaches have been employed for dynamic resource adjustment purposes. Offline training with a large dataset and high prediction accuracy from the beginning of the system operation are two advantages of using the supervised learning approach. However, this approach may be affected by the less accurate prediction of unknown input patterns. The preparation of all possible patterns of the training dataset is laborious. Unsupervised online ML training approaches are desirable for reducing human involvement in the laborious training phase and improving the prediction accuracy of unseen input patterns. For example, the unsupervised reinforcement learning technique presented in [19] has been applied to dynamically adjust the window size of multipath TCP to avoid congestion. An initial learning period without any TCP window adjustment was considered in [19] to configure the parameters of the initial window adjustment rules. By contrast, our scheme can learn and adjust resources dynamically from the beginning.

For the evaluation of resource adjustment schemes presented in previous studies [11]–[18], resource allocation and utilization data collected offline (e.g., Google cluster trace) were used, whereas the data collected in real time from the IoT-DS experimental system were used for the evaluation of this study.

III. DYNAMIC RESOURCE ADJUSTMENT SCHEME

A. SYSTEM MODEL

A simple diagram of the system model is shown in Fig. 1. An edge cloud hosts N Docker containerized VNFs in its physical server. Each of the VNFs belongs to either latency-sensitive, mission-critical applications or latency-tolerant batch processing applications (e.g., scientific computation task without strict completion deadlines). Both the VNFs of latency-sensitive and latency-tolerant applications may reside in the same physical server and compete for available computational and bandwidth resources. The VNFs of latency-sensitive applications have a higher priority for resources than the VNFs of latency-tolerant applications. The system model shown in Fig. 1 comprises three components: VNFs, end users (EUs), and resource controller (RC). The EUs are client devices that send service requests to the VNFs. The RC hosts the proposed resource control
algorithm, continuously monitors the resource allocation and utilization status of each VNF, predicts the resource demand, and executes resource adjustment decisions through underlying platform interfaces (e.g., the ‘docker update’ commands [20]). To assess the system performance, we measured the two-way latency observed by the EUs of latency-sensitive VNFs, i.e., the duration from the instant an EU submits a service request to the instant it receives the service response. This latency includes the two-way communication latency between the EU and VNF.

B. RESOURCE ADJUSTMENT SCHEME

As mentioned earlier, the proposed scheme operates in both offline and online training modes. In the offline training mode, it first performs offline data collection and trains the ML models before they are deployed. In the online training mode, the models are retrained online after they are deployed in the system. Namely, the models are continuously retrained using real system data and then applied for dynamic resource adjustment simultaneously. Next, we describe both the offline and online training modes of operation.

The following offline training operation steps were executed sequentially:

- Monitoring and performance data collection: The workload, resource allocation and utilization data (i.e., as feature values of the training dataset) were collected every second for a predetermined duration (e.g., a few hours).
- Data preparation: The most relevant features according to the correlation coefficient of the target variable were selected. Subsequently, the dataset was categorized into two groups, i.e., training and test datasets.
- Model training: The regression algorithms were trained using the training dataset and tuning hyperparameters. The prediction accuracy of the trained models was then evaluated using the test dataset. The trained models were ranked according to their prediction accuracy and training time.

After the above steps were completed, the trained ML models exhibiting the lowest prediction errors were deployed.

Fig. 2 shows the flowchart of the deployment and simultaneous online retraining operational stages. The deployment stages are indicated by a dashed red rectangle on the left, whereas the online retraining stages are indicated by a dashed green rectangle on the right. In the deployment stage, the currently observed workload is input to the best-trained ML models to determine the required amount of resources to satisfy the target service latency and resource utilization requirements. The decision is executed by either increasing or decreasing the amount of allocated resources. The latest data of observed workload, resource allocation, utilization, and performance are then added to the training dataset. When Z observations are added to the dataset, w% of them are copied to create a new training dataset. The training dataset size is retained such that a deterministic online training time is ensured for the models.

The ML models were retrained using the new training dataset. The retrained models, which were saved as a file, replaced the old ones at the beginning of the next workload observation time slot. If the workload variation is significant, then a small value of Z is preferred, and vice versa. This online retraining approach is suitable for addressing unknown future workload variations. It is noteworthy that multiple models can be retrained, and among them, we can only select the best model or create an ensemble of a few top models to improve the accuracy of the resource adjustment mechanism.

