Burst-Aware Predictive Autoscaling for Containerized Microservices

Muhammad Abdullah, Waheed Iqbal, Josep Ll. Berral, Jorda Polo, David Carrera

Abstract—Autoscaling methods are used for cloud-hosted applications to dynamically scale the allocated resources for guaranteeing Quality-of-Service (QoS). The public-facing application serves dynamic workloads, which contain bursts and pose challenges for autoscaling methods to ensure application performance. Existing State-of-the-art autoscaling methods are burst-oblivious to determine and provision the appropriate resources. For dynamic workloads, it is hard to detect and handle bursts online for maintaining application performance. In this paper, we propose a novel burst-aware autoscaling method which detects burst in dynamic workloads using workload forecasting, resource prediction, and scaling decision making while minimizing response time service-level objectives (SLO) violations. We evaluated our approach through a trace-driven simulation, using multiple synthetic and realistic bursty workloads for containerized microservices, improving performance when comparing against existing state-of-the-art autoscaling methods. Such experiments show an increase of $\times 1.09$ in total processed requests, a reduction of $\times 5.17$ for SLO violations, and an increase of $\times 0.767$ cost as compared to the baseline method.

Index Terms—Cloud Computing, Autoscaling, Burstiness, Microservices, Containers, Service-level Objectives, Response Time Guarantees

1 Introduction

Cloud computing [1] is widely used to deploy and manage applications over the last decade. Pay-as-you-go and dynamic resource provisioning are main features of cloud computing, which are attractive to many enterprises, small businesses, and individual users. Dynamic resource provisioning allows building efficient autoscaling methods for applications, which are required to satisfy specific response time service-level objectives (SLO) [2], [3], [4], [5]. The response time is one of the most critical SLO for user-facing applications as the applications with high response time are slow and decrease the users traction. Therefore, maintaining a good response time for the applications are important for cloud-hosted scalable applications to attract and retain users.

Cloud computing uses virtualization technology for hosting and managing applications. Traditionally, Hardware-level virtualization, also known as the hypervisor-based virtualization, is used to manage virtual machines (VMs) on cloud data centers. Recent advances in virtualization technology introduced containerization, also known as OS-level virtualization, which is getting traction mainly due to better portability, lightweight, and easy scalability compared to VM-based virtualization. Containerized virtualization is suitable for managing microservices based applications because it helps to quickly launch and terminate the containers for rapid scalability, whereas VM-based virtualization requires considerable time to start and terminate the VM [6], [7]. Autoscaling methods for cloud-hosted application developed using microservice architecture requires to maintain good response time for large concurrent user requests. Most of the existing state-of-the-art autoscaling methods are rule-based reactive methods [8], [9], [10], [11], [12], which scale the application resources based on specific events. For example, a typical reactive rule-based autoscaling algorithm can begin increasing the allocated resources whenever average CPU utilization crosses a particular threshold. There have been some efforts to build predictive autoscaling methods using machine learning techniques [13], [14] and analytical queuing [15]. A predictive autoscaling method proactively identifies the need for scaling the allocated resources and then use some intelligent technique to predict the number of resources required to serve the workload under acceptable response time.

Autoscaling decisions are sensitive to the application workload. For example, sudden and dramatic changes in a workload, also known as burstiness, cause autoscaling method to rapidly perform different resource allocation configurations for maintaining response time requirements. However, this introduces high oscillation in scaling decisions and increases response time SLO violations drastically. The existing autoscaling methods are burst-oblivious; therefore, they do not detect and mitigate burstiness to avoid application performance degradation. To show the burst-oblivious effect in autoscaling, we perform a small experiment using a CPU-intensive microservice which performs FFT for given digital signals with World Cup workload traces [16]. We used the autoscaling method proposed by Moreno et al. [15], which uses the Support Vector Regression (SVR) for workload forecasting and analytical queueing method to identify the required resources to serve the forecasted workload. This method identifies the maximum number of concurrent user requests that one container can serve without showing any performance issue and then use it to determine the required number of containers to serve the forecasted workload. After the identification of the re-

- *M. Abdullah and W. Iqbal are with the Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan. Email: {muhammad.abdullah, waheed.iqbal}@pucit.edu.pk.*
- *Josep Ll. Berral, Jorda Polo, and David Carrera are with the Barcelona Supercomputing Center (BSC) and Universitat Politcnica de Catalunya (UPC) - BarcelonaTECH, Spain Email: {josep.berral, jorda.polo, david.carrera}@bsc.es.*

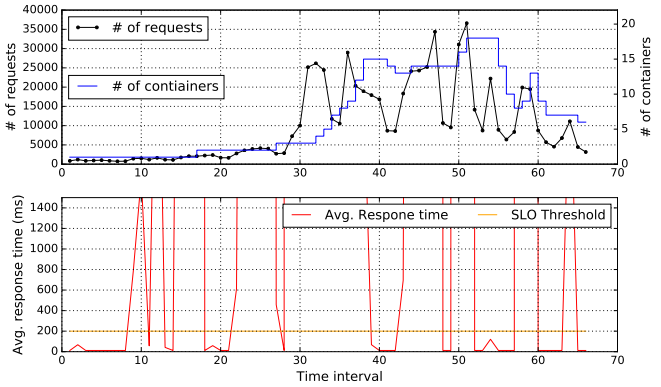


Fig. 1: The number of total requests, dynamic allocation of containers, and average response time using burst-oblivious autoscaling for the benchmark microservice under World Cup workload traces.

quired number of resources, the system dynamically scale-in or scale-out the container instances accordingly. To identify the capacity of one container, we performed a stress testing of the FFT microservice hosted on a container running over a Docker Swarm Cluster. After this experiment, we computed the SLO violations and total scale operations to measure the performance of the autoscaling method. Figure 1 shows the number of requests, dynamic scaling of containers, and average response time for this experiment. We observe 29.56% SLO violations and 22 scale operations during the experiment. We advocate the appropriate detection of the burst can help to decrease the SLO violations and also reduce the oscillation effect in scale operations.

Most of the existing workload burst detection methods use arrival rate patterns, for example, Ahmed et al. [17] and Mario et al. [18], and then employ some statistical or entropy-based technique to identify bursts. These techniques work offline and requiring entire workload traces to be available for detecting bursts. Building an online burst detection technique using arrival rate patterns introduce challenges to dynamically identify appropriate sliding window size and threshold values for statistical or entropy-based methods for the proper burst detection. Then additional efforts are required to train autoscaling methods to handle the burst gracefully for minimizing SLO violations.

In this paper, we propose a novel burst-aware autoscaling method which identifies workload bursts using its own scaling decisions, without considering arrival rate patterns, and then provision appropriate resources to maintain application performance for bursty workloads. In our proposed solution, first, a resource prediction model is trained using historical autoscaling performance traces. Second, the system forecast the workload arrival rate for the next time interval and then identify the resources required to satisfy response time SLO requirements. The system maintains a history of these resource predictions for last k intervals. Finally, the system identifies the bursty workload using the dispersion in these resource prediction and provision appropriate resources to the application. Our trace-driven simulation using a benchmark microservice and multiple synthetic and real bursty workloads shows excellent performance compared to the existing state-of-the-art autoscaling methods. We also validate our proposed solution on a containerized testbed infrastructure for the benchmark microservice, which shows 1.09x times increase in total processed requests and 5.17x times fewer SLO violations as compared to the baseline autoscaling method. Our method is sensitive to identify the workload burst and conservative to unmark the burst to minimize SLO violations. Therefore, we observe the overprovisioning of resources in certain situations by reducing the risk of underprovisioning. The main contributions of this paper include:

- Design a system for automatic resource provisioning of microservices to satisfy response time SLO requirements.
- Develop a novel burst-aware autoscaling algorithm to identify burstiness and provision appropriate resources dynamically.
- Experimental evaluation using trace-driven simulations under multiple synthetic and real bursty workload traces and compare with the existing state-of-the-art autoscaling techniques.
- Validate the proposed solution on a real containerized infrastructure for a benchmark microservice under the bursty workload and compared it with a baseline autoscaling method.

The rest of the paper is organized as follows. Related work is discussed in Section 2. We briefly explain the overall proposed system in Section 3. Resource prediction model and workload forecasting is presented in Section 4. Proposed Burst-aware autoscaling algorithm is discussed in Section 5. The experimental design is discussed in Section 6. Experimental results and evaluations are presented in Section 7. Finally, conclusions and future work are discussed in Section 8.

