IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 7, 2020, accepted April 13, 2020, date of publication April 16, 2020, date of current version May 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2988207

Online Workload Burst Detection for Efficient Predictive Autoscaling of Applications

FATIMA TAHIR[1], MUHAMMAD ABDULLAH[1], FAISAL BUKHARI[1], KHALED MOHAMAD ALMUSTAFA[2], (Associate Member, IEEE), AND WAHEED IQBAL[1], (Member, IEEE)
[1]Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore 54000, Pakistan
[2]College of Computer and Information Sciences, Prince Sultan University, Riyadh 11586, Saudi Arabia

Corresponding author: Waheed Iqbal (waheed.iqbal@pucit.edu.pk)

This work was supported in part by Prince Sultan University, Riyadh, Saudi Arabia, and Higher Education Commission, Pakistan.

ABSTRACT Autoscaling methods are employed to ensure the scalability of cloud-hosted applications. The public-facing applications are prone to receive sudden workload bursts, and the existing autoscaling methods do not handle the bursty workloads gracefully. It is challenging to detect the burst online from the incoming dynamic workload traffic, and then identifying appropriate resources to address the burst without overprovisioning is even harder. In this paper, we address this challenge by investigating the appropriate method for online burst detection and then proposed a novel predictive autoscaling method to use burst detection for satisfying specific response time requirements. We compared the proposed method with multiple state-of-the-art baseline autoscaling methods under multiple realistic and synthetic bursty workloads for a benchmark application. Our experimental results show a 60.8% average decrease in response time violations as compared to the baseline method.

INDEX TERMS Autoscaling, predictive, SLO violations, response time, workload, burstiness, online burst detection.

I. INTRODUCTION

Cloud computing is attractive to host and manage applications for scalability, performance, and cost-effectiveness. One of the core features of cloud computing is on-demand resource provisioning, which enables the users to automatically scale the application resources to satisfy specific service level objectives (SLOs). An application with high response time loses the user attraction and will reduce business opportunities for application owners. Therefore, application response time SLO is critical to ensure for cloud-hosted applications [1]–[3].

Typically, autoscaling methods are used to maintain the performance of cloud-hosted applications by reducing the latency and response time SLO violations [4]. Most of the exiting autoscaling methods are based on either a reactive or a predictive approach. The reactive autoscaling approach manages the resources to the applications based on specific events and a set of rules. For example, a typical reactive autoscaling method increases the number of resources of a cloud-hosted application whenever application-specific metrics, for example, average response time or allocated hardware utilization, for example, CPU, memory, and I/O reaches a specific threshold [5], [6]. These rule-based autoscaling methods do not eliminate the SLO violations as these methods initiate scaling decisions after the occurrence of particular events. Whereas, predictive autoscaling methods proactively scale the application resources by anticipating any possible performance issue may arise in the future. Most of the existing predictive autoscaling methods [7], [8] are based on analytical modeling [9], machine learning [10], and reinforcement learning [11] techniques.

Public-facing applications observe dynamic and bursty workloads. A sudden burst in the workload drastically affects the performance of the applications. If the autoscaling method does not detect the burst and handle it appropriately, the performance of the application will sharply deteriorate. The existing autoscaling methods do not explicitly treat the burst in workloads to minimize the response time SLO violations. There have been some efforts to identify the burstiness in workloads [12]–[15]. These workload detection methods are based on offline techniques that require a large number of historical workload observations as input to identify the burst. It is challenging to identify the burst online in incoming dynamic workloads using small historical observations and then consume the burst detection in the autoscaling

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

method efficiently to minimize the response time SLO violations.

To highlight the challenges associated with identifying bursts in dynamic workloads, we show two real workloads and one synthetic workload in Figure 1. SW1 and SW2 show the two actual workload traces from the Calgary university web server and ClarkNet servers, respectively, whereas SW3 shows a synthetic workload with bursts. The figures highlight the bursts in each workload, which is challenging to detect online and automatically. These bursts deteriorate the performance of the application if appropriate resources are not allocated timely. A typical autoscaling method will try to allocate and deallocate the resources frequently during the bursts and would not be able to identify appropriate allocation timely for minimizing response time SLO violations [16].

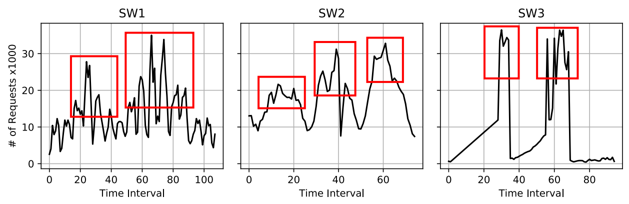

FIGURE 1. Application workloads with bursts.

In this paper, we proposed an efficient autoscaling method that detects bursts online in incoming workloads and then use it in resource allocation decisions to minimum response time SLO violations. In our proposed autoscaling method, first, we predict the future workload using the last k historical workload traces. Second, the predicted workload is used to identify the number of resources instances needed to satisfy the response time SLO threshold. Third, we detect the burstiness in the last k historical workload traces using a burst detection method. Finally, burst detection decision and predicted the number of resource instances are used to scale-out or scale-in the application resources dynamically. We have investigated three different techniques, including Sample Entropy (SE), Normalized Entropy (NE), and Fast Fourier Transformation (FFT), for online burst detection in dynamic workloads. Our proposed method using a trace-driven simulation using four synthetic and five real workloads show excellent performance as compared to the existing state-of-the-art baseline autoscaling methods. We also show the validation of the proposed method on a real containerized testbed infrastructure. The main contributions of this work include:

i. Propose an autoscaling method to minimize the response time SLO violations.
ii. Design an autoscaling method with online burst detection in incoming workload for better performance.
iii. Simulation-based evaluation of the proposed system using four synthetic and five real workloads.
iv. Compare the proposed autoscaling method with state-of-the-art reactive and predictive autoscaling methods as the baselines.
v. Validation of the proposed autoscaling method on a real testbed infrastructure.

The rest of the paper is organized as follows. Related work is discussed in Section 2. We explain the proposed system in Section 3. The experimental setup and design is discussed in Section 4. Experimental results are presented in Section 5. Finally, conclusions and future work are discussed in Section 6.

II. RELATED WORK

There have been many efforts to design autoscaling methods for cloud-hosted applications. For example, Liu and Wee [29] present a reactive autoscaling method. The proposed autoscaling method dynamically scale the application resources whenever CPU utilization or bandwidth utilization of the application resources saturates. Krieger *et al.* [30] proposed a reactive autoscaling method for the bioinformatics and biomedical cloud-hosted applications. The proposed method horizontally scale the application resources to maximize application performance. Chieu *et al.* [31] proposed a reactive autoscaling method. The proposed system used the number of active sessions to find the number of resource instances. However, the proposed method does not maximize the performance of complex applications under dynamically changing workloads. Liu *et al.* [32] also present a reactive autoscaling method. The proposed method uses fuzzy logic to find the resource cluster size to maintain the application response time. Abdullah *et al.* [33] use a simple reactive autoscaling method to maintain the performance of microservices. The proposed method dynamically adds the resources to the microservices whenever response time saturates.

Recently, several researchers designed and evaluated predictive autoscaling methods. For example, Radhika *et al.* [34] present the predictive autoscaling method. The authors use Auto-Regressive Integrated Moving Average (ARIMA) and Recurrent Neural Network-Long Short Term Memory (RNN-LSTM) to predict the future workload using historical CPU and memory utilization and then scale the application resources. Raghunath and Annappa [35] develop a predictive autoscaling method. The author uses a fuzzy-based system to predict the future resources which are needed to maintain the application performance. A recent work by Abdullah *et al.* [36] proposed a predictive autoscaling method using a machine learning to identify the required number of resources for satisfying the response time requirements using a forecasted workload and adjust the resources accordingly.

Some researchers are working to scale the multi-player online game applications resources hosted on the cloud. For example, Khorsand *et al.* [17] proposed a self-learning fuzzy approach for the proactive provision of resources for multi-player online game applications in a cloud environment. The authors applied Maximum Likelihood Estimation and Local Linear Regression for parameter prediction and fuzzy decision-maker to determine appropriate autoscaling decisions. Khorsand *et al.* [18] proposed an approach to provision resources for multi-tier applications hosted on cloud infrastructure using the autonomic computing MAPE-k control loop. Their approach use SVR and Fuzzy Analytical Hierarchy Process in the analysis and planning phase of the MAPE control loop and SVR is used to predict the workload. A similar approach is used by Ghobaei-Arani *et al.* [19] for the resource provisioning approach for multiplayer online games in a cloud environment. Their approach forecasts the workload in the analysis phase using ANFIS (Adaptive Neuro-Fuzzy Inference System) predictor and Fuzzy Decision Tree in the Planning phase to estimate the number of resources to be allocated based on predicted workload to minimize the SLA violation rate. Rafieyan *et al.* [20] proposed an adaptive approach for the multi-criteria task scheduling problems. The proposed method is used to schedule conflicting tasks with minimum QoS violations. Khorsand and Ramezanpour [37] also designed a method for the multi-criteria task scheduling problems with energy efficiency. Safari and Khorsand [38] proposed a Dynamic Voltage and Frequency Scaling method for reducing the energy consumption for time-constrained workflows by distributing different tasks in workflows to appropriate VMs, considering their deadline. The goal of this method is to schedule the user-submitted workflows within given time-constrained with minimum SLO violations.

Workload forecasting is an exciting research area. Many recent works address this problem. For example, Attia *et al.* [39] use the Differential Evolution (DE) algorithm named MSaDE and Artificial Neural Network (ANN) to forecast the workload of cloud-hosted applications. Iqbal *et al.* [40] use the unsupervised learning approach to find future workload patterns for web applications. The author uses URI space partitions using response time and document size to compute the distribution of historically access logs in different partitions. Further, partitions are used to predict workload patterns using probabilistic techniques. There have been some efforts to use the forecasted workload in autoscaling. For example, Roy *et al.* [41] proposed an autoscaling method which uses forecasted workload and application resource utilization for the resource provisioning decisions. Baig *et al.* [28] proposed a method for window size estimation to maximize the prediction accuracy of data center resource utilization using deep neural networks. Any regression-based estimation model can use the predicted window size method for the prediction of resource utilization with minimum error. Similarly, Chen and Wang [27] proposed a method to improve the accuracy and prediction time of resource utilization. The authors use three different components including Ensemble Empirical Mode Decomposition, Run Tests, and ARIMA to improve the prediction results.