The differences between the offline and online training modes of the proposed scheme are listed in Table 2. In the offline training mode, training data collection is laborious as it requires many hours of observations and data collection with different sets of parameters adjusted manually in the experimental system. In online training, we started with a small number of training datasets compared with offline training. To implement online training, we have to create an ML model with only two observations in the dataset corresponding to the no-workload case. For example, if a k nearest neighbor model is used, then k + 1 observations should be performed initially. In the online training mode, ML models are retrained regularly in a fixed interval with Z new observation data, whereas in the offline training mode, the ML models can be retrained only on-demand or when no workload exists for a certain duration.

### Table 2. Comparison of offline and online training modes in terms of training data size and retraining interval.

| Criteria               | Offline training | Online training |
|------------------------|------------------|-----------------|
| Training data size     | Huge             | Very little     |
| Retraining interval    | On-demand        | In fixed intervals |
C. MULTIPLE REGRESSION MODEL
A multiple regression model that accepts a vector of independent variables as an input and yields the predicted value of a dependent variable as an output can be defined as shown in Eq. (1).

\[ y_i^p = f(x_i), \quad i = 1, 2, \cdots, M \]  

where \( y_i^p > 0 \) is the new amount of resource to be allocated to the container hosting a VNF such that the VNF’s performance can be maintained at a desired level despite varying workloads, \( x_i \) is the input vector comprising the observed workload, latency, resource allocation, and utilization parameters, and \( f() \) is an unknown mapping/fitting function to be derived by a suitable multiple regression algorithm that minimizes the error function. \( y_i^p \) is nonzero because a VNF uses a small amount of CPU resource even when no workload exists from the end-users. It is noteworthy that although we could observe more than \( M \) independent variables, we included only \( M \) highly correlated (e.g., Pearson coefficient) variables in input vector \( x_i \). To train the regression models, we utilized the \( M \) observed \( x_i \) values. Moreover, the \( x_i \) values related to resource utilization and latency can be changed when predicting the required resource utilization.

D. ALGORITHM SELECTION CRITERIA
A multiple regression algorithm should satisfy the following criteria for the selection of a suitable \( f() \) using Eq. (1):

1) Its bias and variance should be low such that it can minimize the prediction error.
2) It should avoid underfitting as much as possible such that it can reduce latency requirement violations. If the allocated resource is lower than the demand, then the worst-case CPU utilization is 100\% and, consequently, the latency increases.
3) The training and decision procedure should not take a long time so that 1) resource adjustment decisions can be updated every second, and 2) model retraining with updated training data can be completed within 1 s (an ideal case).

Since regression algorithms can be categorized as non-parametric, parametric, and ensemble, we included at least one algorithm from each category in our evaluation. We evaluated 10 regression algorithms using our offline training data. Based on the lowest prediction error obtained from the test dataset, we employed either one of the following four algorithms as \( f() \) in our regression model:

- Linear regression (LR) (parametric),
- \( k \)-nearest neighbor regression (\( k \)NNR) (non-parametric),
- Gradient boosting regression (GBR) and extremely randomized tree regression (ETR) (ensemble).

LR is a simple algorithm that attempts to minimize the ordinary least-squared error. \( k \)NNR calculates the output of a prediction variable from similar \( k \) observations in the training data. Both LR and \( k \)NNR demonstrate excellent performances with both small and large numbers of training datasets while requiring a short training time. GBR begins with a weak learner algorithm and iteratively optimizes a cost function over the training dataset. ETR is similar to the RF algorithm; only the tree stopping criterion differs from that of the RF [23]. For a detailed description of \( k \)NNR, LR, GBR, and ETR, the reader is referred to [22], [23].

