Optimizing the Performance of Containerized Cloud Software Systems using Adaptive PID-Controllers

MIKAEL SABUHI AND NIMA MAHMOUDI, Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada
HAMZEH KHAZAEI, Department of Electrical Engineering and Computer Science, York University, Toronto, Canada

Control theory has proven to be a practical approach for the design and implementation of controllers, which does not inherit the problems of non-control theoretic controllers due to its strong mathematical background. State of the art auto-scaling controllers suffer from one or more of the following limitations: 1) lack of a reliable performance model, 2) using a performance model with low scalability, tractability or fidelity, 3) being application or architecture-specific leading to low extendability and 4) no guarantee on their efficiency. Consequently, in this paper, we strive to mitigate these problems by leveraging an adaptive controller, which is comprising of a neural network as the performance model and a PID controller as the scaling engine. More specifically, we design, implement and analyze different flavours of these adaptive and non-adaptive controllers, compare and contrast them against each other to find the most suitable one for managing containerized cloud software systems at runtime. The controller's objective is to maintain the response time of the controlled software system in a pre-defined range, and meeting the Service Level Agreements (SLA) while leading to efficient resource provisioning.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; • **Computer systems organization** → *Availability*; • **Software and its engineering** → **Cloud computing**.

Additional Key Words and Phrases: Control Theory, Cloud Software System Adaptation, Auto-Scaling, Adaptive PID Controller, Neural Networks, Performance Analysis.

ACM Reference Format:
Mikael Sabuhi and Nima Mahmoudi and Hamzeh Khazaei. 2021. Optimizing the Performance of Containerized Cloud Software Systems using Adaptive PID-Controllers. *ACM Trans. Autonom. Adapt. Syst.* xx, xx, Article xx (x 2021), 27 pages. https://doi.org/xx.xxxx

1 INTRODUCTION

Modern distributed systems need to have robust mechanisms for dealing with changes in their performance in order to be responsive and cost-effective at the same time. This is mainly due to the stochastic nature of the underlying infrastructure's performance, variable workload, and possible failures common in the current complex computing systems. Therefore, the modern distributed systems need to have self-adaptive capabilities to sense the changes in the environment and react accordingly. Various methods have been proposed to address and implement this [2, 4, 35, 56, 57], but there has been very limited research on using control-theory-based solutions.

Authors' addresses: Mikael Sabuhi and Nima Mahmoudi Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada, {sabuhi,nmahmoud}@ualberta.ca; Hamzeh Khazaei Department of Electrical Engineering and Computer Science, York University, Toronto, Canada, hkh@yorku.ca.

1556-4665/2021/x-ARTxx $15.00
https://doi.org/xx.xxxx

Control theory has been the go-to approach for many adaptable systems, especially in physical systems due to its predictability, mathematical guarantees, and effectiveness. To use control-theory-based approaches in managing the performance of large-scale software systems effectively, we need accurate performance models of the computing software system [12]. However, building models with acceptable accuracy has proven to be lengthy and error-prone for the modern complex distributed systems [13, 34, 36, 37]. This has led to the current ad-hoc control theoretical solutions which are application-specific and not proven to be extendable for other systems [5, 54].

In this work, we plan to leverage neural networks to design an adaptive performance model that can maintain optimized performance for containerized cloud software systems. Our proposed solution leverages the guarantees and robustness associated with control theoretical approaches. We evaluate our approach in various settings for a generic three-tier containerized application that has been deployed on Google Cloud Platform (GCP). The control objective is achieved by scaling the number of containers in the application tier. Moreover, we shed some light on the control theory's applicability in designing performant software systems. More specifically, we address the following research questions:

- **RQ1:** *Can we use Neural Networks to build a reliable performance model for designing control-theoretical adaptive auto-scalers?*
  Designing performance models for large-scale distributed systems that maintain an acceptable trade-off between tractability and fidelity is a very challenging task [36, 37]. In this work, we aim to evaluate if Neural Networks can be used to design a tractable performance model with a high degree of fidelity and extendability.
- **RQ2:** *Which type of controllers can effectively maintain the performance and metrics of interest, i.e., Service Level Agreement, Mean Squared Error, and Mean Absolute Error in software systems?*
  Several run-time indicators are considered to see which controller is capable of maintaining the system's response time in the desired operating region with fewer violations. We designed and evaluated fixed PI/PID controllers and adaptive PI/PID counterparts to investigate this question. We compare these controllers to find the most suitable one for distributed software systems.
- **RQ3:** *Which controller can handle the task of resource provisioning more efficiently?*
  In this case, we strive to find out which controller can make the best use of the available resources (in our case containers). The goal is to use the resources as it is necessary without over/under-provisioning. To this end, the performance of the proposed adaptive PI and PID controllers are compared against their non-adaptive counterparts as well as a pure reactive (Scaling Heat Algorithm [12]) algorithm by considering two different workloads.

The remainder of this paper is organized as follows. Section 2 discusses the background concepts for this paper. Section 3 introduces and compares the related work in the literature. Section 4 presents our novel control-theory-based methodology. Section 5 outlines our evaluation methodology for the approaches presented in this work. Section 5.3 discusses the results obtained in our evaluation. In Section 6, we tried to identify the most important threats to our work's validity as well as some avenues for future research. Finally, Section 7 concludes the paper.

2 BACKGROUND

In this section, we briefly introduce the important concepts that are being used in this paper, namely, container virtualization and control theory.

Docker containers provide us with a lightweight and low overhead solution to implement containerized application and the well known microservice architectures [47]. Their fast startup

enables us to perform rapid-scaling actions, both horizontal and vertical, compared to other solutions like virtual machines.

Cluster management platforms like Kubernetes [28], Mesos [48], and Swarm [20] allow the software system to be deployed and reconfigured at large scale. These orchestrators provide the deployment team with the ability to configure the software without worrying about the underlying infrastructure. These platforms also contain simple reactive auto-scaler units for the deployed cloud application out of the box.

As shown in Figure 1, in the context of control theory, each control system consists of several blocks, e.g., sensor, actuator, feedback controller, and system under control. We briefly explain each of these elements.

- *Control Objective*: refers to the control system's purpose. For example, it could be controlling the average response time of the controlled system to be less than t seconds or lay in a pre-defined range.
- *Set-Point*: represents the desired value for the output of the controlled system. For instance, the set point for average response time could be 1000 ms. Note that the set point could be a range as well.
- *Sensor*: refers to a component that enables us read the system output, or in our case, the performance variables. For example, average response, fail ratio, queue length, etc. In fact, it is a software component used for monitoring the controlled system, e.g., Locust and Jmeter.
- *Error*: denotes the difference between the set-point and the measured value of the output read by the sensor, representing the deviation from the desired value for the control objective.
- *Control Signal*: represents the value computed by adopting a specific controller with regards to the error, e.g., having a 400 ms error, the controller might signal for creating two more containers.
- *Controller*: describes a mechanism or algorithm that calculates the control signal to achieve the required control objective, considering the error value. We refer to this algorithm as the control law.
- *Actuator*: refers to a mechanism that can be used to affect the controlled system. For instance, the number of Virtual Machines or containers.
- *System Under Control*: denotes a system to be controlled by adjusting the actuator(s). For example, a containerized cloud application as a system that its control objective is met by creating/removing replicated containers.

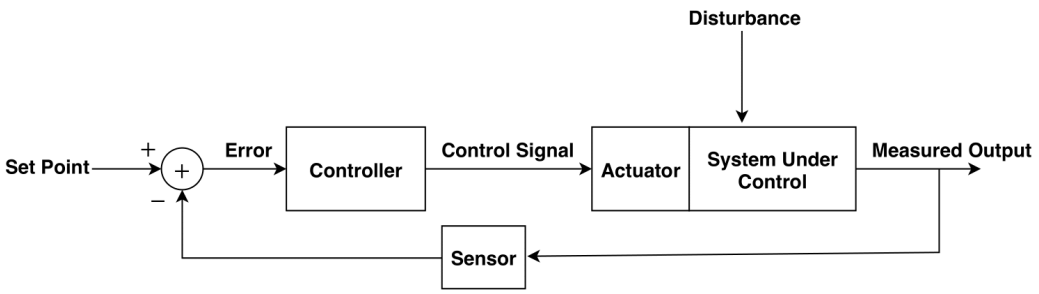


Fig. 1. Block diagram of a simple control-theoretical feedback loop.