2 Related Work

Autoscaling methods are used to maximize the performance of cloud-hosted applications. There have been several research studies in the literature for autoscaling and dynamic provisioning of cloud-hosted applications [19], [20]. Most of these techniques are based on rule-based policies, analytical queuing, machine learning, and reinforcement learning. The autoscaling methods are triggered either reactively or proactively. For example, Liu et al. [21] proposed a reactive autoscaling method which monitors the bandwidth and CPU utilization to vertically scale resources whenever CPU or bandwidth utilization saturate. Michael et al. [22] presents the horizontal autoscaling of biomedical and bioinformatics cloud-hosted applications with a reactive method to enhance application performance. Liu et al. [23] presents a reactive autoscaling method which can dynamically adjust the scaling threshold and cluster sizes using fuzzy logic. Chieu et al. [24] presents an autoscaling method which reacts based on the number of active users. This work scales the application whenever an active user count goes beyond a certain threshold in all allocated resources.

There have been several efforts for building predictive autoscaling methods using machine learning techniques. For example, Radhika et al. [25] presents a predictive autoscaling method which uses a recurrent neural network (RNN) and autoregressive integrated moving average (ARIMA) to predict the CPU and memory usage and autoscale the application resources based on the prediction.

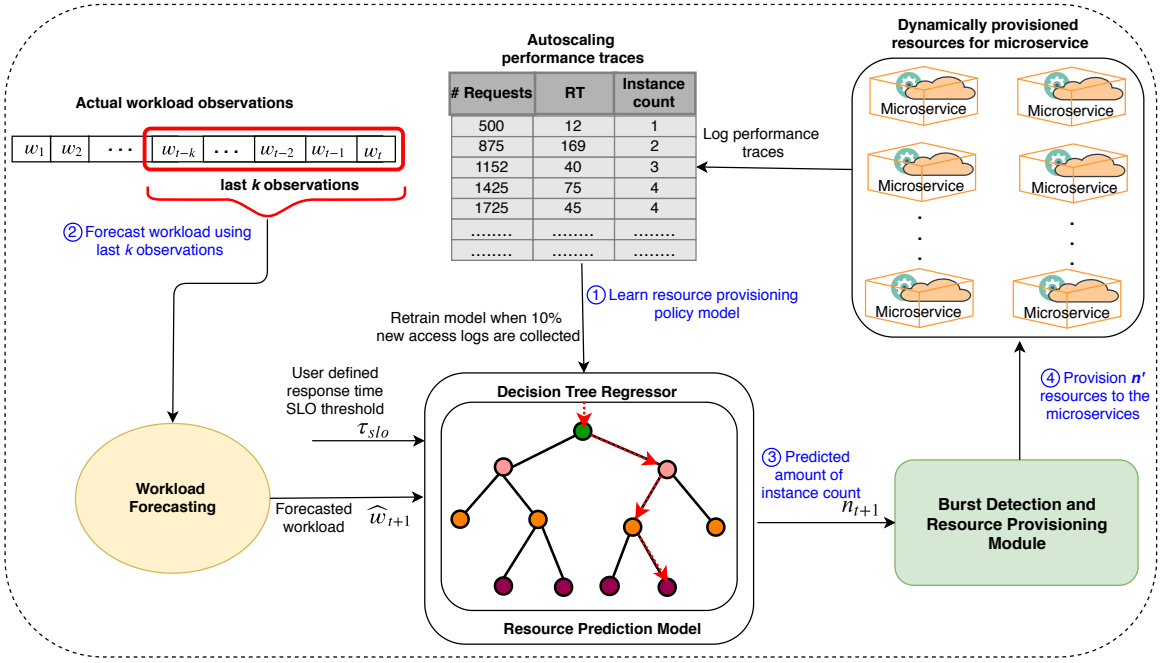


Fig. 2: The proposed system overview showing resource prediction model learning, workload forecasting, and dynamic provisioning of containers to microservices.

Maryam et al. [26] presents a survey on the predictive autoscaling methods and discuss the different machine learning algorithms used by the researcher for the dynamic resources provisioning. Several researchers are using reinforcement learning methods [13], [14] to learn resources provisioning policies. Some researchers are also using analytical queueing for the autoscaling of cloud-hosted applications. For example, Rafael et al. [15] use the SVM method for the workload forecasting and further use the analytical queueing model to find the number of resource instance to handle the forecasted workload. There have been some efforts to use fuzzy logic to build autoscaling controllers. For example, Valerio et al. [27] proposed a fuzzy logic controller to scale cloud resources allocated to the applications horizontally.

Recently, some efforts are made to autoscale microservices-based applications. For example, Abdullah et al. [28] use the reactive autoscaling method for the microservices automatically extracted from a monolithic application to improve the performance. Issaret et al. [29] presents a cost-effective autoscaling method for the cloud-hosted microservices. The proposed method uses artificial and recurrent neural networks to predict the workload and use optimization to allocate the resources to the microservice. Alipour et al. [30] present the resources provisioning method for the microservices using linear regression and multiclass classification. The proposed method predicts the future workload and future CPU demand using cluster CPU usage and provision of the resources accordingly. Xuxin et al. [31] presents the autoscaling framework for containerized applications by monitoring CPU utilization metrics.

There have been a few efforts to detect the burstiness in the workloads. For example, Ahmed et al. [17] use the sample entropy method to find the burst in the given cloud workload. Mario et al. [18] use average based prediction models to identify the burst behavior in the given workload. Both of these burst identification methods require entire workload traces available offline. AdaFrame [32] timely detects the change in cloud application workloads for minimizing the time to trigger autoscaling using a probabilistic approach based on CUSUM [33]. Once the workload change is detected, their proposed method scale-out by adding one virtual machine to handle the change. This strategy is not efficient in handling bursts, which required more than one virtual machine to provision at a time.

Online workload burst detection using real-time application traffic is a challenging task. Existing state-of-the-art autoscaling methods do not consider workload burstiness, which can drastically affect the application performance. We propose and evaluate an autoscaling method which can identify workload bursts using its own scaling decisions, without considering arrival rate patterns, and then provision appropriate resources for minimizing the response time SLO violations. The proposed solution shows excellent performance compared to the existing state-of-the-art autoscaling methods [3], [15], [24], [34], [35] under bursty workloads.

3 Proposed System Overview

The overall proposed system of burst-aware autoscaling for microservices is illustrated in Figure 2. The system workflow consists of the following steps:

1) The proposed system learns the predictive resources provisioning policy model using historical autoscaling performance traces for the target microservice. We propose to use a reactive autoscaling method initially to gather sufficient performance traces to learn resource prediction model for the first time. The resource prediction model learning is explained in Section 4.1.
2) Once, the resource prediction model is trained, the proposed system forecast the future workload using the

last k actual workload observations by the microservice. Then the forecasted workload is used to predict the number of containers required to satisfy the application response time SLO requirement using resource prediction model. At every specific time interval, the system forecast the workload and identify the required number of containers. We explain the workload forecasting method in Section 4.2.

3) The predicted number of container instances is used by the burst detection and resources provisioning module to identify the appropriate number of containers to provisions the microservice. Burst detection method uses last k number of predicted instances count by the resources prediction model. The variation in last k predicted amount of container instances detects the burstiness. The details about burst detection and overall autoscaling method is explained in Section 5.
4) Finally, the system automatically scales the number of allocated containers to the microservice with n' identify by the burst detection and resources provisioning module.

The proposed system retrains the resources prediction model whenever 10% new autoscaling performance traces are collected. This ensures to reflect the underlying hardware resources utilization behavior. The initial performance traces are important for the proposed system which can be collected using different autoscaling strategies including reactive, what-if policies, reinforcement learning, and analytical modeling. In the beginning, we employ reactive autoscaling to collect performance traces to train the initial resource prediction model. Then the system automatically retrains the model whenever the sufficient new performance traces are collected.

4 Resource Prediction and Workload Forecasting

In this section, we explain the learning of our proposed resource prediction model and workload forecasting method in the following subsections in turn.

4.1 Resource Prediction Model Learning

Initially, the resource prediction model training requires some initial performance traces using a typical autoscaling strategy. Then these initial performance traces can be used to identify the appropriate machine learning algorithm to be used in the resource prediction model. We explain the methodology to gather initial performance traces, learning model with different machine learning algorithms, and evaluation results for the resource prediction model are discussed in the following subsections.