IV. EXPERIMENTAL SYSTEM AND PARAMETERS
A. IoT-DS TESTBED CONFIGURATION
The IoT-DS function was regarded as a latency-sensitive VNF [21] to validate and evaluate the performance of the proposed resource control scheme through the experiment. The IoT-DS is used to store IoT device’s profile information (such as device name, ID, network address, location, owner’s name, data types they generate, security keys and certificates) as records in its database. By allocating enough computing resources through the dynamic control scheme, IoT-DS function can provide a fast lookup service for retrieving records with a low latency of a few milliseconds to the EU hosting IoT client applications [9]. As shown in Fig. 3, the IoT-DS experimental system comprises three virtual machines, of which two VMs serve as a resource controller (RC) and an EU that sends the record lookup queries, whereas the third VM contains a Docker containerized IoT-DS function with 100 thousand records stored in its database. The VMs were created using VirtualBox software on a Windows 8.1 host PC with an Intel Core i7-5930K 12 core CPU (3.5 GHz) and 64 GB of memory. The front-end of the IoT-DS received and processed a record lookup query from the EU (sent at time \( \theta_1 \)) and returned a response after the query was processed from the backend database. The response was received by the EU at time \( \theta_2 (\theta_2 > \theta_1) \). In the experiment, we incremented the latency requirement violation counter by one if the time difference \( (\theta_2 - \theta_1) \) exceeded the desirable tolerable latency of 8 ms. We exclusively allocated a CPU core to the Docker container that implemented the front-end function. The cycles of this CPU were dynamically allocated based on resource adjustment decisions made in accordance to the proposed scheme.

B. WORKLOAD PATTERNS
We generated three patterns of record lookup workloads: proportional, step, and Poisson. Namely, we sent the record lookup request queries from the EU by changing the query frequencies (i.e., requests per second, RPS) in these patterns.

- The proportional workload pattern, which is used for collecting the offline training dataset, is defined as shown in Eq. (2).

\[ \text{RPS}(t_1) = m_1 \times t_1 \]  

where \( m_1 \) is the ratio of the maximum RPS to the total duration of sending queries (\( D \)), and \( t_1 = 1, 2, \cdots, D \). We prefer the proportional workload pattern for training data collection because it allows us to obtain the CPU resource allocation and utilization data for the entire
The Poisson workload pattern was used for predicting the step workload pattern is intended for both online and offline resource adjustments. The pattern is defined by Eq. (3).

$$\text{RPS}(t_2) = \begin{cases} 200 & \text{for } 1 \leq t_2 \leq 20 \\ 400 & \text{for } 21 \leq t_2 \leq 40 \\ 600 & \text{for } 41 \leq t_2 \leq 60 \\ 800 & \text{for } 61 \leq t_2 \leq 80 \\ 1000 & \text{for } 81 \leq t_2 \leq 100 \end{cases}$$

(3)

where the RPS is constant for a defined duration of time $t_2$, where $t_2 = 1, 2, \cdots, 100$. We selected the step workload pattern for emulating abrupt changes in the workload for which dynamic resource adjustment would be challenging, and a threshold rule-based approach is found to be less responsive to such abrupt changes in the input values [7].

- The step workload pattern was intended for both online and offline resource adjustments. The pattern is defined by Eq. (3).

- The Poisson workload pattern was used for predicting resource utilization and adjusting the online retraining mode of our scheme. The Poisson workload pattern is defined by Eq. (4).

$$\text{RPS}(t_3) = \frac{(\lambda t_3)^n}{n!} \times e^{-\lambda t_3},$$

(4)

where $t_3$ is used to define the interval 0 to $t_3$, $n = \text{RPS}_{\text{avg}} \times t_3$ is the total number of arrivals in the interval 0 to $t_3$, and $\lambda = 0, 1, 2, \cdots$. The Poisson workload pattern was selected to study the gradual improvement in the prediction accuracy of an ML model using the proposed online retraining approach.

### TABLE 3. Description of observation variables of input vector $x_7$.

| Variable | Description |
|----------|-------------|
| $x_0$   | Timestamp for debugging |
| $x_1$   | RPS from EU to the front-end, max value is 2000 |
| $x_2$   | Queries per second (QPS) sent from front-end |
| $x_3$   | Requests in queue (RQ), max queue size is 1000 |
| $x_4$   | Average lookup latency in ms, < 8ms desirable |
| $x_5$   | Average database search latency in ms, $x_5 < x_4$ |
| $x_6$   | Average CPU utilization normalized to [0,1] |
| $x_7$   | Allocated CPU quota value normalized to [0,1] |