The Proportional-Integral-Derivative (PID) controller, also known as "three term" controller [3], still is the most popular controller in the industry due to its simplicity and transparency. The control

signal of the continuous-time PID controller is described as follows; please note that in this study we used discrete-time form of the PID controller (discussed in Section 4.2) and this continuous-time form is discussed here only due to its simplicity:

$$u(t) = K_P e(t) + K_I \int_0^t e(t) + K_D \frac{de(t)}{dt} \tag{1}$$

Where $u(t)$ is the control signal calculated by the controller, $e(t)$ is the current error, and K_P denotes the proportional gain, K_I the integral gain, and K_D the derivative gain. The purpose of each of these gains are described as follows [3]:

- *Proportional Gain* K_P: shows the sensitivity of the control system to the current error.
- *Integral Gain* K_I: shows the sensitivity of the control system to past errors. Integral term acts as a low-frequency compensator to reduce the steady-state error.
- *Derivative Gain* K_D: shows the sensitivity of the control system to the future trend of the error. Derivative term acts as a high-frequency compensator to improve the transient behaviour of the system.

In order to tune the aforementioned PID controller parameters, interested readers are encouraged to refer to [3] for a thorough overview of *PID* controllers and their tuning methods.

3 RELATED WORK

In this section, we discuss the prior work related to our study of optimizing the performance of containerized software systems using adaptive PID-controllers. In particular, we discuss the related work on performance modelling of cloud software systems, non-adaptive and adaptive control-theoretical methods for auto-scaling.

3.1 Cloud Software System Performance modelling

Performance unpredictability and responsive auto-scaling of the cloud applications are listed in the top 10 obstacles for adopting the cloud [8]. Xiong et al. [60] present a novel approach for studying computer service performance in cloud computing. To fulfill the Quality of Service (QoS) guaranteed services in such a computing environment, they seek to find the relationship among the maximum number of users, the minimal service resources, and higher service levels. The authors introduce a queuing network model and then use an approximation method to compute the Laplace transform of a response time distribution. Moreover, they model the Web server and service center as an infinite queue for single-class customers.

Qian et al. [52] propose a hierarchical modelling approach to analytically evaluate Quality of Experience (QoE) of Online Service providers (OSPs) who are using cloud environments. They use four sub-models, namely an outbound bandwidth model, a cloud computing availability model, a latency model, and a cloud computing response time model. These sub-models are combined into one whole model using a redirection strategy graph. Their proposed approach is suitable for endless interactions in this environment. Moreover, one can easily change these identified sub-models and also add other sub-models to the existing model.

Khazaei et al. [38] put forth an analytical model for performance evaluation of cloud server farms. The authors model a cloud server farm as a $M/G/m$ queuing system, considering it a Markov process. Then, they employ embedded Markov chain techniques to analyze the performance of the cloud server farms and verify its accuracy. In their follow-up study [39], they describe a new approximate analytical model based on the Markov chain model for performance evaluation of cloud server farms. The model determines the relationship between the number of servers and input buffer size with performance metrics such as the mean number of tasks in the system, blocking probability, and the probability that a task will obtain immediate service as well as response

time distribution characteristics. In another study [40], the authors introduce a performance model suitable for analyzing large-sized Iaas clouds' service quality, using interacting stochastic models. They leverage both analytical and simulation modelling to address the complexity of cloud computing systems. Their proposed model fuses cloud centers' important characteristics such as batch arrival of user requests, resource visualization, and realistic servicing steps to obtain the important performance indicators such as task blocking probability and total waiting time incurred on user requests. Also, cloud centers' performance with a high degree of virtualization and Poisson batch task arrivals is evaluated in [41]. The proposed model is based on a two-stage approximation technique where first, they model the non-Markovian process with an embedded semi-Markov process, which is then modelled by an embedded Markov process but only at the time instants of super-task arrivals.
Malik et al. [46] provide an in-depth analysis, modelling, and verification of some of the open-source state-of-art VM-based cloud management platforms. They leverage high-level Petri nets (HLPN) to model and assess the software systems' structural and behavioural characteristics with the advantage of providing firm mathematical representations. The authors verified the models using SMT-Lib and Z3 solvers.
Chang et al.[15] highlight the fact that the heterogeneity of the workload in real Iaas Cloud Data Centers (CDCs) makes the performance modelling of complicated Iaas CDCs a challenging task. Their study studies a situation in which the number of virtual CPUs requested by each customer job is different. They present a hierarchical stochastic modelling approach for performance analysis of CDCs to quantify the impact of variation in job arrival rate, buffer size, and the maximum vCPUs numbers on the cloud service quality.
Shekhar et al. [53] present an online data-driven approach that leverages Gaussian Process-Based machine learning techniques to build run-time predictive models of the performance of the system under different interference intensity. This model can adapt itself to the changes in the workload. The reason for selecting Gaussian Process to model the latency variations due to varying workload is that these types of models require a small number of hyperparameters, and also they can model the non-linear behaviour of the target system. However, these models are probabilistic, and Then they use this model to make run-time decisions for vertical scaling of the resources.

3.2 Non-adaptive Control Theoretical Auto-Scaling

Gergin et al. [25] propose a decentralized autonomic architecture based on a fixed PID controller for a n-tier application. They implemented the proposed method on a custom data mining web application based on the FIFA 1998 Workload. For this three-tier application, three separate PID controllers control the number of Virtual Machines (VMs) for each tier to maintain the corresponding CPU utilization at a constant value. Another interesting study Barna et al. [11], evaluates the performance of a non-adaptive PID controller in maintaining the desired behaviour for a web application on SAVI as the private and Amazon EC3 as the public cloud provider. The controller has been tuned manually by trial and error, and the control objective of this controller is to keep the CPU utilization within a specific range.

The problem of control granularity and decoupled control is discussed in [44]. They point out that most of the available cloud controllers function without direct knowledge about the cloud software system behaviour and performance metrics.

Control theory based adaptation using a Fixed PI controller is evaluated and compared with the threshold-based and model-based method in [24]. Introducing smooth and sharp variations in the cloud application workload, these three controllers' performance in maintaining the CPU utilization around 70% is evaluated on a minimal three-tier web application. Looking into results

from an efficiency and effectiveness perspective, the PI controller exhibits better resource utilization and faster settling time and rise time.

3.3 Adaptive Control Theoretical Auto-Scaling

Several studies on adaptive controllers have been carried out; for example, a vertical elastic controller for memory is presented in [22] along with a method for error smoothing. The control actuator is the memory size allocation, and the target is to maintain the desired response time.

To handle the capacity shortage in cloud infrastructure, the brownout concept is extended from electrical grids to cloud software systems in [43]. During high load, brownout downgrades the user experience, e.g., by decreasing the optional content to be served (i.e., dimmer value); in doing so, the Service Level Objective (SLO) can be maintained. Brownout-compliant applications help support more users and consume less resources while satisfying the SLO. An adaptive PI controller is synthesized for coping with changes in the number of users and the environment. There are several studies on brownout-compliant cloud software systems. For example, Maggio et al. [45] study the applicability of adaptive PI, adaptive deadbeat, adaptive PID, and Feedforward-feedback controllers on Brownout compliant systems. The performance of these controllers against changes in application requirements and resource availability is evaluated. The results of this work indicate that the feedforward-feedback controller has better performance while requiring significant engineering effort. Therefore, since the adaptive PID controller is simpler to implement, it's a preferable choice.

Event-based application brown-out as an improved approach to brown-out is presented in [19] based on the queue-length of pending requests. In this study, improvement in control objectives is reported combining PI controller with machine learning algorithms. Moreover, Nydlander et al. [49] improve this event-based control by proposing a more accurate model of brown-out applications using queuing theory. Quantitative comparison for the proposed cascade controller with original Brown-out and event-based brown-out shows that better performance is achieved by having two control levels: one for the inner-loop and one for the outer-loop. The flexibility of control theory in dealing with software systems is undeniable; one can find interesting adaptive controllers designed using brown-out for load balancing strategies, e.g. [21, 51].

Baresi et al. [10] enumerate the benefits of containerizing the applications and investigates the performance of auto-scaling in both container and Virtual Machine level. They extend the EcoWare [9] framework to achieve compatibility with containerized applications and then develop an autonomic control theoretical approach for auto-scaling of a cloud application. A new dynamic model for the controlled system is presented to model the application's response time as a function of assigned cores and request rate.