4.1.1 Initial Autoscaling Performance Traces Collection

To collect the initial autoscaling performance traces, we deployed the benchmark microservice, explained in Section 6.2, on a containerized infrastructure. The containerized cluster consists of five homogeneous nodes with core i-7 8-core CPU, 16 GB physical memory, and 2 TB disk. Four nodes are used as Swarm workers, and one node is used as a Swarm manager. Each container uses 1 CPU core and 2 GB physical memory. We generated linearly increasing synthetic workload using httperf [36]. The workload makes concurrent user requests in a step-up fashion to dramatically saturate the response time of the microservice. Each workload request to the benchmark application requires Fast Fourier Transform for a 200 random samples. The performance traces are recorded during this experiment and used to model the response time of the benchmark application. We model the response time using an exponential curve fitting technique, which is also used in our recent work [37]. The response time model is an exponential curve fitted using the performance traces capable of giving the response time for any given number of requests for the benchmark application. Using this response time model, we performed a trace-driven simulation to collect the autoscaling performance traces using a reactive policy for increasing workload. The reactive policy dynamically adds one container instance whenever 95^{th} percentile of the microservice response time goes higher than 200 milliseconds. This approach appropriately enables to collect sufficient autoscaling performance traces to build the initial resource prediction model for any target application.

4.1.2 Model Learning

We use the initially collected autoscaling performance traces to learn the predictive resources provisioning model and evaluate different machine learning methods to use as the model learning algorithm. To prepare the data set for evaluating different machine learning algorithm, we discard all those performance traces showing response time SLO violations. For each time interval, we use the number of requests and 95^{th} percentile of response time as input parameters and then use the number of container instance count as output which can satisfy a specific response time SLO requirements. Equation 1 shows our resource prediction model:

$$\varphi: (w, \tau_{slo}) \rightarrow n, \tag{1}$$

where φ represents the trained model which can predict the required number of container instance count n for a given number of requests w and a desired response time τ_{slo}.

We evaluate various state-of-the-art machine learning method including Linear Regression (LR), Polynomial Regression (PR), Elastic Net (EN) regression, XGBoost (XGB) regression, Random Decision Forests (RDF) regression, Decision Tree Regression (DTR), Support Vector regression (SVR), and Multi-layer Perceptron regression (MLP) for learning an adequate resources prediction model. We split the data set into 80% training and 20% testing sets and computed Mean Square Error (MSE) for the resource prediction model trained using different machine learning methods. The evaluation results are presented in Table 1. We select the machine learning method with minimum MSE on testing data in our resources prediction model. We found DTR outperforms all other methods and yield minimum MSE using the autoscaling performance traces data set.

Decision Tree Regression is a supervised machine learning algorithm commonly used to address regression problems by constructing a model on decision rules in the form of a tree structure. Comparing to the other machine learning algorithms, DTR can be trained with less number of training data and without normalization. The important decision

TABLE 1: Prediction error on the test data set for learning resource prediction model.

Method	LR	EN	PR	XGB	RDF	DTR	SVR	MLP
MSE	0.434	0.453	0.453	0.129	0.136	**0.009**	65.816	8.103

tree implementations include Iterative Dichotomiser 3 (ID3) and Classification And Regression Trees (CART). We use CART implementation of DTR in the current work.

For our CART implementation, we consider the data on a node k of the tree is represented as d. For each node, we split the data into two subset $d_L(\theta)$ and $d_R(\theta)$, where $\theta = (x, t_x^k)$ and x represents the input feature and t_x^k represents the threshold of that feature. The cost function of DTR is given by:

$$\mathbf{\Gamma}(D, \theta) = \frac{N_L}{N} H(d_L(\theta)) + \frac{N_R}{N} H(d_R(\theta)), \tag{2}$$

where $N_L + N_R = N$ and H is the impurity function which is calculated using the following equation:

$$H(X_k) = \frac{1}{N} \sum_{i=1}^{N} (n_i - \mu_N)^2, \tag{3}$$

where X_k is the training data at given node k, $\mu_N = \frac{1}{N} \sum_{i=1}^{N} n_i$, n_i is the actual result from training data and N is the total number of training data samples. We select the features to split data for minimizing the impurity:

$$\theta^* = \text{argmin}_\theta \mathbf{\Gamma}(d, \theta). \tag{4}$$

4.1.3 Model Evaluation

Table 1 shows the MSE on test dataset for using different regression methods for training the resource prediction model. The resource prediction model predicts the count of container instances required to satisfy the given workload with SLO guarantees. SVR and MLP perform worst by yielding highest MSE as compared to other regression methods. DTR outperforms all other regression methods by producing a minimum error. Therefore, we use DTR as a learning method in our resource prediction model.

4.2 Workload Forecasting

We use the forecasted workload in our autoscaling method for identifying appropriate resources for allocating to the microservice. Our propose workload forecasting method predicts the workload $\widehat{w}_{t+1}$ for the next interval $t+1$ using the last k actual workload observations. At any time interval, a workload is a number of requests also known as arrival rate. We trained different machine learning (ML) regression models including Linear Regression (LR), Elastic Net Regression (EN), Lasso Regression, Ridge Regression and Support Vector Regression (SVR) to predict the future workload for the different values of k including 5, 10, 15, and 20. Other possibilities to use for workload forecasting includes ARIMA and RNN; however, these models need a large number of historical observations to train the parameters for yielding high accuracy. Therefore, we only considered regression techniques to evaluate the workload forecasting. Figure 3 shows the normalized mean absolute error (MAE) of different ML regression models using different values of k for the Calgary and World Cup workload.

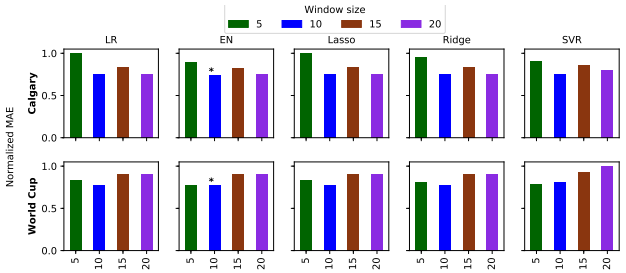


Fig. 3: Normalized Mean Absolute Error (MAE) of different ML models for the World Cup and Calgary workloads.

In our solution, we select EN as the workload forecasting method as it yields the minimum MAE compared to other regression methods for window size 10.

Elastic Net (EN) regression model use the last k workload observations at every time interval and predicts the next interval workload. Let the observed workload for the last k intervals be $\mathbf{w}_{t-k}^t = [w_{t-k} \cdots w_{t-2}\ w_{t-1}\ w_t]^\top$. Considering that the variations in the workload are locally linear during the last k intervals, we can estimate the arrival rate for the next interval by using the following equation:

$$\widehat{w}_{t+1} = \theta_0 + \theta_1 w_t, \tag{5}$$

where the regression parameters θ_0 and θ_1 are estimated over the last k intervals using following objective function:

$$\begin{aligned} \min_{\theta_0, \theta_1} & \sum_{i=0}^{k-1} ||w_{t-i} - \theta_0 - \theta_1 w_{t-i-1}||_2^2 \\ & + \lambda \left(\rho \sum_{j=0}^{1} ||\theta_j||_1 + \frac{1-\rho}{2} \sum_{j=0}^{1} ||\theta_j||_2^2 \right), \end{aligned} \tag{6}$$

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 θ_0 and θ_1 are learned, we can estimate arrival rate for the next time interval as

$$\widehat{w}_{t+1} = \begin{pmatrix} 1 \\ w_t \end{pmatrix}^\top \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}. \tag{7}$$

5 Burst-aware Autoscaling Algorithm

Algorithm 1 shows the proposed burst-aware autoscaling method. The algorithm accepts the following inputs: an application monitoring interval (ξ), a response time SLO threshold (τ_{slo}), a window size for workload forecasting and burst detection (k), a trained resources prediction model (φ), and a list of currently allocated container instances ($\mathbb{M}$) to the microservice.

In each iteration, the system waits for ξ seconds to collect sufficient application access logs which represent workload for the current interval. The system forecasts $\widehat{w}_{t+1}$ the workload for the next interval $t+1$ using the last k workload observations $w_t, w_{t-1}, \cdots, w_{t-k}$. After the workload forecasting, the resource prediction model φ predicts the number of containers n_{t+1} to satisfy the given response time SLO threshold τ_{slo} for the forecasted workload $\widehat{w}_{t+1}$. The system maintains a list $(n_t, n_{t-1}, \cdots, n_{t-k})$ of last k predictions for number of containers. Then, the algorithm

Algorithm 1: Proposed Burst-aware Autoscaling Method.