### C. OFFLINE TRAINING DATA COLLECTION AND PREDICTION

Offline training data were collected by performing the following seven steps:

- Step 1: Begin
- Step 2: Set CPU quota to 10% \(^2\)
- Step 3: Send queries from EU based on the proportional workload patterns
- Step 4: Record the values of eight observed variables $x_i, i = 0, 1, \cdots, 7$ as defined in Table 3
- Step 5: Increment CPU quota by 10%
- Step 6: If the allocated CPU quota $\leq$ 100%, proceed to Step 2, else Finish.
- Step 7: Finish

We recorded 2184 observations as the training dataset. The eight recorded variables from each observation are described in Table 3. It is noteworthy that $x_7$ is a dependent variable, whereas $x_i, i = 1, 2, \cdots, 6$ are independent variables included in the input vector. At this point, a question may arise: How many observations are adequate for training purposes? In general, the prediction accuracy improves as training data volume is increased at the expense of increased training time. However, a large training dataset may be problematic when the retraining of regression models must be completed within a short time in a dynamic, agile resource adjustment scenario. Therefore, we should select the size of the training dataset based on the tradeoff between the available training time and the desired prediction accuracy.

The trained regression models were saved in a file. To predict the required value of the CPU quota for allocation at instant $t$, the input vector comprising four observed features $x_i, i = 1, 2, 3, 5$ and two desired features, $x_4 < 8$ and $x_6 < 100\%$, were prepared at time ($t-1$). The trained models were input with these data to predict the CPU resource demand to achieve the desired utilization and service quality, i.e., the lookup latency.

### D. ONLINE RETRAINING AND PREDICTION

For online retraining, we collected $x_i, i = 0, 1, 2, \cdots, 7$ every second from the monitoring of the IoT-DS function and stored them as the training dataset. We prepared an input

\(^2\)The default Docker CPU period was 100 $\mu$s, represented by 100K. To allocate 10% of CPU cycles, the CPU quota value was set to 10K in the ‘docker update’ command [20].
vector comprising three observed features \( x_i, i = 1, 2, 3 \) at time \((t-1)\), and used the online retrained model to predict the CPU quota for allocation to maintain the average lookup latency below the given threshold, and improve the CPU utilization as much as possible.

V. EXPERIMENTAL RESULTS

For both the offline and online training modes, we evaluated the prediction accuracy, latency requirements violations, and resource utilization/allocation performances. For the offline training mode, the following metrics were evaluated:

- First, we defined an error metric for the prediction and investigated the effect of the desired \( x_4 \) and \( x_6 \) feature values of the input vector to obtain the lowest prediction error. The predictions (by trained regression models) were compared to actual CPU usage values obtained from the experimental IoT-DS system.
- Second, we compared the dynamic resource adjustment performance of the proposed model with a conventional threshold-rule-based algorithm [7] in terms of CPU resource demands and latency requirement violations. Similarly, for the online retraining mode, we evaluated the following two metrics:

  - First, we investigated how the prediction performance of ML models improved with online retraining in terms of the previously defined error metric. The predicted values obtained from the online retrained regression models were compared with the actual CPU usage values of the experimental IoT-DS system.
  - Second, we compared the dynamic resource adjustment performance of the proposed model in terms of the reduction in additional CPU resource allocation and latency requirement violations. The reference point for comparison was the case when 100% of the CPU cycles was allocated to the target VNF without performing any resource adjustment.

Next, we present the results of CPU resource adjustment by assigning the sufficient amount of storage and bandwidth resources to Docker containers such that the performance impacts were purely induced by the demand of CPU resources due to the variation in workloads.

A. EVALUATION METRICS

The prediction accuracy of trained regression models can be represented by the mean absolute error (MAE), which is defined as shown in Eq. (5).

\[
\text{MAE} = \frac{1}{X} \sum_{t=1}^{X} |y_t^p - y_t^a|, \tag{5}
\]

where \( X \) is the number of observations; \( y_t^p \) and \( y_t^a \) are the predicted value and ground truth or actual observed value, respectively. The MAE is considered as a better metric than the root mean squared error for the representation of prediction accuracy [24].