To address the incompatibility of the non-functional software models derived from the architectural description of the software with control theoretical approaches, Arcelli et al. [6] use the Modelica library to represent a Queuing Network. This library provides some adjustable parameters for controlling the behaviour of the cloud software system. Moreover, Model Identification Adaptive Controller(MIAC) based on the layered queuing model and optimal control is presented in [14]. This non-linear model is linearized around the operating point to tune the optimal controller parameter. Moreover, Incerto et al. [32] propose a control algorithm for horizontal and vertical auto-scaling of a cloud software system based on Model Predictive Control(MPC) strategy is presented. In this study, they consider a compact approximate representation of queuing networks based on ordinary differential equations(ODEs) to meet the performance requirements by the model predictive controller. However, the authors point out that the only technical limitation of their proposed method is the single class assumption in the QN model, which they address this limitation in their follow up paper [33].

Several comprehensive surveys on the application of control theoretical methods are found in the recent literature. Ullah et al. [58] look into available methods from both control solution view and elasticity view and review available work on control theoretical methods for cloud application elasticity and outlines the existing challenges and trends in adopting such methods. Filieri et al. [23] review and elaborate on the current control strategies for self-adaptive software systems starting from goal identification to the verification and validation of the controlled system and at the end, highlight the open challenges, both from the software engineering and the control theory perspective. Moreover, a systematic literature review on control theoretical software adaptation has been presented in [55].

Considering the previous studies on this area, we found that the most challenging problem in adopting control theory in cloud software system adaptation is modelling the software systems. According to previous studies in this area, researchers propose various modelling methods for software systems, such as using queuing networks, to make their auto-scaler adaptive to the environment's changes. However, modelling such highly stochastic and nonlinear systems,e.g., cloud applications, is a painstaking and costly task and requires a significant engineering effort. Therefore, in this paper, we propose an approach for data-driven modelling of the software systems leveraging neural networks due to the performance data's abundance. To evaluate our proposed method in **RQ1**, we investigate the possibility of using neural networks for reliable performance modelling of cloud applications. Moreover, in **RQ2** and **RQ3**, we evaluate the performance of the proposed performance modelling method along with an adaptive controller in maintaining the control objectives and from an efficiency point of view.

4 METHODOLOGY

In this section, we go over the details of the methodology proposed in this paper.

4.1 Performance Modelling

The inherent nature of the cloud software systems exhibits some non-linear behaviours at run-time. Modelling such systems is a non-trivial task and sometimes impossible due to the complexity of the non-linear systems. Nevertheless, due to the constant changes in the cloud environment, this model will not accurately represent the system after a while. Linear approximation of these non-linear models only works around the operating point and will not be valid if any significant changes occur in the system's operating point. Considering all these problems, we attempt to develop a new approach for modelling cloud software systems and further adjusting it to its environment changes. Due to the abundance of the data, we propose a data driven approach to obtain a non-linear model of the cloud software system.

The neural networks can be a convenient solution to tackle the problems mentioned above. According to the universal approximation theorem [18, 31], we can use a feed-forward network with a linear output layer and at least one hidden layer with some specific activation functions,e.g. logistic sigmoid functions, to approximate any function provided that the enough hidden layers and neurons are given [26]. Several modelling approaches are developed and validated in [50]. Most of these modelling approaches are performed on the electrical and physical systems. For instance, [1] uses Recurrent High-Order Neural Networks (RHONN) for real-time discrete non-linear modelling of an induction motor. A neural network-based non-linear auto-regressive moving average with exogenous inputs for modelling a piezoelectric actuator has been presented in [17]. These studies motivated us to carry out some experiments to examine neural networks' applicability for modelling cloud software systems. Here, we formalize Hypothesis 1 to obtain the performance model for cloud applications.

Hypothesis 1. In any situation, the performance of a containerized cloud application is a non-linear

function of the number of requested containers, the number of successfully provisioned containers, and the number of users with some time-delays.

$$\begin{aligned}\hat{y}(n+1) =& f(u(n), ..., u(n-n_u),\\ & z(n), ..., z(n-n_z), d(n), ..., d(n-n_d))\end{aligned} \tag{2}$$

where $\hat{y}$ is the response time, f is the non-linear function, u is the number of requested containers, z is the number of running containers, d is the number of users and n is the discrete time index.

4.1.1 Neural Networks. Neural Networks are employed extensively for system identification due to their satisfactory performance in non-linear dynamic system modelling [16]. In this work, we use the Multi-Layer Perceptron (MLP) as a feed-forward neural network for system identifications. Figure 2 shows the proposed architecture of the MLP neural network. For simplicity, only one hidden layer is considered.

The first layer is the input layer, the second layer is the hidden layer, and the last one is the output layer. We use hyperbolic tangent as the activation function for the hidden layer, and the output layer is a linear function.

According to Figure 2, the input vector for the Neural Network is:

$$\begin{aligned}x_i(n) =& (u(n), ..., u(n-n_u),\\ & z(n), ..., z(n-n_z), d(n), ..., d(n-n_d))\end{aligned} \tag{3}$$

Where in our case, u, z, d, and n are the number of requested containers, number of running containers, number of users, and discrete-time index, respectively. Also, n_u, n_z, and n_d denote the corresponding time delay for each of these inputs.

The cost function for the system identification purpose is defined as:

$$J = (y(n+1) - \hat{y}(n+1))^2 = \frac{1}{2}(e_m(n+1))^2 \tag{4}$$

where $y(n+1)$ represents the measured output, $\hat{y}(n+1)$ is the predicted output, and $e(n+1)$ is the prediction error.

The system identification procedure is required to obtain the sensitivity of the system's output to the change in control input, which is the number of requested containers in our case. The Jacobian of the system can be derived as follows:

$$\frac{\partial y(n+1)}{\partial u(n)} \approx \frac{\partial \hat{y}(n+1)}{\partial u(n)} \tag{5}$$

Data-driven system identification comprises two stages: 1) data collection and 2) model training and verification. In the following subsection, we discuss the best practices to collect the appropriate data for better system identification.

4.1.2 Data Collection. Data collection is an indispensable part of data-driven system identification, and the resultant performance of the modelling highly depends on the quality of the acquired data. System identification can be carried out in an offline/online manner. Most of the time, it is preferable to train the neural networks model online. However, this requires a high sampling rate. Since the monitoring systems for cloud software systems suffer from a low sampling rate, we are bound to first use some offline data for pre-training, and then do further online training at run-time to adapt the model to the changes in application or underlying infrastructure. Hence, we have a model that can be used in a highly dynamic environment. To this end, we have collected offline data from a cloud software system as our system under study to be identified for about 48 hours and with a sample rate of 20 seconds. To acquire meaningful data from the system, it is crucial to

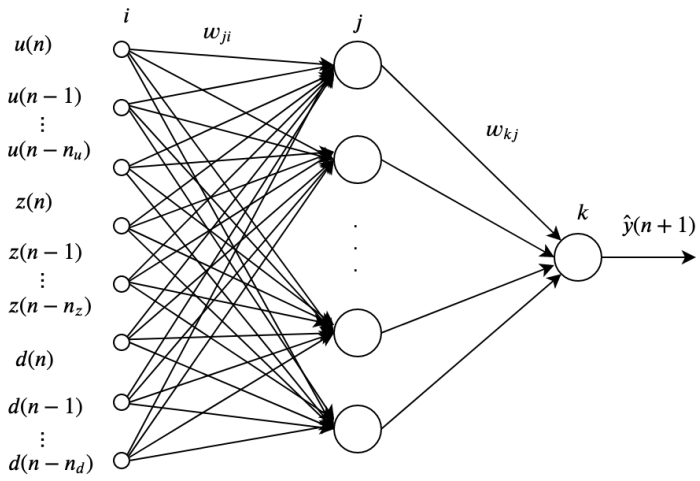


Fig. 2. The structure of the proposed neural network for modelling the behaviour of the cloud software system.

monitor the system's behaviour in most system states. Furthermore, we can apply the data logged in the cloud monitoring system's repository for identification purposes. One can find an example of data logging module in our online repository[1].