Input: Application monitoring interval (ξ) in seconds, response time SLO threshold (τ_{slo}), window size (k) for workload forecasting and burst detection, resource prediction model φ: $(w, \tau_{slo}) \rightarrow n$, list of allocated container instances ($\mathbb{M}$) to the microservice.

Output: Updated list of allocated container instances $\mathbb{M}^{t+1}$ to the microservice.

$isBurst \leftarrow false$
$n_{bb} \leftarrow 0$
while *true* **do**
  Wait for ξ seconds
  $\widehat{w}_{t+1} \leftarrow$ forecast the workload using $w_t, w_{t-1}, \cdots, w_{t-k}$ observed workload.
  $n_{t+1} \leftarrow \varphi(\widehat{w}_{t+1}, \tau_{slo})$ ▷ Predict resources for the forecasted workload.
  $\sigma_{max} \leftarrow 0$
  $n_{max} \leftarrow 0$
  for $i = 1$ *to* k **do**
    Calculate standard deviation σ_i of data $n_{t+1}, n_t, \cdots, n_{t-i}$
    if $\sigma_i > \sigma_{max}$ **then**
      $\sigma_{max} = \sigma_i$
      $n_{max} = Max(n_{t+1}, n_t, \cdots, n_{t-i})$
    end
  end
  if $\sigma_{max} \geq 2$ & $isBurst == false$ **then**
    $n^{'} \leftarrow n_{max}$
    $isBurst \leftarrow true$
    $n_{bb} = nt$
  else if $\sigma_{max} \geq 2$ & $isBurst == true$ **then**
    $n^{'} \leftarrow n_{max}$
  else if $\sigma_{max} < 2$ & $isBurst == true$ **then**
    if $n_{bb} > n_{t+1}$ **then**
      $n^{'} \leftarrow n_{t+1}$
      $isBurst \leftarrow false$
      $n_{bb} \leftarrow 0$
    else
      $n^{'} \leftarrow n_{max}$
    end
  else if $\sigma_{max} < 2$ & $isBurst == false$ **then**
    $n^{'} \leftarrow n_{t+1}$
  end
  if $|\mathbb{M}| < n^{'}$ **then**
    $\mathbb{M}^{+}$ ▷ Scale-out the microservice by adding $n^{'} - |\mathbb{M}|$ containers.
  else if $|\mathbb{M}| > n^{'}$ **then**
    $\mathbb{M}^{-}$ ▷ Scale-in the microservice by removing $|\mathbb{M}| - n^{'}$ containerss.
  end
  $t \leftarrow t + 1$
end

identifies maximum standard deviation σ_{max} from different window sizes on list $n_{t+1}, n_t, \cdots, n_{t-i}$ where i start from 1 to k. This identifies the dispersion in autoscaling predictions for different window sizes and can help to detect burst. If σ_{max} is greater than 2, which shows the burstiness in the current workload, the algorithm sets the burst mode to ON and provision the maximum number of instances used from the window which yielded σ_{max}. If σ_{max} is less than 2 and the predictive instances count n_{t+1} is also less than the allocated instance count which is observed just before the burst, then the algorithm dynamically sets the burst mode to OFF state and adjust the allocated resources to the microservice with the predicted container instance count.

6 Experimental Design

We evaluate the proposed burst-aware autoscaling method through trace-driven simulations for a benchmark application and then we also validate our experimental results using a real containerized testbed infrastructure. We employ multiple synthetic and real workload traces to compare the performance of the proposed autoscaling method and then compare it with multiple existing state-of-the-art autoscaling methods, as explained in Section 6.1. In this Section, we describe the benchmark application, workloads, and evaluation criteria in the following subsections.

6.1 Baseline Autoscaling Methods

There have been several autoscaling methods proposed by different researchers to improve application performance by reducing the response time latency and satisfying the response time SLO. Mostly, autoscaling methods reactively decide the need for autoscaling and then provision the resources using rule-based policies. There are also some autoscaling methods using a proactive approach to scale applications. We use four different state-of-the-art autoscaling methods, which represent both reactive and predictive autoscaling methods commonly used in the literature, as a baseline to compare our burst-aware autoscaling method for different realistic and synthetic workloads. The selected baseline autoscaling methods are among the most highly-cited in autoscaling groups represented in the survey [38]. The baseline autoscaling methods are briefly explained in the following subsections.

6.1.1 Reactive Autoscaling Method (React)

We implemented the reactive autoscaling method similar to Chieu et al. [24]. This algorithm dynamically scales the VMs for the application by monitoring the number of requests, active user sessions, and average response time per request of the application. We refer to this autoscaling method as ***React*** in the rest of the paper. Theis autoscaling method determines the VM instances count using predefined maximum user sessions per instance and currently active user session in an instance. The autoscaling method finds the ratio of currently active users sessions in an instance and maximum users sessions per instances and compared it with predefined thresholds for dynamically provisioning.

6.1.2 Hybrid Autoscaling Method (Hybrid)

Ali-Eldin et al. [34], [35] proposed a hybrid autoscaling method using a reactive and proactive approach to dynamically provision the VM resources for a service. We refer to this autoscaling method as ***Hybrid*** in the rest of the paper. This method determines the future workload using change in the arrival rate of requests and dynamically provision the resources to prevent the SLO violations and oscillations in autoscaling decisions. This method improves service performance by decreasing the number of delayed requests.

6.1.3 Regression-based Autoscaling Method (Reg)

Iqbal et al [3] proposed a regression-based autoscaling method to adjust the VM resources of cloud-hosted multi-tier applications dynamically. This autoscaling method uses a reactive approach for the scale-out decisions and uses a regression-based predictive component for the scale-in decisions to dynamically provision the VM resources of a cloud-hosted multi-tier application. This autoscaling method scale-out the VM resources whenever the response time of the application increases from the predefined response time threshold and use a regression model to predict the future workload, which is used for the scale-in decisions. We refer to this autoscaling method as ***Reg*** in the rest of the paper.

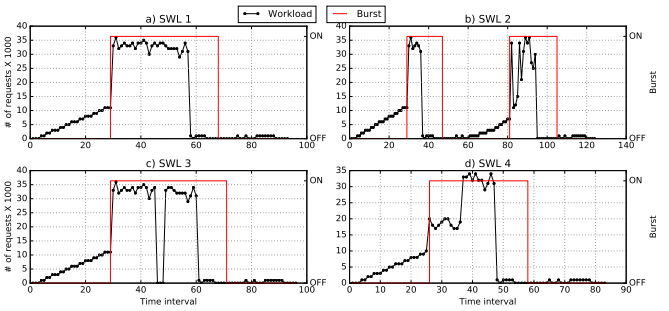


Fig. 4: Synthetic workloads with detected bursts using the proposed autoscaling method used in the experimental evaluation.

6.1.4 Analytical Queue-based Autoscaling Method (Analytical)

A recent autoscaling method proposed by Moreno et al. [15] optimizes the service response time and fulfill services level agreement between cloud services provider and user by minimizing the SLO violations. This autoscaling method uses the Support Vector Machine (SVM) to predict the future workload and then use the analytical queuing model to determine the number of resources needed for serving the predicted workload under specific response time guarantees. We refer to this autoscaling method as ***Analytical*** in the rest of the paper.

6.2 Benchmark Application

Typically, microservices are designed to perform fine-grained tasks, for example, audio/video content conversion, searching algorithms, batch processing, and machine learning algorithms. We used Fast Fourier Transformation (FFT) [39] algorithm implemented in PHP as a benchmark microservice for experimental evaluations. The microservice computes and outputs Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) for digital signals. This microservice emulates a typical CPU-bound application workload. We generate concurrent digital signals as a workload for the microservice by following specific workload traces, explained in the next subsection.

6.3 Workloads

We use four synthetic bursty workloads and five real workloads to evaluate and compare the proposed burst-aware autoscaling method. Figure 4 shows the synthetic workload 1 (SWL 1), synthetic workload 2 (SWL 2), synthetic workload 3 (SWL 3), and synthetic workload 4 (SWL 4) which emulates different burstiness behaviors. The figure also shows the burst detected by the proposed autoscaling method. Our method shows one burst in SWL 1, two bursts in SWL 2, first from time interval 20 to 37 time-interval and second from 72 to 95 time-interval whereas, SWL 3 and SWL 4 show only one burst during 30 to 71 time-interval and 17 to 48 time-interval respectively.