A few efforts are made to measure the burstiness in application workloads. For example, Balaji *et al.* [21] use a combination of Hurst Exponent and Sample Entropy methods to detect burst patterns using offline workload traces. Zhang *et al.* [42] use a two-state (ON/OFF) Markov chain model to detect the burst in the given workload. Zhang *et al.* [43] also present a system to identify the burstiness using the search query analysis in web applications. The authors use a probabilistic model on search queries and URLs of a web application to determine the burstiness. Tamime *et al.* [44] proposed a model to measure the burstiness in health-related Wikipedia articles. The authors use the ARIMA model and then classify the burstiness as high, low, or moderate. Benmakrelouf *et al.* [45] proposed a method for detecting abnormal variations in virtualized systems by using a combination of two probabilistic techniques including Z-score and Kullback-Leibler divergence. The proposed solution provides a mapping between resource level and service-level metrics and detects abnormal changes dynamically. Some researcher quantifies the burstiness in workload by analyzing with different methods. For example, Minh *et al.* [12] proposed a system to quantify the burstiness in the given workload by taking workload as a signal. The authors use normalized entropy to calculate the burstiness in the given workload. However, their approach uses a complete offline workload as a single signal and then calculate the burstiness. Ali-Eldin *et al.* [13] also quantify the burstiness using the sample entropy method. Shen *et al.* [14] also proposed a system to indicate the burstiness in the given workload. The authors use the signal processing technique (FFT) to compute the burst density. These methods calculate the burstiness in the given offline workload traces.

Table 1 shows the comparison of existing work with our proposed method. All of the existing burst detection methods work offline on given workload traces, taking them as a single signal. However, to detect the burstiness in workloads online and then use the burst detection decision in the resource autoscaling methods to improve the performance of the cloud-hosted application is a challenging task. In this work, we tackled this challenge and proposed an efficient predictive autoscaling method, which enhances the application performance by using burst detection decisions as input. We use sample entropy (SE), normalized entropy (NE), and Fast-Fourier transformation (FFT) as burst detection techniques and then identify the best method to use in our proposed autoscaling method.

III. PROPOSED SYSTEM

The overall proposed systems is illustrated in Figure 2. The flow and components of the system are labeled and numbered to explain the working of the system. The system works in the following steps:

- First, at every time interval, the last k workload observations $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$ are used to predict the workload for the next interval $\widehat{\alpha}_{t+1}$. We explain this in Section III-A.
- Second, the system uses the last k workload observations $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$ to detect the burstiness using our proposed burst detection method, explained in Section III-B.

TABLE 1. Comparison of related work with proposed method.

Method	Autoscaling Technique	Evaluation Metric	Evaluation Tool	Workload Forecasting	Burst	Technique Used
Shen et al. [14]	Hybrid, Vertical	SLO, Energy cost, Resource utilization	Implementation	No	Yes	Fast Fourier Transformation
Khorsand et al. [17]	Proactive, Horizontal	MAPE (Mean Absolute Percentage Error), Allocated VMs, Cost, RT	Simulation	Yes	No	Self Fuzzy learning, Local Linear regression, Maximum Likelihood Estimation (MLE)
Khorsand et al. [18]	Proactive, Horizontal	Allocated VMs, RT, Cost	Simulation	Yes	No	SVR, FAHP (Fuzzy Analytical Hierarchy Process)
Ghobaei-Arani et al. [19]	Proactive, Horizontal	Allocated VMs, RT, Cost, MSE, RMSE, R^2	Simulation	Yes	No	ANFIS (Adaptive Neuro Fuzzy Inference System) predictor, MAPE control loop, Fuzzy decision tree
Rafieyan et al. [20]	Adaptive, Horizontal	Throughput, Makespan, VM utilization, VM usage cost, Waiting time	Simulation	No	No	Best-Worst Method (BWM), VIKOR
Balaji et al. [21]	Proactive, Horizontal	User error count, Confusion matrix for detection of burst and non-burst	Implementation	No	Yes	Hurst Exponent, Sample Entropy
Calzarossa et al. [22]	Reactive, Horizontal	Completion time, VM utilization, Number of scaling actions	Simulation	No	No	Average Load and Discrete Sampling
Ghobaei-Arani et al. [23]	Hybrid, Horizontal	CPU utilization, VMs allocated, SLA violation, Total cost, Profit	Simulation	Yes	No	Autonomic Computing, Reinforcement learning, Linear Regression model
Aslanpour et al. [24]	Hybrid, Horizontal	Cost, RT, SLA violations, CPU utilization, Number of provisioned VMs, Maximum provisioned VM	Simulation	Yes	No	Radial basis function Neural network
Ghobaei-Arani et al. [25]	Proactive, Horizontal	Response time, CPU utilization, Elasticity	Simulation	Yes	No	MAPE control loop with Learning Automata
Aslanpour et al. [26]	Hybrid, Horizontal	Response time, Cost, SLA violation, Oscillation Mitigation, Resource Utilization, Scaling Actions, VM minutes	Simulation	No	No	MAPE control loop with cost aware approach
Chen et al. [27]	None	Errors to prediction accuracy, Effectiveness, Time-Cost analysis	Implementation	No	No	EEMD (Ensemble Empirical Mode Decomposition), RT (Run Tests), ARIMA
Baig et al. [28]	None	MSE	Implementation	Yes	No	Deep Neural Network
Proposed	**Predictive, Horizontal**	**SLO violations, Scale Operations, Hardware cost**	**Simulation, Implementation**	**Yes**	**Yes**	**Machine Learning (EN, DTR), Entropy**

- Third, the proposed system uses a resources prediction model, explained in Section III-C, to predict the number of resource instances $\widehat{n}$ required to satisfy the response time SLO τ_{slo} requirement to serve the predicted workload $\widehat{\alpha}_{t+1}$.
- Finally, the burst detection decision and predicted number of resource instances $\widehat{n}$ are used by the proposed autoscaling method to adjust the allocated resources to the application dynamically. We explained the proposed autoscaling method in Section III-D.

A. WORKLOAD FORECASTING MODEL

Most of the existing workload forecasting techniques use statistical methods, including ARIMA, ARMA, and Moving Average [46], [47], to estimate the future workloads. However, statistical methods are not efficient in forecasting bursty and dynamic workloads. Some of the recent works [48], [49] use advanced machine learning methods to forecast workload, including neural network, support vector machine, and multi-layer perceptron. However, these methods are compute-intensive and required a large number of training data to train the model to yield better estimation accuracy. In our proposed workload forecasting method, we use a small number of last k observations to capture local trends in the incoming workload. Therefore, we used ElastNet (EN), which is a regularized regression method with absolute and squared penalization. EN performs better than other regression techniques [50]. Figure 3 shows the normalized Mean Absolute Error of different regression algorithms for the World Cup workload with different window sizes. EN shows minimum MAE with window size 10. Therefore, we use EN for workload forecasting.

The proposed workload forecasting model predicts the workload $\widehat{\alpha}_{t+1}$ for the next interval $t+1$ using the last k actual workload observations $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$. For a specific application at the current time interval t, we have the last k workload observations $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$. Then, we can estimate the future workload using:

$$\widehat{\alpha}_{t+1} = b_0 + b_1 \alpha_t, \tag{1}$$

where b_0 and b_1 are the regression parameters, which are estimated using the following objective function:

$$\min_{b_0, b_1} \sum_{i=0}^{k-1} ||\alpha_{t-i} - b_0 - b_1 \alpha_{t-i-1}||_2^2 + \lambda \left(\rho \sum_{j=0}^{1} ||b_j||_1 + \frac{1-\rho}{2} \sum_{j=0}^{1} ||b_j||_2^2 \right), \tag{2}$$

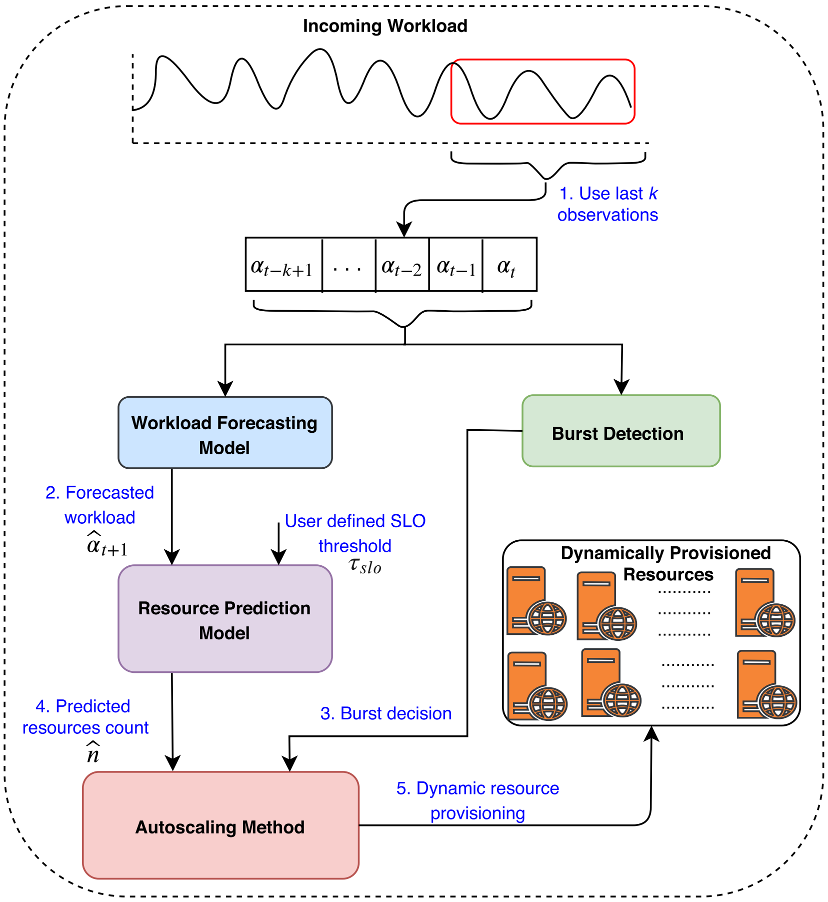
Incoming Workload
1. Use last *k* observations
α_{t-k+1} … α_{t-2} α_{t-1} α_t
Workload Forecasting Model
Burst Detection
2. Forecasted workload $\widehat{\alpha}_{t+1}$
User defined SLO threshold τ_{slo}
Resource Prediction Model
Dynamically Provisioned Resources
4. Predicted resources count $\widehat{n}$
3. Burst decision
Autoscaling Method
5. Dynamic resource provisioning