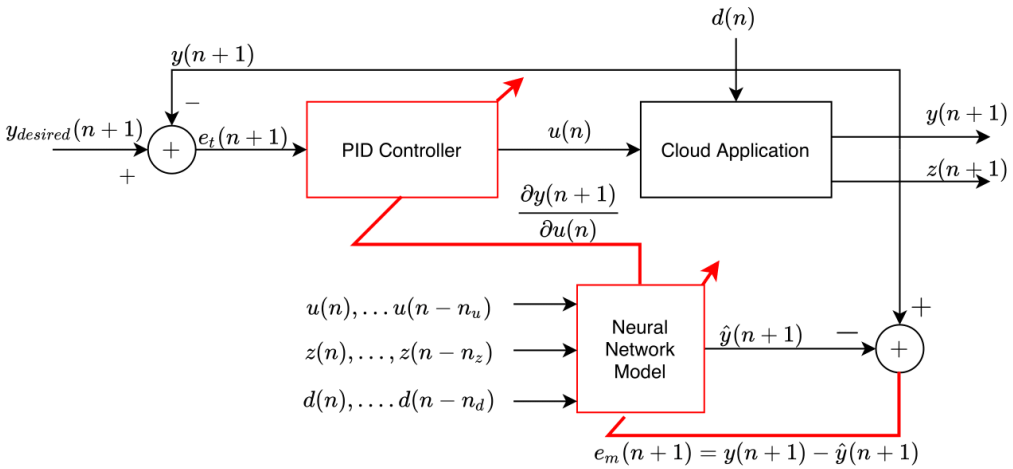
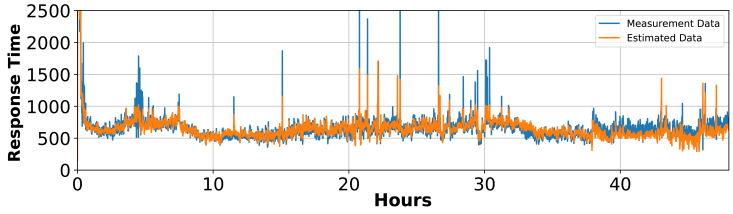


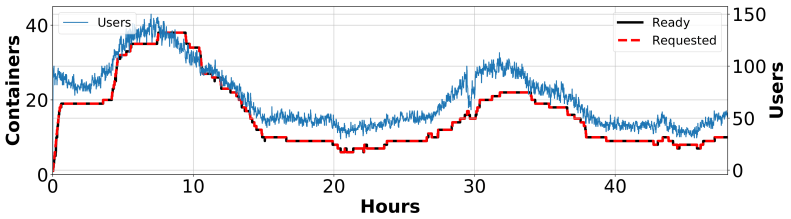
Fig. 3. The overall structure of the identification and control of a cloud application, red arrows on the Neural Network model and PID controller blocks indicate the online adaptation of these blocks.

4.1.3 Model Training. In this section, we use data collected in the data collection phase (refer to Section 4.1.2) to obtain the performance model. A multi-layer perceptron with one hidden layer

[1]https://github.com/pacslab/NNPIDAutoscaler



(a) Measurement vs estimated value from the trained model.



(b) The dynamics of workload and the number of running containers during the off-line training.

Fig. 4. Training the neural network before activating the adaptive controller. During this time, the system uses a pure reactive auto-scaler to bring the adaptive controller up to speed.

containing 40 neurons is adopted for this purpose. According to Equation (3), the input vector consists of two-time delays for u, z, and d, i.e., $n_u = n_z = n_d = 2$. Therefore, the input vector is a nine-by-one vector. The number of time delays is obtained from observations in data. The number of neurons, hidden layers and time delays depends on the non-linearity of the target software system. The more non-linear the system, the more neurons and deeper architecture are needed. Therefore, we selected the number of neurons and hidden layers after several manual training and finding the best fit. These numbers depend on the non-linearity of the target software system. We tried to use the simplest model since increasing the number of neurons, and hidden layers may result in the problem of overfitting and computational overhead. Our goal is to model the non-linear behaviour of the cloud software system with these nine inputs for the next time step.

In order to train the model, 67% of the dataset is used for training and 33% for the validation. In the pre-processing step, we scaled the input data to the range 0 to 1 according to the Min-Max scaling algorithm, to prevent the neural network model from saturating. We used Adam Optimizer to train the weights and biases of the neural network with a training rate of $\eta = 0.002$, $\beta_1 = 0.99$, $\beta_2 = 0.9$, and $\epsilon = 1e - 8$ which are the default training values for the optimizer [42]. We set the batch size to 32 and the training epoch to 10000. We executed the model training five times.

Now, we can respond to **RQ1**. Figure 5 shows the distribution of the evaluation metrics for performance modelling in the training and validation stages. In our offline training phase, our neural network showed a Mean Squared Error (MSE) of 5931.20 and 8248.60, a Mean Absolute Error (MAE) of 42.34 and 71.57, and a Mean Absolute Percentage Error (MAPE) of 5.96% and 11.278% for the training set and the validation set, respectively. This is while the average response time in our training set is 667.79 ms, with a variance of 187,480.34.

The root cause of the reported modelling errors can be due to a lack of sufficient and representative data or an appropriate regularization in the model.

Considering the variations in the data and the obtained evaluation metrics from the model, our neural network provides acceptable predictions during the training and evaluation phases for

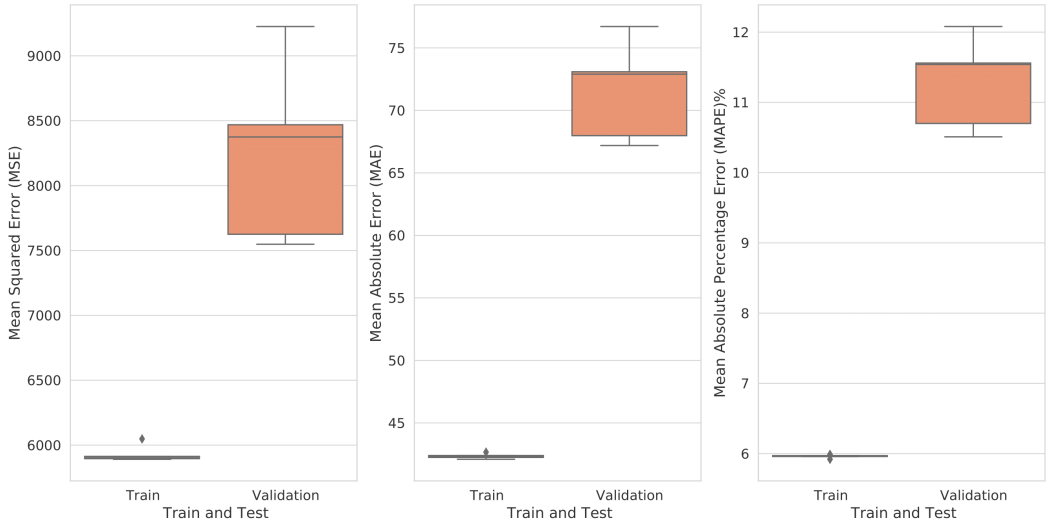


Fig. 5. Distribution of the evaluation metrics for performance modelling in the training and validation stages.

an extremely stochastic system. The training set and validation set results show that the neural network is not over-fitting to the training set during our training.

4.2 Controller Design

In this paper, we opted to use the adaptive PID controller's incremental version, which is available at our online repository [2]. In this scheme, the error signal for the controller is defined as follows and is shown in Figure 3:

$$e_t(n+1) = y_{des}(n+1) - y(n+1) \tag{6}$$

where $y_{des}(n+1)$ is the desired set point for the response time of the cloud software system, and $y(n+1)$ is the measured response time of the system.
Control law for the discrete incremental PID [29] is defined as follows, one should note that since number of containers cannot take float values we use ceiling function on the $u(n)$:

$$\begin{aligned} u(n) = \lceil u(n-1) + k_p(e_t(n) - e(n-1)) + \\ k_I e(n) + k_D(e(n) - 2e(n-1) + e(n-2)) \rceil \end{aligned} \tag{7}$$

where $u(n)$ is a control input at discrete-time n. k_p, k_I, and k_D are the proportional, integral, and derivative gains of the incremental discrete controller respectively. We are going to adaptively change these gains as the system behaviour changes with respect to the following cost function for controller [29]:

$$J = \frac{1}{2}(y_{des}(n+1) - y(n+1))^2 = \frac{1}{2}(e_t(n+1))^2 \tag{8}$$

[2]https://github.com/pacslab/NNPIDAutoscaler

To adjust the controller gains, the gradient descent method has been applied as follows:

$$\begin{aligned} k_P(n+1) &= k_P(n) - \eta_c \frac{\partial J}{\partial k_P} \\ k_I(n+1) &= k_I(n) - \eta_c \frac{\partial J}{\partial k_I} \\ k_D(n+1) &= k_D(n) - \eta_c \frac{\partial J}{\partial k_D} \end{aligned} \tag{9}$$

where η_c is the controller learning rate. From equations 8 and 9, using the chain rule for derivatives, we can derive the following equations for updating the PID controller gains:

$$\begin{aligned} \frac{\partial J}{\partial k_P} &= \frac{\partial J}{\partial y(n+1)} \frac{\partial y(n+1)}{\partial u(n)} \frac{\partial u(n)}{\partial k_P} = -e_t(n+1) \frac{\partial y(n+1)}{\partial u(n)} \theta_1(n) \\ \frac{\partial J}{\partial k_I} &= \frac{\partial J}{\partial y(n+1)} \frac{\partial y(n+1)}{\partial u(n)} \frac{\partial u(n)}{\partial k_I} = -e_t(n+1) \frac{\partial y(n+1)}{\partial u(n)} \theta_2(n) \\ \frac{\partial J}{\partial k_D} &= \frac{\partial J}{\partial y(n+1)} \frac{\partial y(n+1)}{\partial u(n)} \frac{\partial u(n)}{\partial k_D} = -e_t(n+1) \frac{\partial y(n+1)}{\partial u(n)} \theta_3(n) \end{aligned} \tag{10}$$

where $\frac{\partial y(n+1)}{\partial u(n)}$ is Jacobian of the controlled system and it is obtained from the system identification process, and $\theta_1(n) = e(n) - e(n-1)$, $\theta_2(n) = e(n)$ and $\theta_3(n) = e(n) - 2e(n-1) + e(n-2)$. The algorithm for proposed adaptive controller is shown in Algorithm 1.

It is worth noting that the controller adaptation will be performed when the neural network exhibits "satisfactory accuracy". In our experimental evaluations, we define the neural network is satisfactory accurate when the prediction error in response time is less than 300ms. Considering this value helps us prevent controller adaptation from diverging since if any large changes happen in the identified model, it will lead to a wrong Jacobian. According to Algorithm 1, the NN model is fine tuned at runtime to adapt itself to the changes in the environment, therefore, we can wait until the model is learned online again and then adjust the controller to the new model. Moreover, according to Algorithm 1, the controller optimization is only performed when the neural network model's prediction error is lower than the threshold and when the regulation error is not zero, i.e. the response time is not between 500-800 ms. Although a full stability analysis would stray from the scope of this study, the proposed approach for adaptation of the controller is a viable and robust means to prevent the same adaptation from driving the closed-loop system toward instability.

5 EVALUATION

In this section, we describe the experimental setup and the configuration of the underlying infrastructure. Then, we discuss the experimental methodology for more detail on workloads and performance metrics.

5.1 Experimental Setup

To properly evaluate the effect of the proposed algorithm on performance compared to with other auto-scaling algorithms, we deployed a three-tier containerized WordPress cloud application with MySQL as the database and Nginx as the webserver deployed on a Kubernetes cluster. We picked WordPress as the benchmarking application due to its widespread use in more than 34% of all websites over the Internet [59]. For load testing, we leveraged an extended version of Locust Library [30], available publicly in our repository[3] on the same cluster to minimize the effect of

[3]https://github.com/pacslab/pacs_locust

Algorithm 1: Proposed Adaptive Controller Algorithm

input : y_{n+1} — The average response time measured for the current time step.
input : $u_n, u_{n-1}, ..., u_{n-n_u}$ — The number of requested containers for n_u previous time steps.
input : $z_n, z_{n-1}, ..., z_{n-n_z}$ — The number of running containers for n_z previous time steps.
input : $d_n, d_{n-1}, ..., d_{n-n_d}$ — The number of users for n_d previous time steps.
input : $K_P(n), K_I(n), K_D(n)$ — The controller parameters at time step n.
output: $u(n+1)$ — The updated controller command for the next time step.

while *True* **do**
  /* According to the current parameters($K_P(n), K_I(n), K_D(n)$) */
  Calculate $u(n)$ using Equation 7
  Calculate the Jacobian using Equation 5
  Prepare the input vector $x_i(n)$ for NN model according to Equation 3
  /* NN model predicts the average response time. */
  $\hat{y}(n+1) \leftarrow f(x_i(n))$ according to Equation 2
  $e(n+1) \leftarrow \hat{y}(n+1) - y(n+1)$
  /* Checking the accuracy of the predictor. */
  if $|e(n+1)| < Threshold$ **then**
    /* Updating the control variables */
    $k_P(n+1) \leftarrow k_P(n) - \eta_c \frac{\partial J}{\partial k_P}$
    $k_I(n+1) \leftarrow k_I(n) - \eta_c \frac{\partial J}{\partial k_I}$
    $k_D(n+1) \leftarrow k_D(n) - \eta_c \frac{\partial J}{\partial k_D}$
  else
    /* The control variables are not updated due to unsatisfactory prediction. */
    $k_P(n+1) \leftarrow k_P(n)$
    $k_I(n+1) \leftarrow k_I(n)$
    $k_D(n+1) \leftarrow k_D(n)$
  end
  /* According to the new control parameters($K_P(n+1), K_I(n+1), K_D(n+1)$) */
  Calculate $u(n+1)$ using Equation 7
  /* To improve the prediction accuracy the NN-Model is fine-trained. */
  Fine-tune the NN-Model by optimizing Equation 4
end

network latency in the results. Our extended version of Locust gave us more flexibility during experiments than any other available load testing tool. We used a simple REST API developed on the load testing tool[4] that helped us control the load testing and measure the performance of the application at runtime. The details of our deployment, along with the configuration settings, can be found in our GitHub repository[5]. To synchronize the WordPress's upload folder between different instances, we used an NFS server deployed to the same zone as the Kubernetes cluster to minimize the network latency.

[4]https://github.com/pacslab/pacs_load_tester
[5]https://github.com/pacslab/wordpress-kubernetes-deployment

Table 1. Configuration of the Kubernetes Cluster.

Property	Value
Cluster Zone	us-central1-a
Minimum Nodes	1
Maximum Nodes	7
VM Type	n1-standard-1
vCPU	1
RAM	3.75 GB
SSD	30GB
OS	Container-Optimized OS
Client Version	1.13.9
Server Version	1.13.6

5.1.1 Kubernetes Cluster. For the purpose of this work, we deployed a Kubernetes cluster on the Google Cloud Platform (GCP) using the Google Kubernetes Engine (GKE). The cluster configuration used in the experiments can be found in Table 1.

5.1.2 Benchmarking Application. In this section, we introduce the details of our benchmarking three-tier containerized application. For this study, we used WordPress with PHP FPM version 7.3 as our application server. Our application server uses MySQL version 5.6 as the database and Nginx version 1.7.9 as its web server. Our configuration files, as well as the Docker images and deployment procedure, is publicly available on the WordPress deployment GitHub repository mentioned above.

5.2 Experimental Methodology

In this section, we discuss different workloads and the motivations behind selecting them. Moreover, the performance metrics of interest are presented to compare the controllers' ability to satisfy the cloud software system SLA requirements. It is noteworthy that before designing the experiments, we first evaluated the available resources' capacity in handling the number of users. Regarding the VMs, we could create 42 containers of the WordPress Application. Therefore, the number of containers is more than sufficient to satisfy the control objectives,e.g. SLA. We assumed that the orchestration system works correctly in resource provisioning, i.e. the infrastructure is reliable. In other words, all the containers are created successfully when the appropriate command is called.

5.2.1 Step-change Workload. Figure 6 portrays the step-change workload (blue line) in which the number of users changes abruptly, resembling a step signal. The number of users rises from 30 to 60 and then declines back to 30, in a period of 36 minutes. The rationale behind considering such workload is to evaluate the robustness of the controllers against sudden disturbances and also study the adaptive behaviour of the proposed controller in managing the SLA, performance in particular. This is a common situation for online websites that offer short time deals, attracting a considerable number of users in a short time.