| Algorithm | MAE | Training time (ms) |
|-----------|-----|--------------------|
| kNNR      | 0.184 | 8.46 |
| LR        | 0.078 | 7.53 |
| GBR       | 0.065 | 3900.8 |
| ETR       | 0.058 | 1387.11 |

The performance of the proposed scheme was evaluated and compared with the conventional algorithm (denoted by Conv. in graphs) [7] in terms of the following two metrics:

- Cumulative amount of CPU quota allocated in addition to the fixed initial value of the minimal CPU quota \( q \) (which was set as 20K). If \( q_t \geq q \) denotes the CPU quota allocated at the \( t \)th observation second, where \( t = \{0,1,2,...,X\} \), then the cumulative amount of allocated CPU quota for algorithm \( Q_{algo} \) is \( \sum_{t=0}^{T}(q_t - q) \), where \( algo \in \{\text{Conv., kNNR, LR, GBR, ETR}\} \).
- The number of latency requirement violations, denoted by \( N_{algo} \), was measured as the cumulative number of seconds in which the average lookup latency exceeded the desired threshold value of 8 ms.

B. PREDICTION ERRORS IN OFFLINE TRAINING MODE

1) HYPERPARAMETERS TUNING WITH OFFLINE TRAINING DATA

The hyperparameters of the regression models were optimized using the grid search method. For the collected training dataset, the MAE and time required to train a model are listed in Table 4. For \( k\text{NNR}, \text{GBR} \), and \( \text{ETR} \), we set the following five, four, and two parameters, respectively.

- Five parameters: neighbor finding algorithm = auto, leaf size = 10, number of neighbors = 3, weight criteria = distance, and distance type = Manhattan
- Four parameters: criterion for split = MAE, learning rate = 0.3, number of estimators = 50, and subsample size = 0.8
- Two parameters: number of estimators = 28 and criterion for split = MAE

The proposed resource adjustment logic was implemented in Python using the regression algorithm codes available in Scikit-Learn.\(^3\)

\(^3\)https://scikit-learn.org/stable/

2) EFFECT OF LOOKUP LATENCY AND RESOURCE UTILIZATION ON PREDICTION

First, we set the CPU quota of the IoT-DS front-end container to 100% and recorded the actual CPU utilization \( y_t^q \) at every observation interval of 1 s by sending lookup queries based on the step workload pattern represented by Eq. (3). Concurrently, we employed four trained regression models to predict the required CPU quota for the same input workload to achieve the specified lookup latency \( (x_4) \) and resource utilization \( (x_6) \) values. We observed that with 100% CPU
The minimum value of the CPU quota was set to 20% (i.e., the allocated quota was never less than 20%, although no workload existed from the end-users) and the step workload pattern for 100 s as specified by Eq. (3) was sent. The values of \( x_0 \) to \( x_7 \) that resulted from the actual CPU quota adjustment by the Conv., LR, GBR and ETR algorithms for a duration of 110 s were recorded. \( k\)NNR was not considered because of its high MAE. For each model, the experiments were repeated five times. The normalized cumulative amount of additional allocated CPU quota (\( Q_{\text{algo}} \)) and the number of lookup latency requirement violations (\( N_{\text{algo}} \)) of the conventional and three regression algorithms are presented in Table 5, where the results of the second and fourth columns were normalized by the corresponding \( Q_{\text{Conv}} \) values. Similarly, columns two and three illustrate the average values of five independent sets of experiments for each algorithm. The fourth and fifth columns represent the best case (among the five sets of experiments) for each algorithm when the number of lookup latency violations and input workload variations were minimal. The best-case latency vs. the QPS and the utilization vs. the allocated quota for the Conv., LR, GBR, and ETR models are illustrated in Figs. 5(a) and (b), (c) and (d), (e) and (f), and (g) and (h), respectively.