The web traces of Wikipedia [40], FIFA World Cup [16], a department-level Web server at the University of Calgary (Department of Computer Science), the Web server at NASAs Kennedy Space Center, and the Web server from ClarkNet workload traces are used as real workload traces in the experimental evaluation. All of these web traces are available publicly[1]. Figure 5 shows the real workload traces

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

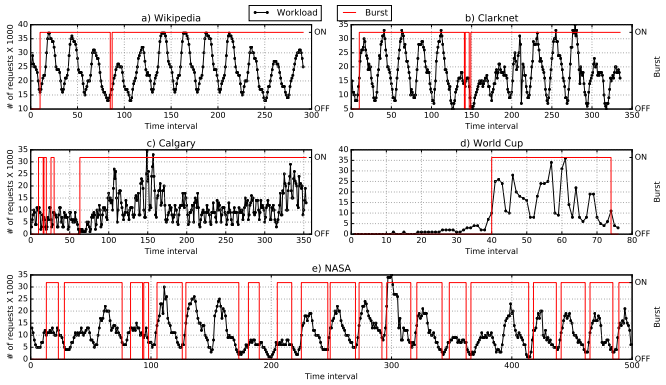


Fig. 5: Real workloads with detected bursts using the proposed autoscaling method used in the experimental evaluation.

used in the experimental evaluations with burst detected using the proposed burst detection method. NASA workload has many small bursts whereas other workloads show few large bursts detected by the proposed autoscaling method.

6.4 Evaluation Criteria

To compare the proposed autoscaling method with existing state-of-the-art methods, we compute *processed requests*, *response time SLO violations*, *scale operations*, and the *cost* for all the experiments. The *processed requests* shows the number of requests served by the application for the given workload, *response time SLO violations* show the percentage of requests used higher response time than desired, and *scale operations* represent the number of times autoscaling method added or removed resources to the application. An autoscaling method yields a higher number of total processed requests, less number of response time SLO violations, and less number of scale operations is considered the best method among the others. To compute the *cost* of the autosclaing method, we use 0.0014\$ per minute cost for each container similar to AWS (Amazon Web Services) *c5.large* instance.

6.5 Experimental Parameters

In our experimental evaluations, to select the value of application monitoring interval ξ used in our proposed algorithm, we run the proposed autoscaling method on real-infrastructure using three different values of ξ, including 60, 90, and 120 seconds for linearly increasing workload. Figure 6 shows the SLO violations % for the different values of monitoring interval ξ. We select the value of ξ, which shows minimum SLO violations. We use application monitoring interval $\xi = 60$ seconds as it shows minimum SLO violations. Further, we use SLO threshold $\tau_{slo} = 200$ milliseconds which user-defined parameter. The lower value is harder for autoscaling methods to ensure for a specific application. We choose $\tau_{slo} = 200$ to be a reasonable requirement for the application to offer to the users. We use window size for workload forecasting and burst detection $k = 10$ because it shows minimum MAE for workload forecasting, as discussed in Section 4.2. The value of k can affect the performance of the burst detection method. For example, the larger value of k enables the burst detection algorithm to search for a large number of window size combinations. Whereas the small value of k bias the algorithm for detecting

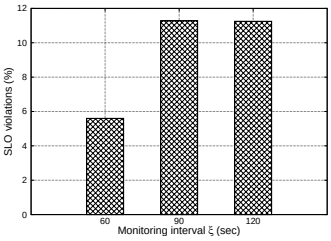


Fig. 6: SLO violations % for different values of monitoring interval ξ.

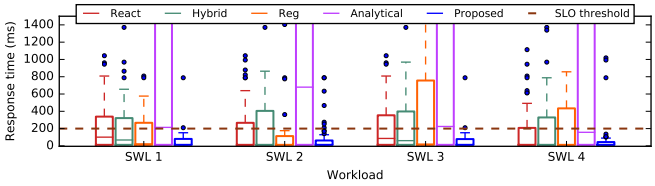


Fig. 7: Response time behavior of the microservice for different autoscaling methods under synthetic workload traces.

only local and short bursts. Therefore, we choose $k = 10$, which considers the last 10 intervals (50 minutes) workload and identify the burst.

7 Results and Evaluations

Table 2 shows the results for different workload traces and autoscaling methods including baseline and the proposed burst-aware autoscaling methods. We compute the percentage of the total *processed requests*, percentage of the total *SLO violations*, the total number of *scale operations*, and the *cost* for all synthetic and real workloads for each baseline autoscaling method and our proposed autoscaling method. The proposed autoscaling method yields better performance to minimize SLO violations and maximize processed requests compared to all the existing state-of-the-art autoscaling methods for different workload traces except for NASA workload traces. For NASA workload, React and Hybrid autoscaling methods show a slightly higher number of processed requests and less number of SLO violations compared to the proposed autoscaling method. Whereas, the proposed method shows less number of scale operations in NASA workload as compared to all other autoscaling methods. NASA workload is challenging as it contains many small irregular bursts which are very hard to handle with using minimal scale operations. Therefore, our technique minimized the scale operations for NASA workload, which shows some performance overhead compared to React and Hybrid autoscaling methods. Overall, the proposed solution shows excellent performance over the existing state-of-the-art autoscaling methods for different synthetic and real workload traces.

To show the response time behavior of the benchmark application under different workload traces for using the proposed and baseline autoscaling methods, we draw box

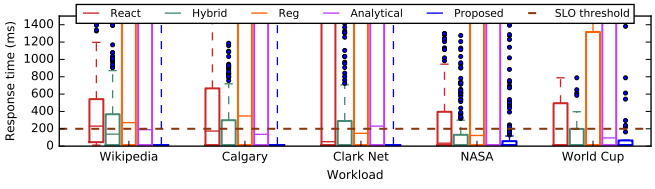


Fig. 8: Response time behavior of the microservice for different autoscaling methods under realistic workload traces.

TABLE 2: Trace-driven simulation results. Percentage of the total number of processed requests, percentage of total SLO violations, the total number of scale operations, and the cost for all workloads using baseline React, Hybrid, Reg, Analytical, and the proposed autoscaling methods.

Workload	Processed Requests (%)					SLO violations (%)					Scale Operations					Cost ($)				
	React	Hybrid	Reg	Analytical	Proposed	React	Hybrid	Reg	Analytical	Proposed	React	Hybrid	Reg	Analytical	Proposed	React	Hybrid	Reg	Analytical	Proposed
SWL 1	97.03	94.13	85.18	86.10	**97.08**	2.97	5.87	14.82	13.90	**2.92**	18	38	53	16	**12**	1.25	1.74	1.49	**1.23**	1.75
SWL 2	89.73	85.64	72.81	63.08	**91.71**	10.27	14.36	27.19	36.92	**8.29**	27	58	71	30	**23**	1.00	1.90	1.23	**0.95**	1.90
SWL 3	94.14	93.79	82.86	86.01	**97.09**	5.86	6.21	17.14	13.99	**2.91**	21	42	61	16	**12**	**1.25**	1.86	1.55	1.33	1.87
SWL 4	95.75	94.49	87.45	78.84	**96.39**	4.25	5.51	12.55	21.16	**3.61**	**14**	34	51	16	16	0.82	1.36	1.13	**0.79**	1.31
Wikipedia	98.02	98.66	90.59	88.49	**99.86**	1.98	1.34	9.41	11.51	**0.14**	138	123	270	142	**15**	7.62	8.82	7.59	**7.45**	10.70
Clark Net	96.22	97.62	80.52	83.21	**99.84**	3.78	2.38	19.48	16.79	**0.16**	174	138	300	169	**20**	6.64	8.53	**5.84**	6.43	11.45
Calgary	89.73	95.38	90.22	87.03	**98.17**	10.27	4.62	9.78	12.97	**1.83**	196	164	253	136	**40**	4.34	5.48	4.45	**4.21**	8.22
NASA	**95.82**	94.51	75.80	78.77	93.51	**4.18**	5.49	24.20	21.23	6.49	186	201	427	210	**162**	6.59	9.02	**5.35**	6.08	10.43
World Cup	83.13	89.52	78.77	70.44	**94.12**	16.87	10.48	21.23	29.56	**5.88**	29	30	39	22	**11**	0.75	1.20	0.72	**0.63**	1.22

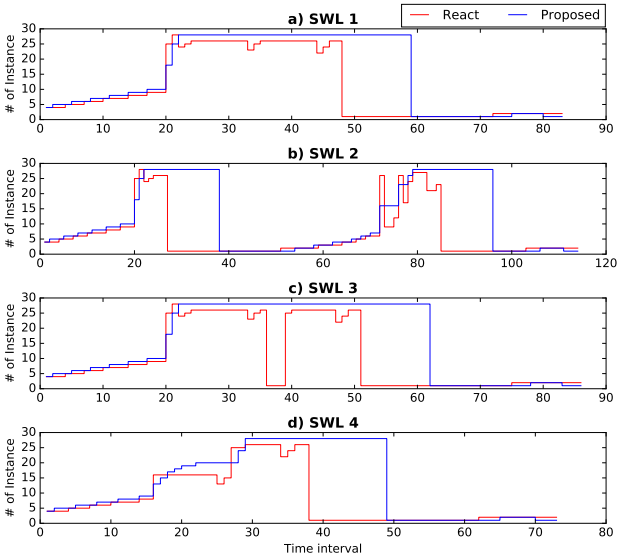


Fig. 9: Oscillation effect due to dynamic resource allocation using React and the proposed autoscaling methods for synthetic workload traces.