5.2.2 FIFA World-Cup Workload. Figure 6 presents the second workload (black line). In this case, the number of users varies according to the variations of the FIFA World-Cup 1998 data set [7] collected from '1998-06-30 08:00:01' to '1998-07-01 08:00:00'. Given the systems' capacity in handling the number of requests, we scaled down the number of users between 30 and 150 while persevering the shape of the trace. The reason for choosing 150 as the number of user upper limit is that the application is close to its saturation point but can still maintain the control target in the desired region. Also, the time scale is transformed from 24 hours to 8 hours, which helped us study the

control system's performance under more intensified fluctuations in the number of users. This behaviour may occur in web services offering live sports game streaming. At the start of each sport event, the number of online streamers increases gradually, and at the end of the event, a gradual decrease can be observed.

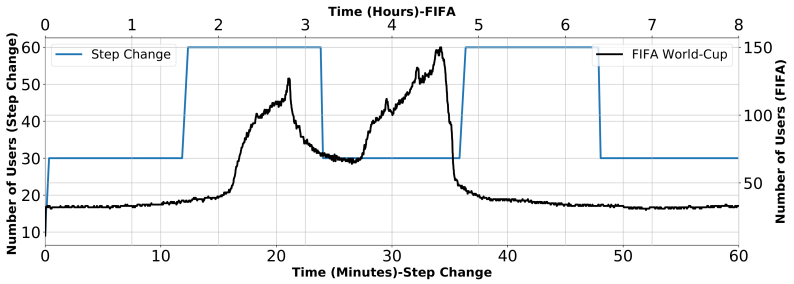


Fig. 6. FIFA World-Cup and Step Change Workload.

5.2.3 Performance Metrics. Various performance metrics are considered to have a clear comparison of controllers' capability and efficiency in maintaining the optimized performance. We consider well-known performance metrics from the control and software community to have the best of two worlds. The Mean Squared Error (MSE), Mean Absolute Error (MAE), and rise time are used as representative performance indices from control theory. The definition of MSE and MAE are given in Equation (11) and (12), respectively. For the sake of simplicity, we define the rise time as the time required for the controller to bring the response time of the system to the desired operating region after the disturbance has been applied.

$$MSE = \frac{1}{N}\sum[e(t)]^2 \tag{11}$$

$$MAE = \frac{1}{N}\sum|e(t)| \tag{12}$$

For software domain performance metrics, Service-Level Agreement (SLA) and efficient provisioning are considered. We define the SLA as follows: the upper limit for the average response time $(1,000ms)$ cannot be violated more than one minute, i.e., three consecutive monitoring intervals. If such a violation occurs, we aggregate the violation time and count this as a violation with a penalty. Since the SLA does not change frequently, i.e. the set point for the response time is not changed; we are dealing with a regulation and disturbance rejection control problem. Therefore, the goal is to regulate the set-point around the defined SLA and reject the effect of change in the number of users on the response time,i.e. disturbance rejection.

Three different provisioning efficiency metrics are considered according to the Cumulative Distribution Function (CDF). The over-provisioning is defined as the percentage of the time that response time is below 500ms, i.e., containers are more than necessary. We prefer this value to be as low as possible. The under-provisioning is defined as the percentage of the time that response time lies between $800 - 1,000ms$, i.e., more containers are required—similarly, the less the under-provisioning, the better. Efficient provisioning is defined as the percentage of the time that response time is between 500-800ms, i.e., the number of containers is according to the system requirements. It is ideal to have higher values for efficient provisioning.

5.3 Experimental Results

To evaluate the efficiency of the proposed method in maintaining the response time of the three-tier containerized web application, several experiments have been carried out with different controllers; in particular, the responsiveness and robustness of the controllers under two types of workloads are investigated.

The scaling heat algorithm is employed from [12] as the baseline, which represents a robust reactive algorithm to maintain the performance metrics. According to this algorithm, when a violation of the upper threshold occurs, indicating the saturation of the container, we increase the heat factor by one, and if a violation from the lower threshold happens, indicating resource under-utilization, we decrease the heat factor by one. If no violations occur, i.e. the utilization is within the range, we decrease the heat factor by one until the heat factor is zero. If the heat factor is equal to a specific number (e.g., $n/-n$), we create/terminate one container. The primary distinction between the scaling heat algorithm and other threshold-based algorithms is that it waits for several consecutive violations to react. This gives the scaling algorithm robustness against the ping-pong effect. The ping-pong effect is an undesirable alternating scale up and scale down of the resources (containers), which results in oscillation in the number of containers and the performance metrics. In our experiments, we chose the number of consecutive violations needed for the scaling to occur to be 5 ($n = 5$), according to the [12], with an upper trigger point of 800 and a lower trigger point of 500.

As for the non-adaptive control theoretical approach, we evaluate the fixed-PI (FPI) and fixed PID (FPID) controllers. Note that in the absence of an appropriate linear model for the target software system, these controllers' gains are obtained after a short period of experimentation and manual tuning. The gains were set to $K_P = 0.0004$, $K_I = 0.0004$, and $K_D = 0.00005$.

The obtained gains from the manual training are used as an initial condition for our proposed adaptive controllers. We consider adaptive neural network based PI (NN-PI) and adaptive neural network based PID (NN-PID) controllers for evaluation. The controller learning rate η_c is set to $2e-13$ for K_P and K_I, and $2e-14$ for K_D, which is tuned manually. The reason for setting small values for these control parameters and the learning rate is because the time is in millisecond and choosing larger values may result in divergence in adaptation. Adam Optimizer is selected for training the weights and biases of the neural network with the training rate of $\eta = 0.002$, $\beta_1 = 0.99$, $\beta_2 = 0.9$, and $\epsilon = 1e-8$. There is no specific way to find the optimal initial value for controller learning rates, and we need to find it by trial and error.

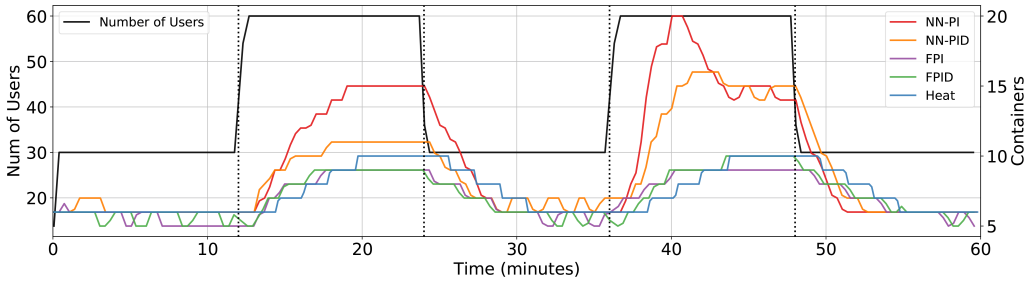
The control objective is to maintain the response time of the system between 500-800ms (set-point) by adjusting the number of containers (actuator) to immediately react to the change in the number of users (disturbance). In this range, the control error is considered zero, and hereafter we refer to this range as the desired operating region. The motivation behind choosing this range is that most of the cloud providers prefer to maintain the response time of their system in a pre-defined range, to prevent SLA violations and over-provisioning at the same time. Therefore, we can evaluate the controllers' applicability to the real world control problems for cloud service providers.

For each workload, we conduct the experiment three times for reproducibility. All the reports are given on average of these three experiments. Care has been taken to carry out the experiment in similar conditions. In other words, we conduct the experiments each day simultaneously and on the same infrastructure. In our experiments, we used Google N1 standard machine types. These types of virtual machines do not use shared CPUs, i.e. each of them has its own dedicated resources. Therefore there is minimal interference between Virtual Machines, which is negligible. According to GCP benchmarks, these instances only have 2.47% standard deviation regarding their

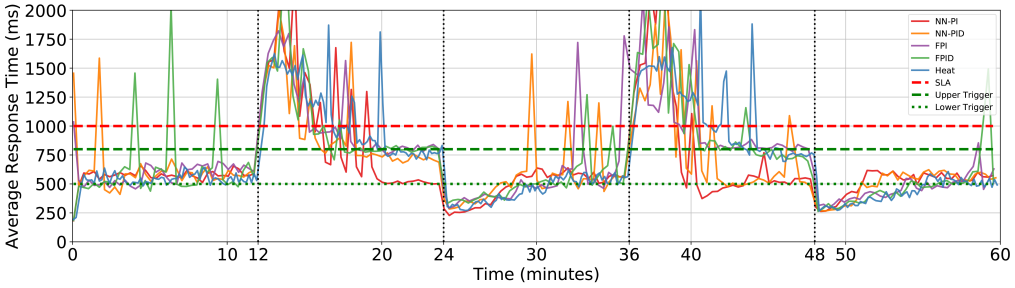
performance [27]. Finally, we compare and discuss the performance metrics for all these controllers and enumerate the pros and cons of each.