We observed that compared with the Conv. algorithm, GBR, ETR, and LR reduced the CPU resource demand (i.e., CPU quota allocations) by 21.9%, 19%, and 54%, respectively. Similarly, GBR and ETR avoided latency requirement violations by 58.2% (\( 1 - 0.42 \times 100\% \)) and 30.1% (\( 1 - 0.76 \times 100\% \)), respectively. Although the LR reduced the CPU 

### TABLE 5. Normalized cumulative amount of CPU allocation and # of latency violations.

| Algorithm | Average of five obs | Best case among five |
|-----------|---------------------|----------------------|
|           | CPU allocation | Latency violations | CPU allocation | Latency violations |
| Conv. [7] | 1 | 11 | 1 | 10 |
| LR        | 0.46 | 21.8 | 0.58 | 16 |
| GBR       | 0.78 | 4.6 | 0.873 | 5 |
| ETR       | 0.81 | 7.6 | 0.903 | 3 |

The lowest MAE resulted in when \( x_4 = 5 \) for LR, GBR, and ETR, and \( x_4 = 6 \) for \( k\)NNR.

(a) The lowest MAE resulted in when \( x_4 = 5 \) for LR, GBR, and ETR, and \( x_4 = 6 \) for \( k\)NNR.

(b) The lowest MAE resulted in when \( x_4 = 90\% \) for \( k\)NNR, GBR, and ETR, and \( x_4 = 85\% \) for LR.

**FIGURE 4.** Effect of \( x_4 \) and \( x_6 \) on the MAE where the predicted values were compared with the online observed CPU utilization values.
D. PREDICTION ERRORS IN ONLINE RETRAINING MODE

A regression algorithm should be selected based on the minimum prediction error and training time, as illustrated in Table 4. A trade-off must exist between these two parameters when we wish to retrain models online with the cumulatively updated dataset, as indicated in Fig. 2. As shown in Table 4, the ETR and \( k \)-NNR had the lowest (0.058) and highest (0.184) prediction errors, respectively. According to the training time requirements, we can set the retraining interval of \( Z \) observations as defined in Section III-B. Both \( k \)-NNR and LR require less than 1/100 s for training and are suitable if 1 s retraining intervals are desired. If we desire a smaller prediction error, 10 s retraining intervals can be set for the GBR or ETR model. In our online retraining experimental results and the following sub-section, we used 10 s retraining intervals for all the four ML models.

We sent a Poisson workload pattern to the front-end container. As shown in Fig. 6, the Poisson workload pattern remained for 300 s, with an average workload of approximately 1000 QPS. We evaluated the predictive performance of the online retrained models with respect to the MAE (defined by Eq. (5)) and the total actual and predicted CPU utilization amounts. It is noteworthy that the predicted resource utilization with 100% CPU quota allocation is a decision value of resource allocation when performing resource adjustment. Furthermore, we defined parameter \( \Delta \) using Eq. (6):

\[
\Delta = \frac{\sum_{t=1}^{X} (y^P_t - y^a_t)}{\sum_{t=1}^{X} y^a_t}
\]  

(6)

where \( X \) is the number of observations; \( y^P_t \) and \( y^a_t \) are the predicted utilization by the retrained models and ground truth or actual observed resource utilization value,
TABLE 6. Prediction error and $\Delta$ of four algorithms with online retraining.

| Algorithm | MAE | $\Delta$ | Avg. util. in % (stdev.) |
|-----------|-----|--------|-------------------------|
| $k$NNR    | 3.498 | 0.0016 | 37.77 (5.27)            |
| LR        | 3.275 | 0.0033 | 37.22 (4.96)            |
| ETR       | 3.560 | 0.00018| 37.29 (5.28)            |
| GBR       | 3.396 | -0.0016| 37.35 (5.00)            |

respectively. A positive $\Delta$ value indicates that the predictions are under-fitted on average, with respect to the actual utilization. Conversely, when $\Delta$ is a negative value, the results are overfitted. It is noteworthy that frequent latency violations may occur when resources are adjusted based on these under-fitting predictions.

We selected the $k$NNR, LR, ETR, and GBR algorithms for online retraining in intervals of 10 s. The training data were collected and updated every second. We used the same optimal hyperparameters for all four algorithms.