FIGURE 2. The proposed system components and their interactions. At the first step, the last *k* workload observations are used to build a "Workload Forecast Model" to get the next interval estimated workload. Second, the forecasted workload and user-specific response time SLO threshold are used to get the required number of predicted resources from the "Resource Prediction Model". Third, the last *k* interval observations of the workload are used by the "Brust Detection" module to identify the burst in the incoming workload. Finally, the predicted resources count and burst detection (on/off) is used to autoscale the application resources.

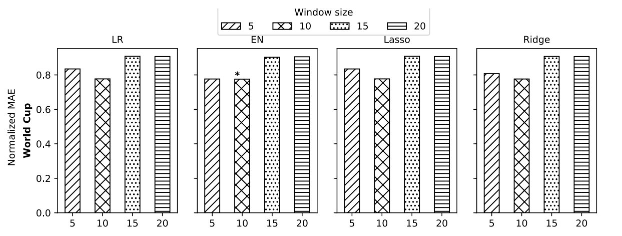

FIGURE 3. Normalized mean absolute error for world cup workload with different regression methods.

where λ is a hyper-parameter which decides the relative importance of reconstruction error and the sparseness of coefficients, $||\cdot||_1$ and $||\cdot||_2^2$ are the ℓ_1 and ℓ_2 norms respectively, and ρ is the mixing ratio or ℓ_1 ratio.

Once the regression parameters b_0 and b_1 are learned, we can estimate the workload for the next time interval as

$$\widehat{\alpha}_{t+1} = \begin{pmatrix} 1 \\ \alpha_t \end{pmatrix}^{\top} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}. \tag{3}$$

B. BURST DETECTION METHOD

Cloud-hosted applications serve dynamic and bursty workloads, which are difficult to detect online. There have been some efforts to detect the burstiness in workloads using offline techniques [12]–[15]. However, online burst detection will help to improve the autoscaling method for minimizing response time violations. In this paper, we investigate the use of different techniques, including Sample Entropy, Normalized Entropy, and Fast Fourier Transformation, for detecting bursts in dynamic workloads for cloud-hosted applications.

Our proposed burst detection technique uses the last k historical workload observation to detect the burstiness online in local workload patterns. Equation 4 shows the burst detection function to identify the burstiness at a current time interval t:

$$\phi : (\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}) \rightarrow \tilde{x}, \quad \text{where } \tilde{x} = \{0, 1\} \tag{4}$$

where ϕ is the burst detection function, $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$ are the last k workload observation which are used to identify the burstiness, and $\tilde{x}$ is the burst detection decision. The value of $\tilde{x} = 0$ shows the current workload observations are not bursty, and $\tilde{x} = 1$ means the current workload observations are bursty. We investigate the use of Sample Entropy (SE), Normalized Entropy (NE), and Fast Fourier Transformation (FFT) techniques to detect the bursts online. We explain these methods in the following subsections.

1) SAMPLE ENTROPY (SE)

Sample Entropy is a measure of information which can be used to detect burst in a given signal and also useful to identify the uncertainty and randomness in time-series data. In our proposed method, last k historical workload observation $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$ are used to detect the burstiness. To compute the sample entropy, first we compute $\{\omega_1, \omega_2, \ldots, \omega_{k-m+1}\}$ sequence vectors of length m in the given k workload observations. Each sequence vector $\omega_i = \{\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+m-1}\}$ where $1 \le i \le k - m + 1$. Once we have sequece vectors, then we use the Equation 5 to compute the sample entropy.

$$SE = \left| -log\left(\frac{\psi^{m+1}}{\psi^m}\right)\right|, \tag{5}$$

ψ^m is computed by using following equation:

$$\psi^m = \frac{1}{k-m}\sum_{i=1}^{k-m}\frac{1}{k-m-1}\psi_i^m, \tag{6}$$

where,

$$\psi_i^m = \sum_{j=1}^{k-m} X_j, \tag{7}$$

$$X_j = \begin{cases} 1 & if\ d[\omega_i, \omega_j] \le r \\ 0 & \text{otherwise} \end{cases} \quad \forall 1 \le j \le k-m \quad \text{and } j \ne i, \tag{8}$$

where, $d[\omega_i, \omega_j]$ is computed using *chebyshev distance* formula [51]. Sample Entropy depends on two parameters m and r. m is the length of the sequence in the given workload, and r is the deviation tolerance or similarity criteria. A large value of m and a smaller value of r gives sharper peaks in the given workload. In our evaluations, we refer the online burst detection technique using Sample Entropy as *B_SE*.

2) NORMALIZED ENTROPY (NE)

Normalized Entropy is another technique commonly used to detect noise, bursts, and randomness in a give time-series signal. In our proposed autoscaling methods, we use normalized entropy to detect the burstiness in last k historical workload observations $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$. Equation 9 is used to compute the Normalized Entropy of the last k workload observations.

$$H_{NE} = -\frac{\sum_{i=t-k+1}^{t} p_i \log p_i}{\log k}, \tag{9}$$

where p_i is the probability of each workload observation according to all workload observations, which is computed using the following equation:

$$p_i = \frac{\alpha_i}{\sum_{j=t-k+1}^{t}\alpha_j}. \tag{10}$$

The value of normalized entropy close to 0 shows that a given workload is bursty. We compare the normalized entropy value of the given workload observation window with the mean of normalized entropy of each previously calculated workload window observations. In our evaluations, we refer the online burst detection technique using Normalized Entropy as *B_NE*.

3) FAST FOURIER TRANSFORMATION (FFT)

Fast Fourier Transformation (FFT) is an algorithm to compute the Discrete Fourier Transformation of a given signal, which converts the signal from the time domain to the frequency domain and vice versa. This technique divides a signal into different frequencies components to analyze the behavior of the signal. Some researchers use FFT to compute the burstiness in the offline workloads by considering it as a signal.

We used FFT in our proposed autoscaling method to detect the workload burstiness in the last k historical workload observations $\{\alpha_t, \alpha_{t-1}, \alpha_{t-2}, \ldots, \alpha_{t-k+1}\}$. First, we compute the Fourier components of the given workload signal. These Fourier components represent the amplitude of different frequencies. Second, we consider the top 80% frequencies as high frequencies and apply inverse FFT over the high-frequency components to calculate positive values, which give the *burst density* metric. These metrics are used to identify the burstiness and non-burstiness in the workload. The percentage of high-frequency is set to 80%. If we choose less value of high-frequency, then it can not clearly differentiate between bursty and non-bursty workload. In our evaluations, we refer the online burst detection technique using FFT as *B_FFT*.

C. RESOURCE PREDICTION MODEL

In our proposed system, the resource prediction model is used to predict the resources instance count $\widehat{n}$ to satisfy the response time SLO threshold τ_{slo} for forecasted workload $\widehat{\alpha}_{t+1}$. The output of the workload forecasting model, the estimated future workload $\widehat{\alpha}_{t+1}$ and user-defined response time SLO threshold τ_{slo} are used as input in our proposed resource prediction model. Equation 11 represents the resource prediction model:

$$\delta : (\widehat{\alpha}_{t+1}, \tau_{slo}) \rightarrow \widehat{n}, \tag{11}$$

where δ is the resource prediction model which predicts the number of resource instances used to satisfy the τ_{slo} for the forecasted workload $\widehat{\alpha}_{t+1}$.

We have evaluated different machine learning methods in the resource prediction model including Linear Regression (LR), Polynomial Regression (PR), Elastic

Net (EN) Regression, Ridge Regression, Lasso Regression XGBoost (XGB) Regression, Random Decision Forests (RDF) Regression, and Decision Tree Regression (DTR). To train the resource prediction model, we used initial performance traces collected from trace-driven simulation. We conducted a small experiment to obtain the dataset for the learning of the resources prediction model. We use reactive autoscaling and increasing workload for the initial examination for the collection of data set. The response time, the number of workload requests and the required number of resources instances are used as the features of the dataset. The reactive autoscaling method dynamically adds the resource instance whenever response time saturates for the linearly increasing workload. We discard all performance traces, which shows response time saturation from the dataset. We split the dataset in 80% for training and 20% for testing to evaluate the model. We select a regression method with a minimum mean square error (MSE).

Figure 4 shows the MSE of different machine learning methods to predict the resources required to satisfy τ_{slo}. We observed DTR yields a minimum error compared to other methods. Therefore, we used DTR in our proposed resource prediction method. DTR is a supervised machine learning technique used for regression problems. DTR develop a rule-based decision tree structure to construct the machine learning model. It can train with less number of training dataset and without normalization of data as compared to the other machine learning techniques. Moreover, DTR performs better as compared to the other tree-based machine learning algorithms, for example, RDF, when the dataset has fewer features.