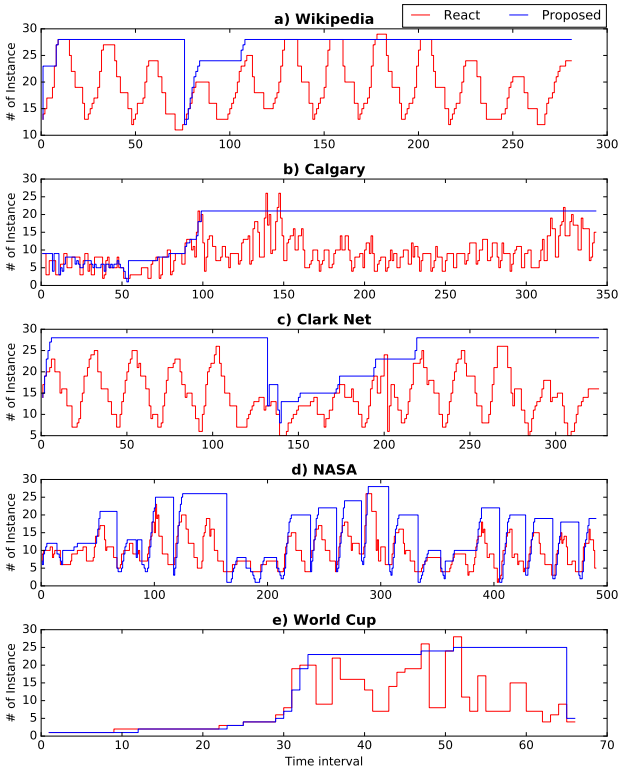


Fig. 10: Oscillation effect due to dynamic resource allocation using React and the proposed autoscaling methods for realistic workload traces.

plots. Figure 7 and 8 show the box plots of response time for synthetic and real workload traces respectively using different autoscaling methods. The proposed burst-aware autoscaling method keeps the response time for the application under the desired SLO threshold for all synthetic and real workload traces. However, some outliers show response time SLO violations which are significantly lower to the other autoscaling methods.

The proposed autoscaling method minimizes the oscillation effect to dynamically allocating the resources for different workload traces, which helps to reduce the response time SLO violations. We show the dynamic resource scaling for the proposed autoscaling method and compare it with React autoscaling technique. Figure 9 and 10 show the dynamic allocation of resources (scale operations) over time for synthetic and realistic workload traces using the proposed and baseline React autoscaling method. The oscillation effect using React autoscaling is significantly higher compared to the proposed autoscaling method. The oscillations in adding and removing resources to the application introduce performance overhead and therefore significantly increase the response time SLO violations. The proposed autoscaling method is efficient to minimize the oscillation effect for allocating resources to the application, which helps to reduce the response time SLO violations significantly.

Figure 11 shows the average processed requests, response time SLO violations, and scale operations of all workloads using different autoscaling methods. The proposed autoscaling method shows maximum processed requests, minimum response time SLO violations, and minimum scale operations on average for all workload traces compared to other state-of-the-art existing autoscaling methods. Specifically, the proposed method yields 1.03x, 1.02x, 1.16x, and 1.20x times higher processed requests as compared to the React, Hybrid, Reg, and Analytical autoscaling methods respectively for all workloads. Whereas, response time SLO violations are 1.87x, 1.75x, 4.83x, 5.52x times minimized over React, Hybrid, Reg, and Analytical respectively. The scale operations count are 2.58x, 2.66x, 4.9x, 2.43x times reduced by the proposed method over React, Hybrid, Reg, and Analytical methods respectively. The proposed method yields 1.62x, 1.22x, 1.66x, and 1.68x times higher cost as compared to the React, Hybrid, Reg, and Analytical autoscaling methods respectively for all workloads.

7.1 Cost Analysis of Proposed Method

To compare the cost analysis of the proposed method, we use two different approaches. First, we introduce a

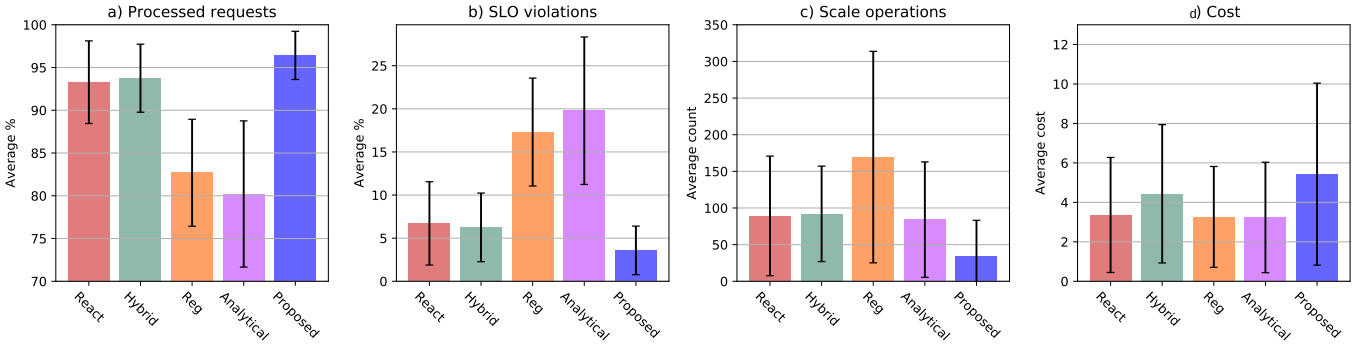


Fig. 11: Processed requests, SLO violations, scale operations, and cost comparison of proposed autoscaling method with all baseline methods using average of all workloads.

TABLE 3: SLO violations and Cost comparison.

Workload	SLO violations (%) / Cost ($)		
	Proposed	**Always_ON**	**Static_OP**
SWL1	2.92 / 1.25	2.87 / 2.65	0.00 / 3.25
SWL2	8.29 / 1.90	3.71 / 3.87	0.00 / 4.47
SWL3	2.91 / 1.87	2.86 / 2.77	0.00 / 3.37
SWL4	3.61 / 1.31	3.54 / 2.21	0.00 / 2.86
Wikipedia	0.14 / 10.70	0.08 / 10.94	0.00 / 11.02
Clark Net	0.16 / 11.45	0.16 / 12.68	0.00 / 12.74
Calgary	1.83 / 4.34	0.84 / 8.49	0.35 / 10.08
NASA	6.49 / 10.43	0.17 / 17.00	0.00 / 17.84
World Cup	5.88 / 1.22	5.88 / 1.25	0.53 / 2.31 1

new autoscaling method by changing the proposed method, which ON the burst for each time interval after the first detection of bursts. We name this approach as *Always_ON*. Second, we compare the proposed method with the static over-provisioned allocation of resource instances. In this method, we always initiate the maximum number of resources instance to the server the workloads. We name this method as *Static_OP*.

Table 3 shows the SLO violations and cost of the proposed, Always_ON, and Static_OP methods for synthetic and real workloads. Figure 12 and 13 shows the cost comparison of proposed, Always_ON, and Static_OP methods for synthetic and real workloads, respectively. Always_ON methods yield 51%, 103%, 48%, and 68%, whereas Static_OP method yields 85%, 135%, 80%, and 117% higher cost as compared to the proposed method for the SWL1, SWL2, SWL3, and SWL4 workloads respectively. Similarly, Always_ON methods yields 3%, 11%, 4%, 62%, and 3%, whereas Static_OP method yield 3%, 12%, 23%, 71%, and 89% higher cost as compared to the proposed method for the Wikipedia, Clark Net, Calgary, NASA, and World Cup workloads respectively.