5.4 Step-Change Workload

This experiment's main objective is to investigate the efficiency of the controllers against abrupt disturbance imposed on the system. Besides, we aim to observe the effect of the PI and PID controllers' adaptability when facing a recurrent disturbance in the future.

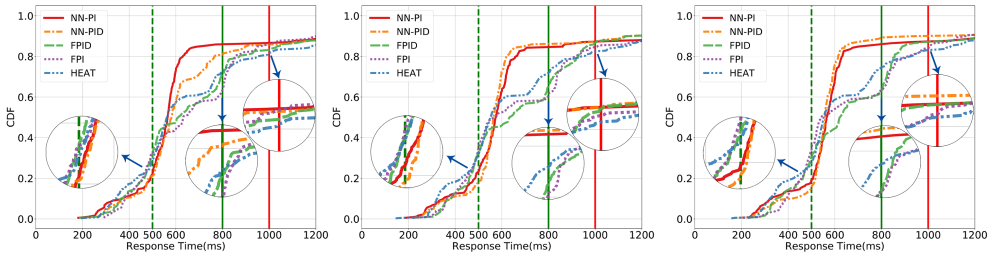


(a) Variations in the number of containers imposed by the variation in the workload.



(b) Response time of the cloud software system for step change in number of users. Vertical dashed lines indicate the time of these step changes.

Fig. 7. Comparison of five controllers in managing the response time of the cloud software system.



(a) First experiment (b) Second experiment (c) Third experiment

Fig. 8. Cumulative Distribution Function (CDF) for all the experiments.

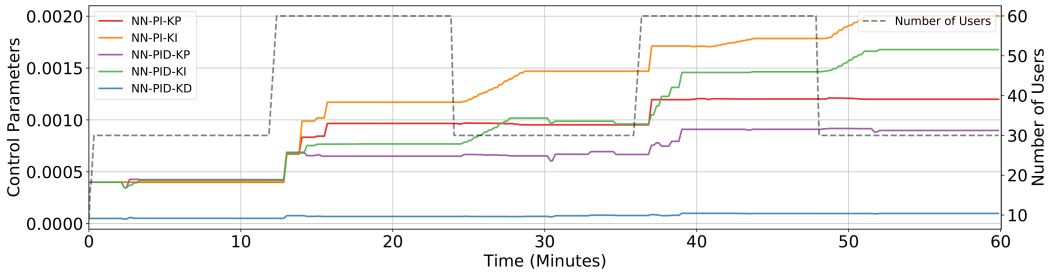


Fig. 9. Adaptation in control parameters for Step-Change workload.

Figure 7b depicts the response time of the system under study for the first experiment. Upper and lower trigger points are shown by horizontal dashed and dotted green lines, respectively. The dashed red line shows the SLA limit for the response time.

Figure 7a shows the control signal (i.e., the number of successfully provisioned containers) generated by each controller in response to the disturbance (i.e., number of users) injected to the system. In Figure 8, Cumulative Distribution Function (CDF) is shown separately for each conducted experiment. In this figure, the dashed/solid vertical green line shows the lower/upper trigger point of the desired operating region. Also, SLA is shown by a solid red line. Figure 9 shows the changes in the adaptive controller parameters in response to the step-change workload. According to this figure, the controller parameters' changes become smaller with respect to time, indicating the convergence to an optimal value. Moreover, since the controller parameters are adjusted when the number of users changes, we can conclude that the NN model has a good prediction ability.

Table 2 shows a summary of the results by presenting the average and the Standard Deviation (STD) of the three experiments performed in this study. Small STD for these three experiments indicates that the results are reproducible. Table 3 demonstrates the improvements achieved by adaptive controllers, compared to the scaling heat algorithm.

Response Time: According to Table 2, all controllers have average response-time laying in the desired operating region.

Consequently, we need to do a deeper investigation on other metrics.

Provisioning Efficiency: According to Table 2, NN-PID and NN-PI significantly decreased the resource over-provisioning ratio.

According to Table 3, our results indicate that adaptive controllers achieve an improvement of 34.46% and 22.28% for handling under-provisioning for NN-PID and NN-PI, respectively. This finding suggests that adopting the proposed method uses less resources and, consequently, less energy. This fact can be seen in Figure 8. Magnification around lower trigger point (dashed green line) reveals that both NN-PI and NN-PID are more capable of handling the problem of over-provisioning.

As reported in Table 3, similar behaviour is observed in the efficient provisioning of the resources. NN-PID and NN-PI were successful in assigning an efficient amount of resources to the cloud software application. According to the efficient provisioning percentage reported in Table 3, the efficiency of the resource allocation improved by 74.31% for NN-PID and 65.83% for NN-PI compared to the scaling heat algorithm. This observation can be made looking to both magnified lower and upper trigger points in Figure 8.

Similarly, one can confirm these improvements by looking at the system's behaviour around the upper trigger point and SLA, which is magnified for better perception in Figure 8.

From Table 2, leveraging adaptive controllers, we observed an increase in the average number of containers used over the experiments. This is mainly due to the fact that efficient resource

provisioning doesn't necessarily lead to using fewer containers or to less changes in the resources used. On the contrary, the control algorithm should instantly provide enough resources according to the control objective.

Service Level Agreement: This comparison is mainly based on the description of SLA in Section 5. According to Table 2, results confirm that control-theoretical controllers are more capable of preventing SLA violations. And among them, adaptive controllers comply with the SLA requirements more effectively. This fact is highlighted when comparing the improvements in SLA violations. As can be seen, compared to the scaling heat algorithm, SLA violations decreased by 43.2% and 49.26% for NN-PI and NN-PID, respectively.

Rise Time: In Section 5.2.3, we described the rise-time and the reason behind considering this performance metric. In this experiment, we have four rise-times, i.e., two rising edge disturbance t_1 and t_3, and two falling edge disturbance t_2 and t_4. Based on the results presented in Table 2, control-theoretical approaches considerably outperform the scaling heat algorithm. Moreover, adaptive controllers exhibited better disturbance rejection characteristics compared to the fixed PI and fixed PID controllers.

Evaluating this performance metric revealed two interesting observations. Considering the scaling heat algorithm as the baseline in Table 3, the first observation is that there is a considerable improvement in the rise time of the system adopting the proposed adaptive controller. For example, we observed an almost 50% improvement in the control system's rise time for the first rising edge disturbance. The second observation is that the second rising/falling edge disturbance's effect becomes less severe, confirming the controller's optimized behaviour for similar disturbances. This shows the adaptability of the proposed controller when dealing with future disturbances. This phenomenon can be observed in Figure 7a. After the first rising edge disturbance, NN-PI and NN-PID have the highest slope in creating containers. This slope gets even steeper for the second rising edge disturbance. Similar behaviour is seen for the falling edge.

Table 2. The average and standard deviation of the performance metrics for three conducted experiments for step-change workload.

	Controller Type				
Performance Metrics	Heat	F-PI	NN-PI	F-PID	NN-PID
avg. RT[STD] (ms)	699.82[11.30]	729.52[5.16]	672.98[10.48]	734.97[6.01]	667.07[27.19]
avg. Containers[STD]	7.50[0.02]	7.11[0.06]	9.26[0.30]	7.06[0.10]	8.86[0.06]
Over-Provisioning[STD] %	28.00[2.70]	33.88[1.18]	21.76[2.35]	31.27[0.87]	18.35[2.82]
Efficient Provisioning[STD] %	38.46[6.06]	28.83[2.27]	63.78[2.84]	35.20[4.06]	67.04[7.07]
Under-Provisioning[STD] %	9.20[1.07]	22.28[1.41]	1.31[0.85]	10.09[6.46]	2.24[2.45]
SLA Violations[STD] (s)	540.00[20.00]	413.33[11.55]	306.67[64.29]	366.67[23.09]	340.00[87.18]
t_1[STD] (s)	460.00[34.64]	386.60[50.33]	240.00[40.00]	373.40[90.18]	233.40[30.55]
t_2[STD] (s)	386.60[11.54]	293.40[11.55]	260.00[0.00]	346.60[46.18]	233.40[41.63]
t_3[STD] (s)	413.34[41.63]	373.40[100.66]	140.00[20.00]	340.00[138.56]	146.60[[11.55]
t_4[STD] (s)	393.40[11.54]	306.60[41.63]	160.00[20.00]	300.00[40.00]	140.00[40.00]

Discussion: Taking into account all the performance metrics, we can respond to research questions 2 and 3. For **RQ2**, all controllers could maintain the average response time in the desired operating region. However, the scaling heat algorithm could not adequately react to the sharp disturbances, accounting for more SLA violations. On the contrary, control theoretical methods had fewer SLA violations and reacted to the disturbances faster. Amongst control-theoretical approaches, the proposed adaptive controllers significantly outperformed the non-adaptive counterparts in both SLA and rise time.