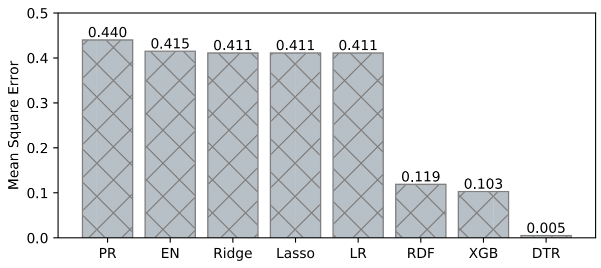

FIGURE 4. Mean square error of different machine learning methods used in the resource prediction model. The Decision Tree Regression (DTR) yields minimum error on test dataset compared to other methods for identifying the appropriate number of resources to the application.

D. PROPOSED AUTOSCALING METHOD

Algorithm 1 shows the proposed autoscaling method. The algorithm takes input including monitoring time interval (ξ), response time SLO threshold (τ_{slo}), window size (k), trained resource prediction model (δ), burst detection method (ϕ), and a set of allocated resources to the application ($\mathbb{R}$).

We used ξ as a monitoring interval to aggregate the incoming user requests as one workload observation.

Algorithm 1 Proposed Autoscaling Method

```
Input: Application monitoring interval ($\xi$), response time SLO threshold ($\tau_{slo}$), workload window size ($k$), resource prediction model ($\delta$), burst detection method ($\phi$), set of allocated resources to the application ($\mathbb{R}$)
Output: Updated set of allocated resources to the application ($\mathbb{R}$)
$burstMode \leftarrow false$
while true do
    Wait for $\xi$ seconds
    $\widehat{\alpha}_{t+1} \leftarrow$ forecast the workload using $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$
    $\widehat{n}_{t+1} \leftarrow \delta(\widehat{\alpha}_{t+1}, \tau_{slo})$
    $\tilde{x} \leftarrow \phi(\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1})$
    $n_{max} \leftarrow Max(\widehat{n}_{t+1}, \widehat{n}_t, \ldots, \widehat{n}_{t-k+1})$
    if $\tilde{x} == 1$ & $burstMode == false$ then
        $burstMode = true$
        $newResourceCount \leftarrow n_{max}$
    else if $\tilde{x} == 1$ & $burstMode == true$ then
        $newResourceCount \leftarrow n_{max}$
    else if $\tilde{x} == 0$ & $burstMode == true$ then
        if $\widehat{n}_{t+1} < |\mathbb{R}|$ then
            $burstMode \leftarrow false$
            $newResourceCount \leftarrow n_{t+1}$
        else
            $newResourceCount \leftarrow n_{max}$
        end
    else if $\tilde{x} == 0$ & $burstMode == false$ then
        $newResourceCount \leftarrow n_{t+1}$
    end
    if $|\mathbb{R}| < newResourceCount$ then
        Scale-out $\{\mathbb{R}\}$
    else if $|\mathbb{R}| > newResourceCount$ then
        Scale-in $\{\mathbb{R}\}$
    end
end
```

For example, $\xi = 60$ will aggregate the workload observation on 60 seconds time interval. In each iteration, the autoscaling method waits for ξ seconds and then used the forecasting method to identify future workload $\widehat{\alpha}_{t+1}$ using the last k workload observations $\{\alpha_t, \alpha_{t-1}, \ldots, \alpha_{t-k+1}\}$. After workload forecasting, the autoscaling method predicts the required number of resource instances using resource prediction model δ for the forecasted workload $\widehat{\alpha}_{t+1}$ to satisfy the response time SLO threshold τ_{slo}. Then the autoscaling method uses the burst detection method ϕ to detect the burstiness. Once the burst decision $\tilde{x}$ is made, then the system predicts the number of resource instances required to satisfy τ_{slo}. Finally, the predicted number of resource instances $\widehat{n}_{t+1}$ are used to dynamically scale-out or scale-in the application resources $\mathbb{R}$.

In experimental evaluations, we used application response time SLO threshold $\tau_{slo} = 200$ milliseconds, application monitoring time interval $\xi = 60$ seconds, and workload window size $k = 10$. A large value of ξ can slow down the autoscaling method to react, and the autoscaling method waits longer before making a decision. Whereas a smaller value of

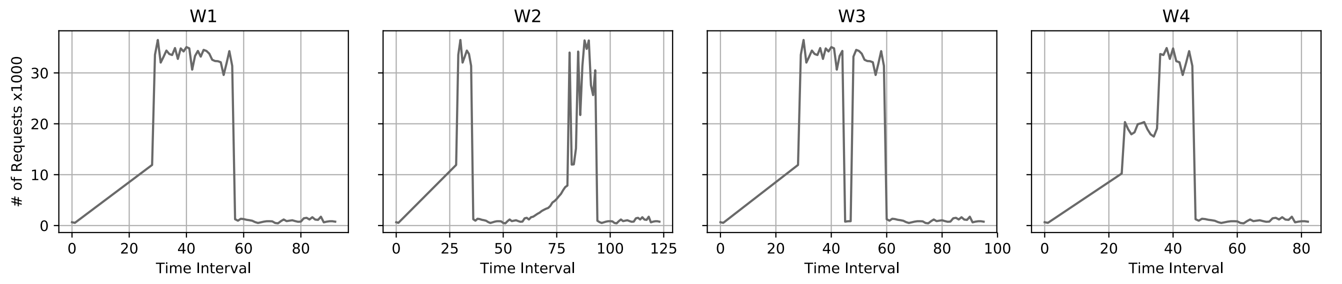

FIGURE 5. Synthetic workloads used in experimental evaluation.

ξ enables the autoscaling method to react quickly and make decisions to manage application resources. However, a large value of ξ ignores the small bursts in the workload. Therefore, we use $\xi = 60$ seconds to monitor the application traces before the autoscaling method triggers. The value of τ_{slo} is also important; a lower value of τ_{slo} is challenging to satisfy by the autoscaling method. In contrast, the larger value of τ_{slo} is easier to ensure by the autoscaling method for web applications. We set $\tau_{slo} = 200ms$, which is reasonable for a typical web application to offer as a response time service level objective (SLO). The value of k also affects the autoscaling method for the workload forecasting and burst detection. A smaller value of k may not sufficient for forecasting model and burst detection method, whereas a large value of k will ignore local and small bursts. We use $k = 10$, which is reasonable to train the forecasting model and burst detection method.

IV. EXPERIMENTAL SETUP AND DESIGN

We evaluate the proposed autoscaling method through a trace-driven simulation for a benchmark application and also validate the simulation results through experiments on a real testbed infrastructure. We compared the effectiveness of the proposed solution with multiple existing baseline autoscaling methods under different synthetic and real workloads. We explained the benchmark application, workloads used in evaluation, baseline autoscaling methods used to compare the proposed system, trace-driven simulation environment, and validation infrastructure and experiments in the following subsections.

A. BENCHMARK APPLICATION

Nowadays, most of the applications are embedding intelligence using Machine Learning (ML) models. The ML models need historical data to train before providing intelligence in the applications. Therefore, to emulate a machine learning workload, we used a regression-based supervised machine learning algorithm (Support Vector Regression) to train on IoT data of size 200K. The dataset consists of time-series data obtained from IoT devices that capture temperature data. The benchmark application is a web application that processes each request by first training the SVR model and then predicts the temperature for the next interval as HTML output. This benchmark application emulates a typical CPU-bound workload.

B. WORKLOADS

In our experimental evaluations, we used four synthetic and five real workloads. Figure 5 shows the synthetic workload 1 (W1), synthetic workload 2 (W2), synthetic workload 3 (W3), and synthetic workload 4 (W4) reflecting different burstiness behaviors. Whereas Figure 6 shows FIFA World Cup [52], web traces of Wikipedia [53], Calgary University web server traces, web traces of NASA's Kennedy Space Center web server, and web traces of ClarkNet web server, which are real workloads used in our experimental evaluation. All of these real workload traces are available publicly.[1]

C. BASELINE AUTOSCALING

We use multiple baseline autoscaling methods to compare and evaluate the proposed autoscaling method. We use two state-of-the-art autoscaling methods, namely React and Predict, used as baseline methods. The React [54] is a reactive autoscaling method that uses a predefined set of rules to scale-out and scale-in the application resources automatically. The React scale-out the application resources whenever the response time of the application increase from the user-defined threshold and the React scale-in the application resources whenever the response time of the last three time interval of the application decreases from the half of the user-defined threshold. The Predict [36] autoscaling method is a recent work that uses a predictive resources provisioning model to identify the required number of resources to satisfy the response time requirements using a forecasted workload and adjust the resources accordingly.

D. TRACE-DRIVEN SIMULATION

We evaluate the proposed autoscaling method using a trace-driven simulation presented in our recent work [36]. For a trace-driven simulation, we model the response time of the benchmark application. We then use the model to identify the application behavior on different workloads and use autoscaling algorithms to manage the allocated resources dynamically. We evaluated three different burst detection

[1]The Internet Traffic Archive, http://ita.ee.lbl.gov/index.html

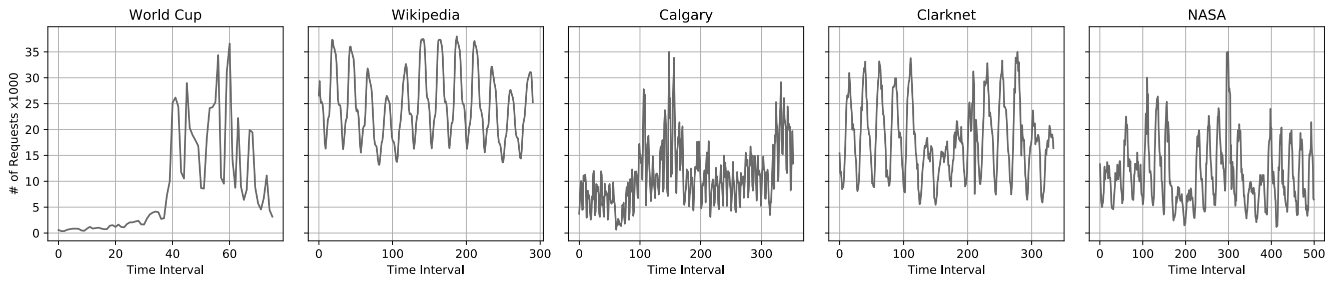

FIGURE 6. Real workloads used in experimental evaluation.

techniques with our proposed autoscaling method to detect the burstiness, as explained in Section III-B. We evaluated the proposed solution under nine different workloads and compared it with two state-of-the-art autoscaling method.