7.2 Validation Experiments

To validate the proposed autoscaling method, we performed experiments on a Docker Swarm Cluster [41] using the proposed autoscaling method and compared it with *React*. The testbed cluster consists of five homogeneous nodes with core i-7 8-core CPU, 16 GB physical memory, and 2 TB disk. Four nodes are used as Swarm workers, and one node is used as a Swarm manager. Each container uses 1 CPU core and 2 GB physical memory. We use World Cup workload traces to emulate concurrent requests for the benchmark microservice using httperf [36]. However, we scale the workload traces to a maximum of 15,000 requests per minute.

Figure 14a shows the total number of processed requests, dynamic allocation of containers, and average response time using the React autoscaling method during the validation experiments for World Cup workload. Figure 14b shows the total number of processed requests, dynamic allocation of containers, and average response time using the proposed autoscaling method during the validation experiments for World Cup workload. The response time SLO violations are higher compared to the proposed autoscaling method. We only observed SLO violations during time interval 30th to 34th using the proposed autoscaling methods. Whereas, React shows multiple SLO violations during 24th to 65th time interval. The baseline autoscaling method also yields higher oscillation in scaling decisions compared to the proposed method. Figure 14b results show how our method displays inertia due to the observation time window. Our proposed method requires a certain "critical" amount of samples to consider that the workload is in a burst scenario also after a burst scenario, it requires a certain amount of samples to return to normal. This "time to reaction" is also shown in other methods (e.g., the React method in Figure 14a), also it must be considered that random occurrence cannot detect lone bursts (and "first bursts"). As we can see in Figure 14b, during the first burst, our technique already detects the increasing trend, but it is not until instance 34^{th} that the method recognizes the burst scenario, maintaining it during all its period. In comparison to the "React" method, we can deal with SLA with the only inconvenience of the first hard-to-predict "burst".

Table 4 summarizes the results for validation experiments to compare our solution with baseline React autoscaling method. Our technique increases the count of processed requests by 1.09x times as compared to the baseline method. The proposed method also decreases the SLO violations by 5.17x times as compared to the baseline with only taking 0.767x times extra cost. However, the total scale operations during the validation experiments remain the same for both autoscaling methods. The validation experiment shows the effectiveness of the proposed system on a real containerized infrastructure.

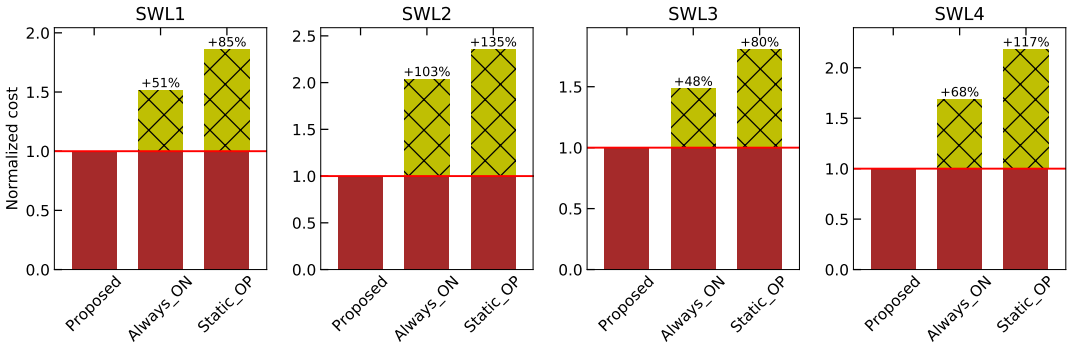


Fig. 12: Cost comparison of the proposed autoscaling method with Always_ON and Static_OP methods for synthetic workloads.

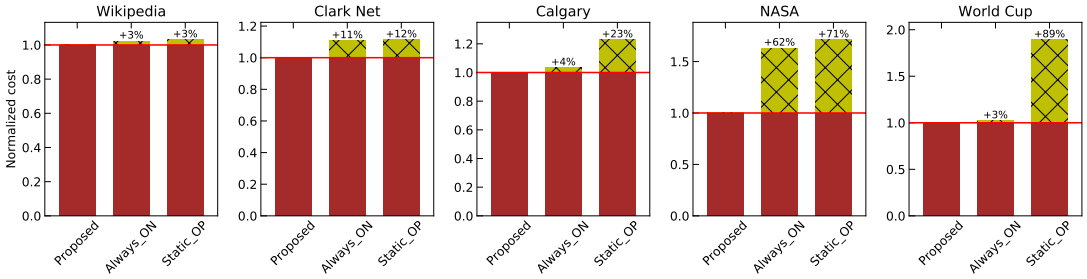
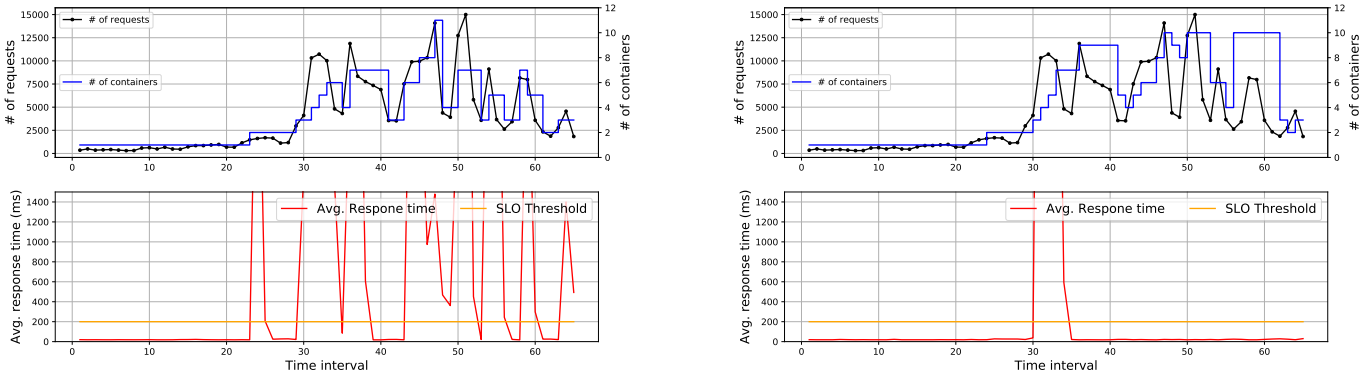


Fig. 13: Cost comparison of the proposed autoscaling method with Always_ON and Static_OP methods for real workloads.



(a) React Autoscaling Method

(b) Proposed Autoscaling Method.

Fig. 14: Total number of processed requests, dynamic allocation of containers, and average response time for React and proposed autoscaling methods during the validation experiment under World Cup workload.

7.3 Discussion

We extensively evaluate the effectiveness of the proposed solution using a single microservice under different dynamic and unseen workloads. The proposed solution can be applied to applications consisting of multiple microservices by deploying the solution independently for each of the microservice. To achieve this, we need to collect the performance traces of each microservice and then build the resource provisioning policy models for each of the microservice.

The evaluation is performed for a CPU-bound microservice; however, the proposed solution is generic and can be applied to other resource-bound microservices. The proposed method scales the microservice horizontally; therefore, any resource saturation can be avoided by adding more containers to the microservice and then load balance the workload.

Moreover, if the hardware infrastructure is changed, then the proposed solution needs to stress test the microservices on the target infrastructure and collect the performance traces for retraining the resources prediction model. Otherwise, the resource provisioning model will not be able to predict the appropriate resources for the microservice.

TABLE 4: Summary for validation experiments. Percentage of the total number of processed requests, percentage of total SLO violations, the total number of scale operations, and Cost for World Cup workload using the baseline React and the proposed autoscaling methods.

Evaluation Metric	Baseline	Proposed
Processed Requests (%)	90.64	98.69
SLO violations (%)	28.98	5.6
Sclae Operations	20	20
Cost ($)	0.309	0.403

8 Conclusion and Future Work

Burst detection in real-time workloads is a challenging problem because it can drastically degrade the performance of the application even in the presence of autoscaling methods. In this paper, we propose a novel burst-aware au-

toscaling method which identifies burstiness in workloads using its own scaling decisions and intelligently provisions the resources to the application for minimizing the response time SLO violations. Our extensive trace-driven simulation shows the excellent performance of the proposed burst-aware autoscaling method as compared to the different existing state-of-the-art autoscaling methods under the various realistic and synthetic bursty workloads. Our experimental evaluation on real testbed infrastructure using benchmark microservices validated the trace-driven simulation results. The proposed autoscaling method is capable of handling bursty workloads with minimal response time SLO violations.