The workload and comparisons of actual and predicted resource utilization values for the Poisson workload pattern are illustrated in Fig. 6. The findings shown in Fig. 6 are summarized in Table 6. We observed from Fig. 6 that the prediction performances of all four models improved with retraining (between 50 and 150 s). However, applying more training data might degrade the prediction accuracy owing to the overfitting of predictions (between 151 and 300 s). For the Poisson workload, because the variation in workload was less, the predictions were almost similar to or marginally lower than the actual ones until the first two rounds of training (around 20 s in Fig. 6), resulting in a negative value in both models. The $k$NNR, LR, ETR, and GBR models scored 3.498%, 3.275%, 3.56%, and 3.396% MAE, respectively. The $\Delta$ values were also very similar and positive for the $k$NNR, LR, and ETR, while the GBR resulted in a negative $\Delta$ value. The average utilization (avg. util.) and standard deviation (stdev.) are shown in Table 6. There were sporadically high resource utilization values, specifically for the Poisson workload, which we suppose were due to the experimental system’s virtualization platform. To minimize the impact, we ran the experiment several times and presented the average of best-case results here.

TABLE 7. Normalized cumulative allocated CPU quota and number of times observed average latency exceeding the desired value of 8 ms in the online retraining mode.

| Algorithm | CPU allocation | Latency violations |
|-----------|----------------|--------------------|
| 100% quota| 1              | 8                  |
| $k$NNR    | 0.4749         | 8                  |
| LR        | 0.4745         | 9                  |
| GBR       | 0.5074         | 25                 |
| ETR       | 0.5178         | 3                  |

E. RESOURCE-SAVING AND LATENCY VIOLATIONS IN ONLINE RETRAINING MODE

Similar to the offline training mode, we set the minimum value of the CPU quota to 20%, sent the step workload patterns five times, and recorded $x_0$ to $x_7$ resulting from the CPU quota value adjustment by the $k$NNR, LR, GBR, and ETR models for a duration of 460 s. For each of the four models, we executed the experiments independently. The normalized cumulative allocated CPU quota ($\frac{\sum_{t=1}^{\text{460}} y^\text{a}_t}{\sum_{t=1}^{\text{460}} 1}$) and the total number of lookup latency violations ($N_{\text{algo}}$) for the four regression algorithms are presented in Table 7, where the results shown in the second column are normalized by the corresponding 100% quota allocation to represent the case of no resource adjustment. Columns two and three illustrate the results for 460 s of observation.

We observed that compared with the no-resource-adjustment case (i.e., with 100% CPU quota allocation), the $k$NNR, LR, GBR, and ETR reduced the CPU quota allocation by 52.51%, 52.55%, 49.26%, and 48.22%, respectively. Similarly, $k$NNR and LR exhibited latency violations similar to the no-adjustment case. The GBR predictions were sporadic, resulting in the worst-case latency violation among the four algorithms. The ETR algorithm outperformed the other three by reducing the latency violations by 62.5% ($\frac{8-3}{8} \times 100\%$), while requiring up to 4.33% more CPU resources than the other models.
VI. CONCLUSION AND FUTURE WORK

We proposed an intelligent and agile edge resource control scheme for VNFs by employing multiple regression models. We conducted experiments by implementing the latency-sensitive VNF of IoT-directory service to evaluate the proposed scheme by comparing it with a conventional algorithm in terms of their performances in predicting CPU resource demands and performing dynamic adjustments. Both the offline and online training modes were used to train and update the regression models. In the offline training mode, the effect of the desired input parameters on the prediction error measured in terms of MAE was evaluated. It was demonstrated that the GBR algorithm reduced the CPU resource allocation by 21.9% and avoided cases involving latency requirement violations by 58.2% compared with the conventional algorithm. By adding the online retraining mode, the ETR algorithm outperformed the \( k \)NNR, LR, and GBR by avoiding cases involving latency requirement violations by 62.5% at the expense of 4.33% more CPU resource allocation.

The resource adjustment scheme presented here could not prevent some cases of latency requirement violations in volatile workload situations as it operated reactively. In future work, we will extend our study by adopting long short-term memory-based time series analytical approaches to predict future workloads and proactively adjust resources. Moreover, we will extend our study to enable the monitoring of multiple competing VNFs (such as web servers and media streaming) and adjust not only the CPU, but also the bandwidth and storage resources simultaneously.

REFERENCES

[1] S. Shekhar, H. Abdel-Aziz, A. Bhattacharjee, A. Gokhale, and X. Koutsoukos, “Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications,” in Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD), Jul. 2018, pp. 82–89.

[2] Z. Zhou, J. Gong, Y. He, and Y. Zhang, “Software defined machine-to-machine communication for smart energy management,” IEEE Commun. Mag., vol. 55, no. 10, pp. 52–60, Oct. 2017.