E. VALIDATION INFRASTRUCTURE AND EXPERIMENT

To validate our proposed autoscaling method, we performed experiments on a Docker Swarm testbed Cluster [55]. The testbed cluster consist of four *core i7* physical machines with *8-cores* CPU, *16 GB* physical memory and *2 TB* disk. We configured three physical machines as Docker swarm worker nodes and one physical machine configured as Docker swarm manager node. We can run a maximum of 12 containers instances simultaneously with *one-core* CPU and *2 GB* physical memory on the testbed cluster.

To validate the proposed solution on the real testbed, we used the Clarknet workload to study the effectiveness of the proposed autoscaling method. We compared the results with React and Predict baseline autoscaling methods. For the validation experiment, we generated the workload using httperf [56] for the benchmark application.

F. EVALUATION METRICS

To evaluate the proposed method and compared it with baseline methods, we compute *total processed requests*, *response time SLO violations*, and *the number of scale operations* for each experiment.

i. *Total process requests* shows the number of requests served by the application for a given workload. The autoscaling method yields higher total process requests is considered better as it shows that maximum possible requests are served and less requests are rejected compared to the other autoscaling methods.
ii. *Response time SLO violations* refer to the number of requests takes higher response time as compared to the expected response time threshold. The autoscaling method yield low number of SLO violations are considered better.
iii. *The number of scale operations* refers to the number of scaling decisions performed by the autoscaling method to dynamically add or remove the resources. The autoscaling method yield scale operations are considered better as it will use less number of instances and will incur less operational cost.

V. EXPERIMENTAL RESULTS

Table 2 summarizes the experimental results of React, Predict, and the proposed autoscaling method using B_SE, B_NE, and B_FFT burst detection techniques under different synthetic and realistic workloads. For each experiment, we computed the total number of processed requests, the total number of response time SLO violations, and the total number of scale operations for all workloads and autoscaling methods. The autoscaling method yields the maximum number of processed requests, the minimum number of SLO violations, and the minimum number of scale operations are considered the best. The experimental evaluations show that the proposed autoscaling method using burst detection techniques outperform the baseline autoscaling methods. Overall, the proposed autoscaling method using B_SE techniques shows better performance by minimizing SLO violations, maximizing number of processed requests, and minimizing the scale operations.

To explain the effectiveness of the proposed autoscaling method, we show the box plot of application response time for all experiments. Figure 7 and 8 show the box plot of response time using each autoscaling method with synthetic and real workloads respectively. Figures show that the application response time with burst detection techniques is less as compared to the other autoscaling method without burst detection. Moreover, Sample Entropy-based burst detection technique (B_SE) performs better as compared to the other two burst detection techniques during the experiments to minimize the response time violations.

Figure 9 shows the application response time and the dynamic allocation of resource instances of each time interval during the experiments using different autoscaling methods under the W4 synthetic workload. We use 200ms as a response time SLO threshold. The figure shows the response time SLO violations are high using the baseline methods (React and Predict) as compared to the autoscaling with burst detection techniques. Moreover, Sample Entropy-based burst detection technique (B_SE) outperforms other autoscaling methods to minimize the response time SLO violations.

TABLE 2. Percentage of the total number of processed requests, percentage of total SLO violations, and the total number of scale operations for all workloads using baselines and the proposed autoscaling method using B_SE, B_NE, and B_FFT burst detection techniques.

Workload	Processed Requests %					SLO Violations %					Scale Operations				
	Baseline		Proposed			Baseline		Proposed			Baseline		Proposed		
	React	Predict	B_SE	B_NE	B_FFT	React	Predict	B_SE	B_NE	B_FFT	React	Predict	B_SE	B_NE	B_FFT
W1	87.26	96.94	**97.03**	96.99	96.99	12.74	3.06	**2.97**	3.01	3.01	55	25	14	**13**	**13**
W2	78.58	91.73	**96.2**	91.73	91.73	21.42	8.27	**3.8**	8.27	8.27	77	31	**14**	25	25
W3	86.37	90.18	94.56	**96.99**	**96.99**	13.63	9.82	5.44	**3.01**	**3.01**	63	31	14	**13**	**13**
W4	90.2	95.33	**96.08**	96.02	96.02	9.8	4.67	**3.92**	3.98	3.98	53	26	**18**	**18**	**18**
World Cup	82.08	84.51	**93.26**	**93.26**	**93.26**	17.92	15.49	**6.74**	**6.74**	**6.74**	39	35	**10**	**10**	**10**
Wikipedia	92.17	96.26	**99.56**	96.58	96.27	7.83	3.74	**0.44**	3.42	3.73	248	208	**29**	142	135
Calgary	92.04	90.65	**98.94**	98.64	95.8	7.96	9.35	**1.06**	1.36	4.2	221	228	**12**	19	88
Clarknet	87.69	92.59	**99.68**	98.64	92.71	12.31	7.41	**0.32**	1.36	7.29	276	258	**18**	69	182
NASA	81.68	91.09	**99.45**	98.2	91.77	18.32	8.91	**0.55**	1.8	8.23	418	351	**28**	57	226

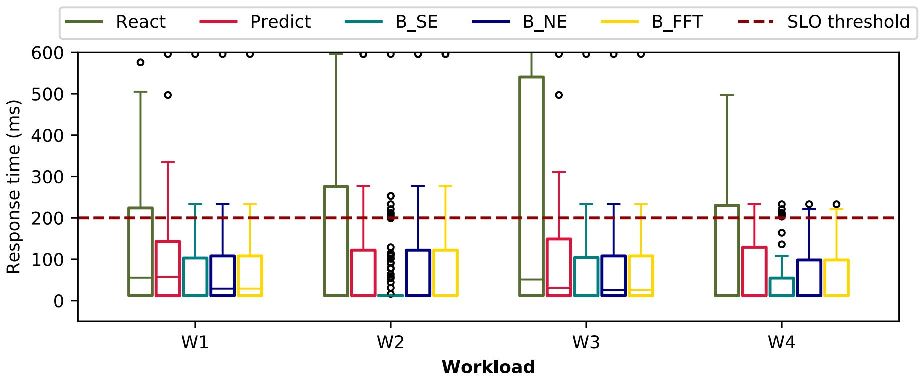

FIGURE 7. Box plot for application response time using different autoscaling methods under synthetic workloads.

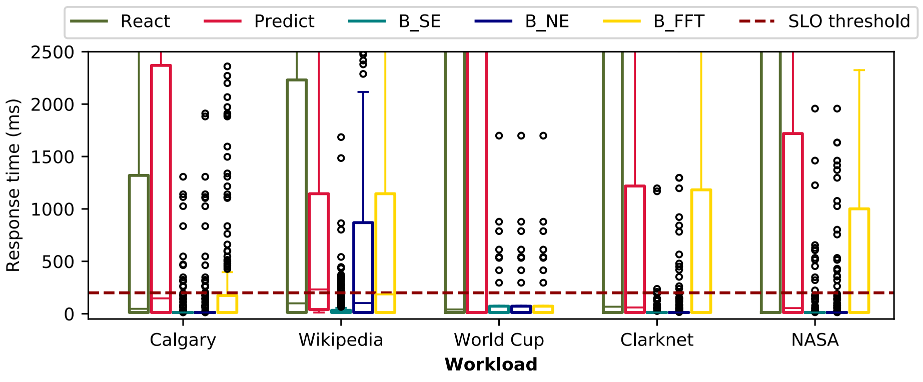

FIGURE 8. Box plot for application response time using different autoscaling methods under real workloads.

Figure 10 shows the application response time and the dynamic allocation of resource instances in each time interval using different autoscaling methods under the Calgary workload. This figure shows the response time SLO violations are higher using the React and the Predict baseline methods as compared to the proposed autoscaling with burst detection techniques. Moreover, B_SE burst detection outperforms other burst detection techniques.

Figure 11 shows the comparison of processed requests, SLO violations, and scale operations for autoscaling methods relative to the React baseline autoscaling method using real workloads. These results show that the Sample Entropy-based (B_SE) predictive autoscaling method outperforms other methods. B_SE yield $0.1\times$, $0.12\times$, $0.1\times$, $0.12\times$, and $0.2\times$ times higher total number of requests as compared to the reactive autoscaling method for Calgary, World cup, Wikipedia,

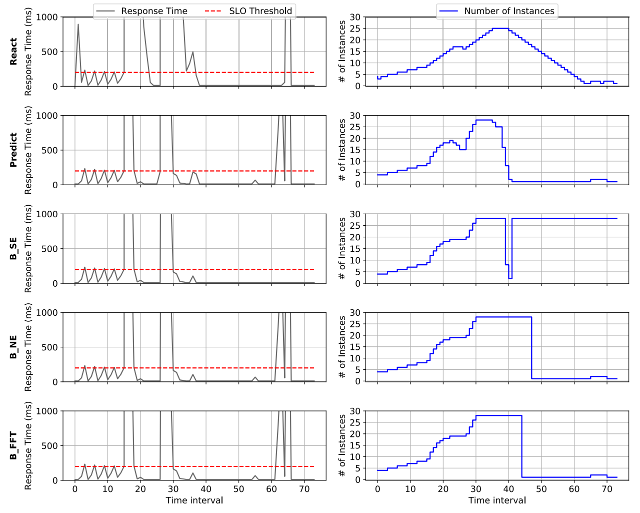

FIGURE 9. Experimental results showing application response time and the dynamic allocation of instances for each time interval using React, Predict, and the proposed autoscaling using B_SE, B_NE, and B_FFT burst detection techniques under the W4 synthetic workload.