Currently, we are extending our work to optimize the resource instance count to the applications while ensuring specific response time guarantees. In the future, we also plan to adjust hardware resources to containers dynamically for maximal utilization of physical infrastructure.

ACKNOWLEDGMENT

This work is partially supported by the European Research Council (ERC) under the EU Horizon 2020 programme (GA 639595), the Spanish Ministry of Economy, Industry and Competitiveness (TIN2015-65316-P and IJCI2016-27485) and the Generalitat de Catalunya (2014-SGR-1051).

REFERENCES

[1] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "Nist cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, pp. 1–28, 2011.

[2] S. Sotiriadis, N. Bessis, C. Amza, and R. Buyya, "Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling," *IEEE Transactions on Services Computing*, vol. 12, no. 2, pp. 319–334, March 2019.

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

[4] V. Hayyolalam and A. A. P. Kazem, "A systematic literature review on qos-aware service composition and selection in cloud environment," *Journal of Network and Computer Applications*, vol. 110, pp. 52–74, 2018.

[5] 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 Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, 2018.

[6] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, March 2018.

[7] M. Abdullah, W. Iqbal, and F. Bukhari, "Containers vs virtual machines for auto-scaling multi-tier applications under dynamically increasing workloads," in *Intelligent Technologies and Applications*. Springer Singapore, 2019, pp. 153–167.

[8] W. Iqbal, M. Dailey, and D. Carrera, "SLA-driven adaptive resource management for web applications on a heterogeneous compute cloud," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 243–253.

[9] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai, "Optimal autoscaling in a iaas cloud," in *Proceedings of the 9th international conference on Autonomic computing*. ACM, 2012, pp. 173–178.

[10] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 644–651.

[11] W. Dawoud, I. Takouna, and C. Meinel, "Elastic virtual machine for fine-grained cloud resource provisioning," in *Global Trends in Computing and Communication Systems*. Springer, 2012, pp. 11–25.

[12] A. Gambi, M. Pezz, and G. Toffetti, "Kriging-based self-adaptive cloud controllers," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 368–381, May 2016.

[13] X. Bu, J. Rao, and C.-Z. Xu, "A reinforcement learning approach to online web systems auto-configuration," in *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ser. ICDCS '09. IEEE Computer Society, 2009, pp. 2–11.

[14] W. Iqbal, M. N. Dailey, and D. Carrera, "Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multitier web applications," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1435–1446, 2016.

[15] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic cloud services based on machine learning techniques," *Journal of Cloud Computing*, vol. 8, no. 1, p. 5, Apr 2019.

[16] M. Arlitt and T. Jin, "1998 world cup web site access logs," August 1998. [Online]. Available: http://www.acm.org/sigcomm/ITA/

[17] A. Ali-Eldin, O. Seleznjev, S. Sjöstedt-de Luna, J. Tordsson, and E. Elmroth, "Measuring cloud workload burstiness," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 566–572.

[18] M. Lassnig, T. Fahringer, V. Garonne, A. Molfetas, and M. Branco, "Identification, modelling and prediction of non-periodic bursts in workloads," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010, pp. 485–494.

[19] 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, Jul. 2018.

[20] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014.

[21] H. Liu and S. Wee, "Web server farm in the cloud: Performance evaluation and dynamic architecture," in *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 369–380.

[22] 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 Generation Computer Systems*, vol. 67, pp. 329–340, 2017.

[23] B. Liu, R. Buyya, and A. Nadjaran Toosi, "A fuzzy-based auto-scaler for web applications in cloud computing environments," in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds. Cham: Springer International Publishing, 2018, pp. 797–811.

[24] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *2009 IEEE International Conference on e-Business Engineering*, Oct 2009, pp. 281–286.

[25] 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 *2018 Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, Feb 2018, pp. 1–7.

[26] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," *Journal of Network and Computer Applications*, vol. 82, pp. 93 – 113, 2017.

[27] V. Persico, D. Grimaldi, A. Pescape, A. Salvi, and S. Santini, "A fuzzy approach based on heterogeneous metrics for scaling out public clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2117–2130, 2017.

[28] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *Journal of Systems and Software*, vol. 151, pp. 243 – 257, 2019.

[29] I. Prachitmutita, W. Aittinonmongkol, N. Pojjanasuksakul, M. Supattatham, and P. Padungweang, "Auto-scaling microservices on iaas under sla with cost-effective framework," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, March 2018, pp. 583–588.

[30] H. Alipour and Y. Liu, "Online machine learning for cloud resource provisioning of microservice backend systems," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2433–2441.

[31] F. Zhang, X. Tang, X. Li, S. U. Khan, and Z. Li, "Quantifying cloud elasticity with container-based autoscaling," *Future Generation Computer Systems*, vol. 98, pp. 672–681, 2019.

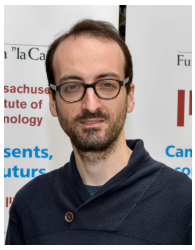
[32] D. Trihinas, Z. Georgiou, G. Pallis, and M. D. Dikaiakos, "Improving rule-based elasticity control by adapting the sensitivity

of the auto-scaling decision timeframe," in *Algorithmic Aspects of Cloud Computing*, D. Alistarh, A. Delis, and G. Pallis, Eds. Cham: Springer International Publishing, 2018, pp. 123–137.
[33] G. Verdier, "An empirical likelihood-based cusum for on-line model change detection," *Communications in Statistics - Theory and Methods*, vol. 49, no. 8, pp. 1818–1839, 2020.
[34] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium*, April 2012, pp. 204–212.
[35] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd Workshop on Scientific Cloud Computing*, ser. ScienceCloud '12. New York, NY, USA: ACM, 2012, pp. 31–40.
[36] D. Mosberger and T. Jin, "httperf: A tool for measuring Web server performance," in *In First Workshop on Internet Server Performance*. ACM, 1998, pp. 59–67.
[37] M. Abdullah, W. Iqbal, A. Erradi, and F. Bukhari, "Learning predictive autoscaling policies for cloud-hosted microservices using trace-driven modeling," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2019, pp. 119–126.
[38] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, Dec 2014.
[39] K. R. Rao, D. N. Kim, and J.-J. Hwang, *Fast Fourier Transform - Algorithms and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2010.
[40] 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.
[41] F. Soppelsa and C. Kaewkasi, *Native Docker Clustering with Swarm*. Packt Publishing, 2017.

Muhammad Abdullah is a Lecturer at Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan. He received MPhil and BS in Computer Science degrees from University of the Punjab, Lahore, Pakistan in 2016 and 2014 respectively. Currently, he is pursuing Ph.D. in computer science from University of the Punjab, Lahore, Pakistan. His research interests include cloud computing, machine learning, scalable applications, and system performance management.

Waheed Iqbal is an Assistant Professor at Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan. He worked as a Postdoc researcher with the Department of Computer Science and Engineering, Qatar University during 2017–2018. His research interests lie in cloud computing, distributed systems, machine learning, and large scale system performance evaluation. Waheed received his Ph.D. degree from the Asian Institute of Technology, Thailand.

Josep Lluis Berral received the degree in informatics in 2007, the M.Sc. degree in computer architecture in 2008, and the Ph.D. degree from BarcelonaTech-UPC, computer science in 2013. He is a Data Scientist, working in applications of data mining and machine learning on data-centric computing scenarios, at the Barcelona Supercomputing Center. He was with the High Performance Computing Group, Computer Architecture Department-UPC, and the Relational Algorithms, Complexity and Learning Group, Computer Science Department-UPC.



Jordà Polo received a M.S. from the Technical University of Catalonia (UPC) in 2009, and a Ph.D. from the same university in 2014. He currently leads a team of Ph.D. students and postdocs at the Data-Centric Computing group at Barcelona Supercomputing Center (BSC), doing research in software-defined infrastructures and data-centric architectures, proposing new models and algorithms for placement of jobs and data in future data-centers.

David Carrera received the M.S. and Ph.D. degrees from BarcelonaTech-UPC in 2002 and 2008, respectively. He is an Associate Professor with the Computer Architecture Department, UPC. He is an Associate Researcher with the Data-Centric Computing, Barcelona Supercomputing Center. His research interests are focused on the performance management of data center workloads. He has been involved in several EU and industrial research projects. In 2015, he was awarded an ERC Starting Grant for the project HiEST. He was a recipient of the IBM Faculty Award in 2010.