[3] V. P. Kafle, P. Martinez-Julia, and T. Miyazawa, “Automation of 5G network slice control functions with machine learning,” IEEE Commun. Standards Mag., vol. 3, no. 3, pp. 54–62, Sep. 2019.
[4] E. B. Lakew, C. Klein, F. Hernandez-Rodriguez, and E. Elmroth, “Towards faster response time models for vertical elasticity,” in Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput., Dec. 2014, pp. 560–565.

[5] S. Taherzadeh and V. Stankovski, “Dynamic multi-level auto-scaling rules for containerized applications,” Comput. J., vol. 62, no. 2, pp. 174–197, Feb. 2019.

[6] Y. Al-Dhuraihi, F. Paraiso, N. Djarallah, and P. Merle, “Autonomic vertical elasticity of docker containers with ELASTICDOCKER,” in Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD), Jun. 2017, pp. 472–479.

[7] V. P. Kafle, Y. Fukushima, P. Martinez-Julia, T. Miyazawa, and H. Harai, “Adaptive virtual network slices for diverse IoT services,” IEEE Commun. Standards Mag., vol. 2, no. 4, pp. 33–41, Dec. 2018.

[8] B. Gu and Z. Zhou, “Task offloading in vehicular mobile edge computing: A matching-theoretic framework,” IEEE Veh. Technol. Mag., vol. 14, no. 3, pp. 100–106, Sep. 2019.

[9] V. P. Kafle, Y. Fukushima, P. Martinez-Julia, and H. Harai, “Scalable directory service for IoT applications,” IEEE Commun. Standards Mag., vol. 1, no. 3, pp. 58–65, Sep. 2017.

[10] A. H. A. Muktadir and V. P. Kafle, “Prediction and dynamic adjustment of resources for latency-sensitive virtual network functions,” in Proc. 23rd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN), Feb. 2020, pp. 238–242.

[11] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacgumus, “SmartSLA: Cost-sensitive management of virtualized resources for CPU-bound database services,” IEEE Trans. Parallel Distrib. Syst., vol. 26, no. 5, pp. 1441–1451, May 2015.

[12] W. Xu, X. Zhu, S. Singhal, and Z. Wang, “Predictive control for dynamic resource allocation in enterprise data centers,” in Proc. IEEE/IFIP Netw. Oper. Manage. Symp. NOMS, Apr. 2006, pp. 115–126.

[13] T. Miyazawa, H. Harai, Y. Yokota, and Y. Naruse, “Sparse regression model to predict a server load for dynamic adjustments of server resources,” in Proc. 22nd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN), Feb. 2019, pp. 249–256.

[14] K. Ye, Y. Kou, C. Lu, Y. Wang, and C.-Z. Xu, “Modeling application performance in docker containers using machine learning techniques,” in Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst. (ICPADS), Dec. 2018, pp. 1–6.

[15] N. J. Waidwadkar, B. Harirahan, J. E. Gonzalez, B. Smith, and R. H. Katz, “Selecting the best VM across multiple public clouds: A data-driven performance modeling approach,” in Proc. Symp. Cloud Comput., Sep. 2017, pp. 452–465.

[16] Y.-W. Seo, J. Kim, and R. Rajkumar, “Predicting dynamic computational workload of a self-driving car,” in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), Oct. 2014, pp. 3030–3035.

[17] F. Farahnazian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen, “Energy-aware VM consolidation in cloud data centers using utilization prediction model,” IEEE Trans. Cloud Comput., vol. 7, no. 2, pp. 524–536, Apr. 2019.

[18] B. R. Raghunath and A. B., “Prediction based dynamic resource provisioning in virtualized environments,” in Proc. IEEE Int. Conf. Consum. Electron. (ICCE), 2017, pp. 100–105.

[19] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, “SmartCC: A reinforcement learning approach for multipath TCP congestion control in heterogeneous networks,” IEEE J. Sel. Areas Commun., vol. 37, no. 11, pp. 2621–2633, Nov. 2019.

[20] Docker Update Command Documentation. Accessed: Sep. 1, 2020. [Online]. Available: https://docs.docker.com/engine/reference/commandline/update/