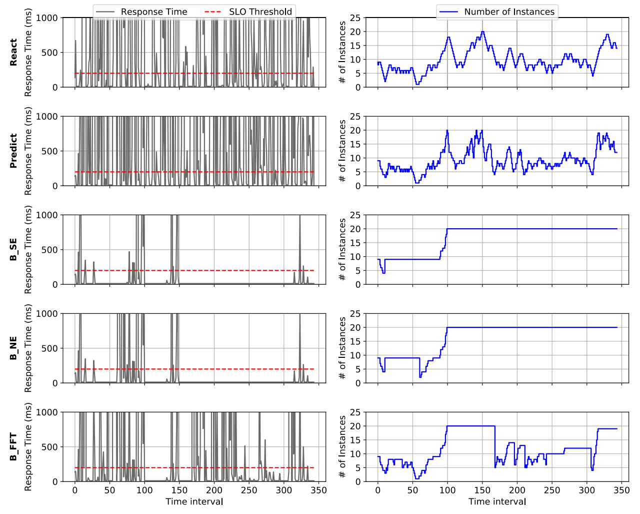

FIGURE 10. Experimental results showing application response time and the dynamic allocation of instances for each time interval using React, Predict, and the proposed autoscaling using B_SE, B_NE, and B_FFT burst detection techniques under the Calgary workload.

ClarkNet and NASA workloads respectively. Moreover, B_SE yield 0.84×, 0.97×, 0.6×, 0.98× and 0.98× times less number of response time SLO violations as compared to the reactive autoscaling method for Calgary, World cup, Wikipedia, ClarkNet and NASA workloads respectively. Whereas, the scale operations count is also minimized by 0.9×, 0.8×, 0.7×, 0.9×, and 0.9× times for the B_SE as compared to the reactive autoscaling method using Calgary, World cup, Wikipedia, ClarkNet and NASA workloads respectively.

To study the cost of using different autoscaling methods, we used 0.0014$ per minute cost for each instance similar to

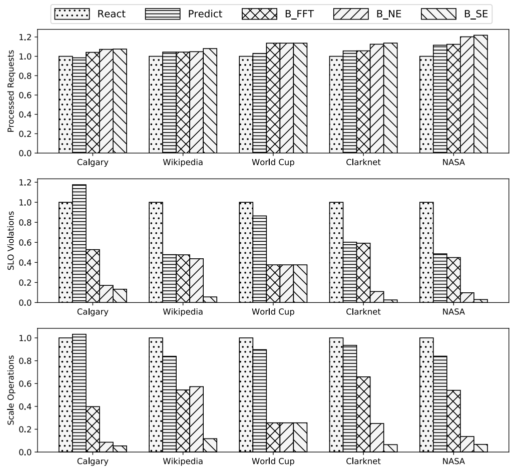

FIGURE 11. Comparison of processed requests, SLO violations, and scale operations for autoscaling methods relative to the React baseline autoscaling method using real workloads.

AWS *c5.large* instance. Figure 12 shows the cost comparison of autoscaling methods relative to the React baseline method for synthetic workloads. B_SE yields 58%, 186%, 67%, and 76% more cost as compared to reactive autoscaling method for W1, W2, W3, and W4 workloads respectively. B_NE yields 6%, 30%, 11%, and 1% more cost as compared to baseline for W1, W2, W3, and W4 workloads respectively. B_FFT method takes 13% and 4% more cost as compared to reactive for W2 and W3 workload. Whereas for W1 and W4 B_FFT takes 1% and 9% less cost as compared to the reactive baseline method.

Figure 13 shows the cost comparison of autoscaling methods relative to the React baseline method for real workloads. B_SE yields 58%, 31%,74%, 85%, and 147% more cost as compared to reactive autoscaling method for World Cup, Wikipedia, Calgary, ClarkNet and NASA workloads respectively. B_NE yields 58%, 12%, 72%, 67%, and 145% more cost as compared to baseline for World Cup, Wikipedia, Calgary, ClarkNet and NASA workloads respectively. B_FFT method yields 58%, 5%, 22%, 19%, and 43% more cost as compared to baseline for World Cup, Wikipedia, Calgary, ClarkNet, and NASA workloads respectively.

To summarize the effectiveness of the proposed solution, we computed the average percentage relative to the React baseline method using the results of all workloads. We used B_SE as a burst detection technique in the proposed autoscaling method. Figure 14 shows the average percentage relative to the React baseline method for Predict and proposed autoscaling methods. The proposed solution shows a 9.65% average increase in the total number of processed requests as compare to the React baseline method. Moreover, it yields an average of 60.8% less response time SLO violations and 61.0% fewer scale operations using results of all workloads as compare to the React baseline method.

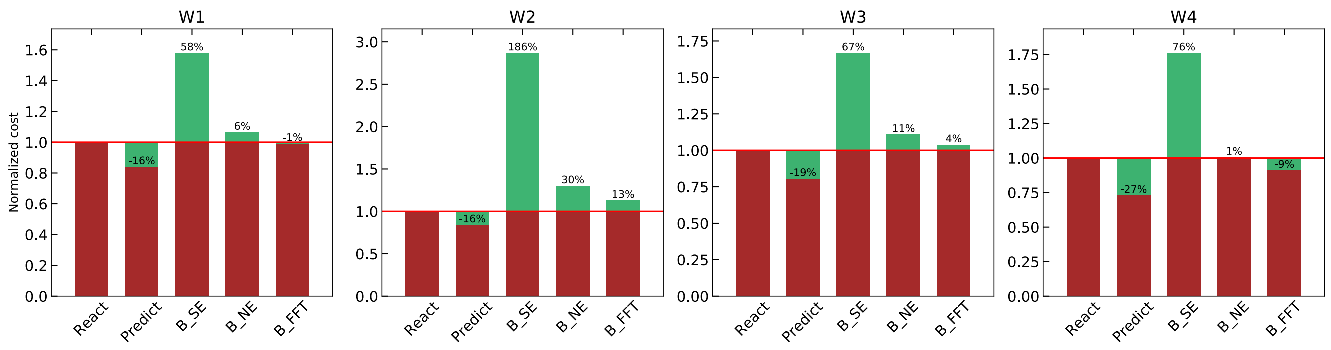

FIGURE 12. Comparison of cost using different autoscaling methods to satisfy response time SLO requirements under synthetic workloads.

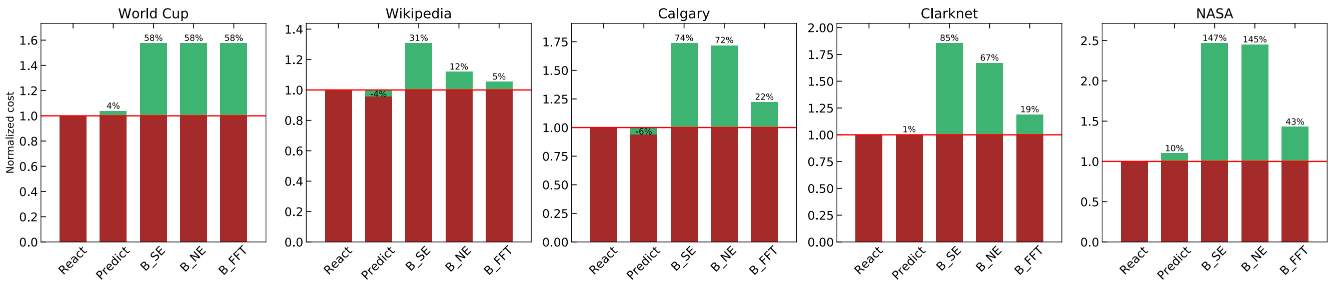

FIGURE 13. Comparison of cost using different autoscaling methods to satisfy response time SLO requirements under real workloads.

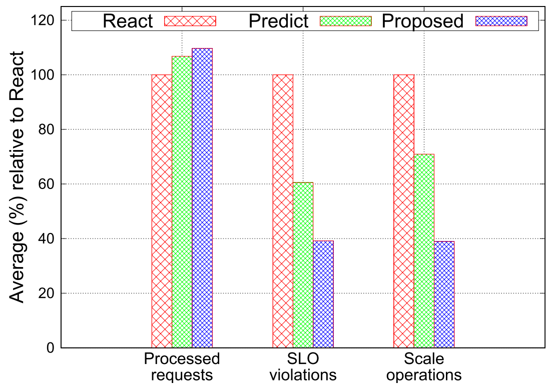

FIGURE 14. The average percentage of total processed requests, response time SLO violations, and scale operations for all workloads results relative to the React baseline for the proposed (B_SE) and predictive autoscaling method.

A. VALIDATION ON CONTAINERIZED TESTBED

To validate our trace-driven simulation results, we performed experiments on a containerized testbed cluster, as explained in Section IV-E. We compare the proposed autoscaling method using B_SE burst detection technique with both of the baseline autoscaling methods under ClarkNet's real workload traces.

Table 3 shows the percentage of total processed requests, percentage of response time SLO violations, and scale operations count during the validation experiments using React, Predict, and the proposed autoscaling methods for the benchmark application. The proposed autoscaling method yields a higher number of processed requests, minimizes the response time SLO violations, and also uses a fewer number of scale operations to satisfy the response time requirements compared to the baseline methods.

Figure 15 shows the comparison of the total number of processed requests, response time SLO violations, and scale operation for the validation experiments using the proposed and baseline autoscaling methods. The proposed autoscaling method outperforms the baseline methods to minimize the SLO violations, maximize the number of processed requests, and minimize the scale operations. The number of process requests during the proposed autoscaling method increases 0.10× times as compare to reactive autoscaling. The proposed autoscaling method yields 0.95× fewer response time SLO violations and 0.30× less scale operations as compared to the React baseline autoscaling method.

1) HYPOTHESIS TESTING

To determine the statistical evidence for the obtained results, we designed the two sets of hypotheses. The first set of hypothesis is related to the SLO violations for the comparison of the proposed and the baseline algorithms.

$$H_{0a} : \mu_1 = \mu_2$$
$$H_{1a} : \mu_1 > \mu_2$$

The second set of hypothesis is related to the feature Proposed Requests as

$$H_{0b} : \mu_3 = \mu_4$$
$$H_{1b} : \mu_3 < \mu_4$$

TABLE 3. Percentage of the total number of processed requests, percentage of total SLO violations, and the total number of scale operations for ClarkNet real workload using baselines and the proposed autoscaling method with B_SE burst detection technique under validation experiment.

Workload	Processed Requests (%)			SLO Violations (%)			Scale Operations		
	React	Predict	Proposed (B_SE)	React	Predict	Proposed (B_SE)	React	Predict	Proposed (B_SE)
Clarknet	92.12	95.04	**99.84**	19.93	15.64	**0.48**	36	73	**7**

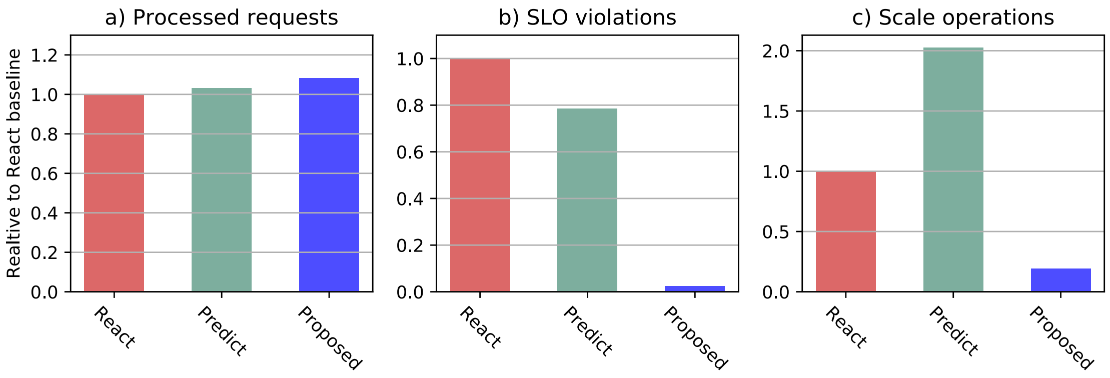

FIGURE 15. Comparison of processed requests, SLO violations, and scale operations for autoscaling methods relative to the React baseline autoscaling method using ClarkNet real workload under validation experiment.

1) μ_1 : The population mean for the SLO violations of the baseline method.
2) μ_2 : The population mean for the SLO violations of the proposed method.
3) μ_3 : The population mean for the Processed Requests of the baseline method.
4) μ_4 : The population mean for the Processed Requests of the proposed method.

The level of significance is set at $\alpha = 0.05$. We rejected the null hypothesis (H_0 or H_{0b}) when the p-value based on the paired t-test is less than 0.05 for any specific feature. The null hypothesis H_{0a} for the first set of the hypothesis is rejected for the Average SLO violations count as the p-value is less than 0.05 for this feature as described in Table 4. It means that the average SLO violations count for the baseline methods is greater than the proposed method.

TABLE 4. Statistical comparison of the proposed algorithm based on SLO violations count and processed requests with the baseline algorithms.

Baseline Algorithms	Statistical Parameters	SLA Missing Count	Processed Requests
Predictive	Mean	569.5963	3461.3727
	SD	982.4765	1484.3271
	t-value	7.0578	-1.0526
	p-value	1.0543e-11	0.1467
React	Mean	725.7019	3355.3168
	SD	1457.5061	1168.9614
	t-value	6.1370	-1.8757
	p-value	2.4841e-09	0.0308

We rejected the null hypothesis H_{0b} for the second set of the hypothesis for the feature Processed Requests as the p-value is less than 0.05, as described in Table 4. It means that one of the baseline algorithm React's average Processed Requests is less than the proposed algorithm. On the other hand, we failed to reject the null hypothesis H_{0b} for one of the baseline algorithm Predictive as it's p-value is greater than 0.05. It means that there is no significant difference between one of the baseline method (Predictive) and the proposed method for the feature Processed Requests. The hypothesis analysis proved the superiority of our method for both the features compared to the baseline algorithms.

B. DISCUSSION

Typical autoscaling methods dynamically provision the application resources without explicitly considering the burstiness in the incoming application workload. The bursty workloads degrade the application performance significantly, even in the presence of traditional autoscaling methods. The proposed autoscaling method detects the bursts in workload automatically and then use the burst detection decision to manage the application resources for minimizing the response time SLO violations. Our solution is capable of identifying the bursty traffic, which is frequently changing, and then disable the scale-in decisions and only performs predictive scale-out operations during the burst. This reduces the oscillation in scale operations and helps to minimize the response time SLO violations. The proposed autoscaling enhances the application performance by maximizing the number of processed requests and minimizing the response time SLO violations under real and synthetic bursty workloads compared to the existing autoscaling methods.

For the validation experiment on a containerized testbed infrastructure, the proposed autoscaling method increase 0.10× processed request, minimized 0.95× response time SLO violations, and 0.30× less scale operations as compared to the React baseline autoscaling. The proposed autoscaling method will be helpful in handling the workloads containing burstiness gracefully compared to the existing autoscaling methods. Moreover, the proposed method yields an average of 60.8% decrease in the number of SLO violations as compared to the baseline methods by efficiently handling the burstiness.

VI. CONCLUSION AND FUTURE WORK

Nowadays, cloud-hosted applications face dynamic and bursty workloads, and autoscaling methods are used to maintain the performance of cloud-hosted applications. However, bursty workloads degrade application performance because the existing autoscaling methods do not explicitly handle the bursts. In this paper, we proposed an efficient predictive autoscaling method which is capable of detecting bursts in dynamic workloads. The detected burst is incorporate in the autoscaling method to satisfy a specific response time SLO requirements. Our extensive evaluation using simulations and then validation experiments on a real testbed show the effectiveness of the proposed method by outperforming the existing state-of-the-art autoscaling methods. The proposed solution will be helpful to ensure application performance in the presence of bursty workloads.

Currently, we are extending our work by investigating the use of multi-objective optimization techniques to ensure response time SLO requirements with minimal cost. In the future, we also plan to improve the proposed method by predicting the bursts to eliminate the chances of response time SLO violations.

ACKNOWLEDGMENT

The authors would like to thank Prince Sultan University, Riyadh, Saudi Arabia, for partially supporting this work.

REFERENCES

[1] S. Sotiriadis, N. Bessis, C. Amza, and R. Buyya, "Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling," *IEEE Trans. Services Comput.*, vol. 12, no. 2, pp. 319–334, Mar. 2019.
[2] V. Hayyolalam and A. A. P. Kazem, "A systematic literature review on QoS-aware service composition and selection in cloud environment," *J. Netw. Comput. Appl.*, vol. 110, pp. 52–74, May 2018.
[3] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 800–813, Apr. 2019.
[4] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Providing performance guarantees for cloud-deployed applications," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 269–281, Jan. 2020.
[5] W. Iqbal, M. Dailey, and D. Carrera, "SLA-driven adaptive resource management for Web applications on a heterogeneous compute cloud," in *Proc. IEEE Int. Conf. Cloud Comput.* Berlin, Germany: Springer, 2009, pp. 243–253.
[6] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai, "Optimal autoscaling in a IaaS cloud," in *Proc. 9th Int. Conf. Auton. Comput. (ICAC)*, 2012, pp. 173–178.
[7] W. Iqbal, A. Erradi, M. Abdullah, and A. Mahmood, "Predictive auto-scaling of multi-tier applications using performance varying cloud resources," *IEEE Trans. Cloud Comput.*, early access, Sep. 27, 2019, doi: 10.1109/TCC.2019.2944364.
[8] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling Web applications in clouds: A taxonomy and survey," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–33, Sep. 2018.
[9] L. Aniello, S. Bonomi, F. Lombardi, A. Zelli, and R. Baldoni, "An architecture for automatic scaling of replicated services," in *Networked Systems*. Cham, Switzerland: Springer, 2014.
[10] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic cloud services based on machine learning techniques," *J. Cloud Comput.*, vol. 8, no. 1, p. 5, Dec. 2019.
[11] W. Iqbal, M. N. Dailey, and D. Carrera, "Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multitier Web applications," *IEEE Syst. J.*, vol. 10, no. 4, pp. 1435–1446, Dec. 2016.
[12] T. N. Minh, L. Wolters, and D. Epema, "A realistic integrated model of parallel system workloads," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, May 2010, pp. 464–473.
[13] A. Ali-Eldin, O. Seleznjev, S. S.-D. Luna, J. Tordsson, and E. Elmroth, "Measuring cloud workload burstiness," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, Dec. 2014, pp. 566–572.
[14] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. 2nd ACM Symp. Cloud Comput. (SOCC)*, New York, NY, USA, 2011, pp. 5:1–5:14.
[15] A. Adegboyega, "Quantifying cloud workload burstiness: New measures and models," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 987–990.
[16] D. Perez-Palacin, R. Mirandola, and J. Merseguer, "Accurate modeling and efficient QoS analysis of scalable adaptive systems under bursty workload," *J. Syst. Softw.*, vol. 130, pp. 24–41, Aug. 2017.
[17] R. Khorsand, M. Ghobaei-Arani, and M. Ramezanpour, "A self-learning fuzzy approach for proactive resource provisioning in cloud environment," *Softw., Pract. Exper.*, vol. 49, no. 11, pp. 1618–1642, Nov. 2019.
[18] R. Khorsand, M. Ghobaei-Arani, and M. Ramezanpour, "FAHP approach for autonomic resource provisioning of multitier applications in cloud computing environments," *Softw., Pract. Exper.*, vol. 48, no. 12, pp. 2147–2173, Dec. 2018.
[19] M. Ghobaei-Arani, R. Khorsand, and M. Ramezanpour, "An autonomous resource provisioning framework for massively multiplayer online games in cloud environment," *J. Netw. Comput. Appl.*, vol. 142, pp. 76–97, Sep. 2019.
[20] E. Rafieyan, R. Khorsand, and M. Ramezanpour, "An adaptive scheduling approach based on integrated best-worst and VIKOR for cloud computing," *Comput. Ind. Eng.*, vol. 140, Feb. 2020, Art. no. 106272.
[21] M. Balaji, C. A. Kumar, and G. S. V. R. K. Rao, "Non-linear analysis of bursty workloads using dual metrics for better cloud resource management," *J. Ambient Intell. Hum. Comput.*, vol. 10, no. 12, pp. 4977–4992, Dec. 2019.
[22] M. C. Calzarossa, L. Massari, and D. Tessera, "Evaluation of cloud autoscaling strategies under different incoming workload patterns," *Concurrency Comput., Pract. Exper.*, p. e5667, Jan. 2020, doi: 10.1002/cpe.5667.
[23] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Gener. Comput. Syst.*, vol. 78, pp. 191–210, Jan. 2018.
[24] M. S. Aslanpour, S. E. Dashti, M. Ghobaei-Arani, and A. A. Rahmanian, "Resource provisioning for cloud applications: A 3-D, provident and flexible approach," *J. Supercomput.*, vol. 74, no. 12, pp. 6470–6501, Dec. 2018.
[25] M. Ghobaei-Arani, A. Souri, T. Baker, and A. Hussien, "ControCity: An autonomous approach for controlling elasticity using buffer management in cloud computing environment," *IEEE Access*, vol. 7, pp. 106912–106924, 2019.
[26] M. S. Aslanpour, M. Ghobaei-Arani, and A. N. Toosi, "Auto-scaling Web applications in clouds: A cost-aware approach," *J. Netw. Comput. Appl.*, vol. 95, pp. 26–41, Oct. 2017.
[27] J. Chen and Y. Wang, "A hybrid method for short-term host utilization prediction in cloud computing," *J. Electr. Comput. Eng.*, vol. 2019, pp. 1–14, Mar. 2019.
[28] S.-U.-R. Baig, W. Iqbal, J. L. Berral, and D. Carrera, "Adaptive sliding windows for improved estimation of data center resource utilization," *Future Gener. Comput. Syst.*, vol. 104, pp. 212–224, Mar. 2020.

[29] H. Liu and S. Wee, "Web server farm in the cloud: Performance evaluation and dynamic architecture," in *Proc. 1st Int. Conf. Cloud Comput. (CloudCom)*. Berlin, Germany: Springer-Verlag, 2009, pp. 369–380.

[30] M. T. Krieger, O. Torreno, O. Trelles, and D. Kranzlmüller, "Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows," *Future Gener. Comput. Syst.*, vol. 67, pp. 329–340, Feb. 2017.

[31] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of Web applications in a virtualized cloud computing environment," in *Proc. IEEE Int. Conf. e-Bus. Eng.*, Oct. 2009, pp. 281–286.

[32] B. Liu, R. Buyya, and A. N. Toosi, "A fuzzy-based auto-scaler for Web applications in cloud computing environments," in *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, 2018, pp. 797–811.

[33] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for Web application auto-decomposition into microservices," *J. Syst. Softw.*, vol. 151, pp. 243–257, May 2019.

[34] E. G. Radhika, G. S. Sadasivam, and J. F. Naomi, "An efficient predictive technique to autoscale the resources for Web applications in private cloud," in *Proc. 4th Int. Conf. Adv. Electr., Electron., Inf., Commun. Bio-Inform. (AEEICB)*, Feb. 2018, pp. 1–7.

[35] B. R. Raghunath and B. Annappa, "Dynamic resource allocation using fuzzy prediction system," in *Proc. 3rd Int. Conf. Converg. Technol. (ICT)*, Apr. 2018, pp. 1–6.

[36] M. Abdullah, W. Iqbal, A. Erradi, and F. Bukhari, "Learning predictive autoscaling policies for cloud-hosted microservices using trace-driven modeling," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2019, pp. 119–126.

[37] R. Khorsand and M. Ramezanpour, "An energy-efficient task-scheduling algorithm based on a multi-criteria decision-making method in cloud computing," *Int. J. Commun. Syst.*, vol. 33, no. 9, p. e4379, Mar. 2020.

[38] M. Safari and R. Khorsand, "Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment," *Simul. Model. Pract. Theory*, vol. 87, pp. 311–326, Sep. 2018.

[39] M. A. Attia, M. Arafa, E. A. Sallam, and M. M. Fahmy, "Application of an enhanced self-adapting differential evolution algorithm to workload prediction in cloud computing," *Int. J. Inf. Technol. Comput. Sci.*, vol. 11, no. 8, pp. 33–40, Aug. 2019.

[40] W. Iqbal, A. Erradi, and A. Mahmood, "Dynamic workload patterns prediction for proactive auto-scaling of Web applications," *J. Netw. Comput. Appl.*, vol. 124, pp. 94–107, Dec. 2018.

[41] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, Jul. 2011, pp. 500–507.

[42] S. Zhang, Z. Qian, Z. Luo, J. Wu, and S. Lu, "Burstiness-aware resource reservation for server consolidation in computing clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 964–977, Apr. 2016.

[43] C. Zhang, S. Zhang, C. Lei, and P. Lin, "Burstiness in query log: Web search analysis by combining global and local evidences," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 1388–1391.

[44] R. Al Tamime, R. Giordano, and W. Hall, "Observing burstiness in wikipedia articles during new disease outbreaks," in *Proc. 10th ACM Conf. Web Sci.*, May 2018, pp. 117–126.

[45] S. Benmakrelouf, C. St-Onge, N. Kara, H. Tout, C. Edstrom, and Y. Lemieux, "Abnormal behavior detection using resource level to service level metrics mapping in virtualized systems," *Future Gener. Comput. Syst.*, vol. 102, pp. 680–700, Jan. 2020.

[46] P. Singh, P. Gupta, and K. Jyoti, "TASM: Technocrat ARIMA and SVR model for workload prediction of Web applications in cloud," *Cluster Comput.*, vol. 22, no. 2, pp. 619–633, Jun. 2019.

[47] D. Janardhanan and E. Barrett, "CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models," in *Proc. 12th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 55–60.

[48] S. Sharifian and M. Barati, "An ensemble multiscale wavelet-GARCH hybrid SVR algorithm for mobile cloud computing workload prediction," *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 11, pp. 3285–3300, Nov. 2019.

[49] J. Kumar and A. K. Singh, "Workload prediction in cloud using artificial neural network and adaptive differential evolution," *Future Gener. Comput. Syst.*, vol. 81, pp. 41–52, Apr. 2018.

[50] Z. Zhang, Z. Lai, Y. Xu, L. Shao, J. Wu, and G.-S. Xie, "Discriminative elastic-net regularized linear regression," *IEEE Trans. Image Process.*, vol. 26, no. 3, pp. 1466–1481, Mar. 2017.

[51] S. Gultom, S. Sriadhi, M. Martiano, and J. Simarmata, "Comparison analysis of K-means and K-medoid with ecluidience distance algorithm, chanberra distance, and chebyshev distance for big data clustering," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 420, Oct. 2018, Art. no. 012092.

[52] M. Arlitt and T. Jin. (Aug. 1998). *1998 World Cup Web Site Access Logs*. [Online]. Available: http://www.acm.org/sigcomm/ITA/

[53] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Comput. Netw.*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009.

[54] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Gener. Comput. Syst.*, vol. 27, no. 6, pp. 871–879, Jun. 2011.

[55] F. Soppelsa and C. Kaewkasi, *Native Docker Clustering With Swarm*. Birmingham, U.K.: Packt, 2017.

[56] D. Mosberger and T. Jin, "Httperf: A tool for measuring Web server performance," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, Dec. 1998.

FATIMA TAHIR received the B.S. degree in software engineering from the University of the Punjab, Lahore, Pakistan, in 2018, where she is currently pursuing the M.Phil. degree in computer science. Her research interests include cloud computing, machine learning, system management, and distributed computing.

MUHAMMAD ABDULLAH received the B.S. and M.Phil. degrees in computer science from the University of the Punjab, Lahore, Pakistan, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree in computer science. He is currently a Lecturer with the Punjab University College of Information Technology, University of the Punjab. His research interests include cloud computing, machine learning, scalable applications, and system performance management.

FAISAL BUKHARI received the M.S. and Ph.D. degrees in computer science from the Asian Institute of Technology (AIT), Thailand, and the M.Sc. degree in statistics and the M.Sc. degree in computer science from University of the Punjab (PU), Lahore, Pakistan. He is currently an Assistant Professor with the Punjab University College of Information Technology (PUCIT), PU, and also supervising the Imaging and Data Science Laboratory. His research interests include computer vision, image processing, data science, and machine learning.

KHALED MOHAMAD ALMUSTAFA (Associate Member, IEEE) received the B.E.Sc. degree in electrical engineering, and the M.E.Sc. and Ph.D. degrees in wireless communication from the University of Western Ontario, London, ON, Canada, in 2003, 2004, and 2007 respectively. He was a General Supervisor of the Information Technology and Computer Services Center (ITCS), Prince Sultan University (PSU), Riyadh, Saudi Arabia, the Chairman of the Department of Communication and Networks Engineering (CME), PSU, and the Vice Dean of the College of Engineering, PSU. He is currently working as an Associate Professor with the Department of Information Systems (IS), College of Computer Science and Information Systems (CCIS), PSU, and the Director of the Research and Initiatives Center. His research interests include error performance evaluation of MIMO communication systems in partially known channels, adaptive modulation, and channel security, text recognition models, control systems with renewable energy applications as well as features selections and data prepossessing.

WAHEED IQBAL (Member, IEEE) received the dual master's degrees in computer science and information technology from the Asian Institute of Technology and the Technical University of Catalonia (UPC), Barcelona, Spain, in 2009, and the Ph.D. degree in computer science from the Asian Institute of Technology, Bangkok, Thailand, in 2012. He was a Postdoctoral Researcher with the College of Engineering, Qatar University, Doha, Qatar, from 2017 to 2018. Since 2014, he has been an Assistant Professor with the Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan. His research interests include big data, cloud computing, distributed systems, and machine learning.