Table 3. Improvement in the performance metrics for the first experiment compared to the scaling heat algorithm, all reported in percentage.

Performance Metrics	Improvement % NN-PI	NN-PID
Efficient Provisioning %	65.83	74.31
Under-Provisioning %	85.76	75.65
SLA Violations %	43.20	37.03
t_1 %	47.82	49.26
t_2 %	32.74	39.62
t_3 %	66.12	64.53
t_4 %	59.32	64.41

Regarding **RQ3**, considering the discussion made in provisioning efficiency, it can be concluded that adaptive controllers are more capable of handling resource management tasks by drastically improving the over-provisioning, Efficient provisioning, and under-provisioning. Furthermore, we can see that both controllers show satisfactory performance compared to the adaptive PI and PID performance. This experiment motivated us to evaluate them against a more realistic workload.

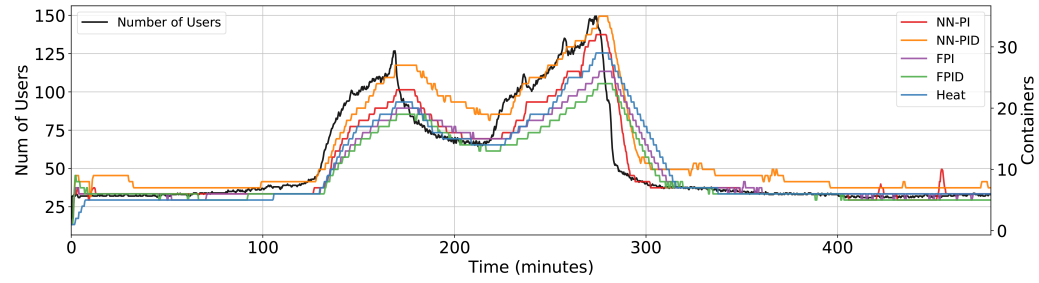
5.5 FIFA World-Cup Workload

In this experiment, exposing the cloud software system to the second workload described in Section 5.2.2, we aim to examine the performance of the proposed controllers against a more realistic workload for a longer time interval. The average response time is shown in Figure 10b. All of the thresholds are defined similarly to the step-change experiment. Figure 10a depicts the changes in the number of containers in response to the workload variations.

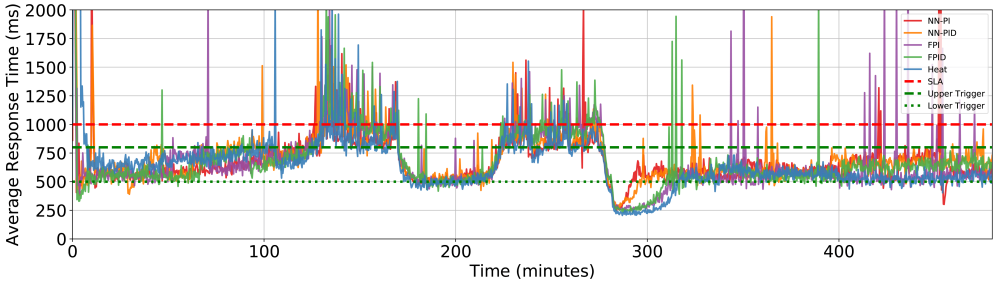
Additionally, Figure 11 presents the cumulative distribution function of response time for all these three experiments conducted for reproducibility validation. Figure 12 shows the changes in the adaptive controller parameters during the experiment. According to this figure, the variations in the controller parameters decrease with respect to time. This behaviour shows that the controller parameters are being optimized, and the proposed adaptive controller is working properly. Details of the experiments are summarized in Table 4 along with the STD of three experiments. Small STD for these three experiments indicates that the results are consistent. Furthermore, assuming the scaling heat algorithm as the baseline, improvements in the metrics are reported in Table 5 for comparison.

Response Time: According to Table 4 and Table 5, the results are in accordance with the findings of the previous experiment on the step-change workload. All the controllers could maintain the average response time in the desired operating region.

Provisioning Efficiency: Referring to Table 4, significant improvement is observed for the proposed adaptive methods in terms of the resource over-provisioning. Control theoretical approaches tend to allocate resources more cost-effectively. The over-provisioning drops from 16.04% to 5.66% and 8.87% for NN-PI and NN-PID, respectively, which results in 64.71% and 44.7% cost reduction. For the fixed versions of PI/PID controllers, this cost reduction is around 15%, which still is significant at a large scale. This can be observed in Figure 11 around the lower trigger point shown by the dashed green line. The magnified picture is presented for better perception. NN-PID and NN-PI showed lower over-provisioning. On the contrary, the scaling heat algorithm suffers from high resource over-provisioning.

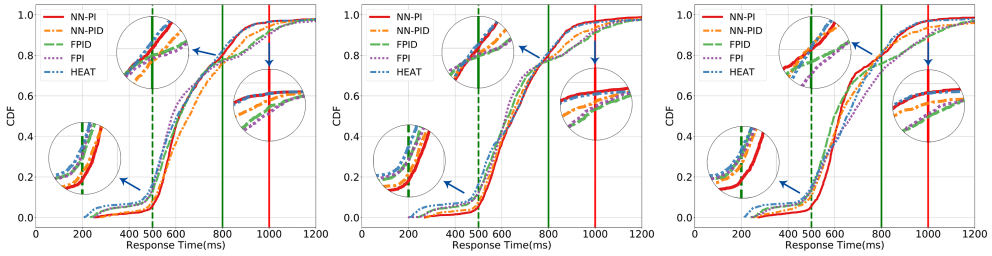


(a) Variations in the number of containers imposed by the variation in the workload.



(b) Response time of the cloud software system for FIFA Wold-Cup workload.

Fig. 10. Comparison of five controllers in managing the response time of the cloud software system against real world workload.



(a) First experiment. (b) Second experiment. (c) Third experiment.

Fig. 11. Cumulative Distribution Function (CDF) for all the experiments.

In terms of efficient provisioning, results substantiate the capabilities of the adaptive controllers in efficient resource management. According to Table 4, all control-theoretical methods allocated the resources more efficiently compared to the scaling heat algorithm. Results in Table 5 suggest 22.12% and 14.21% better resource allocation for NN-PI and NN-PID, respectively. Referring to Figure 11, around the upper trigger point (solid dashed line), all controllers maintain the response time below the upper trigger point for almost 75% of the time. A similar performance is observed in all experiments.

Although the performance was not ideal for under-provisioning, results reveal that NN-PI and NN-PID are conservative in allocating more resources than the scaling heat algorithm. This fact can be seen in Table 5 that shows the increase in under-provisioning from 15.19% to 16.76% and 15.28%.

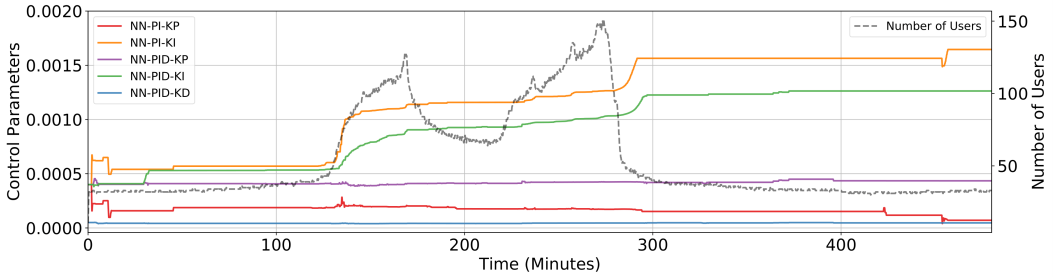


Fig. 12. Adaptation in control parameters for FIFA World-Cup workload.

However, this increase is negligible, and results show that it does not affect the SLA. Therefore, we can infer that adaptive controllers tend to use the resources close to the system capacity, without violating the SLA.

Service Level Agreement: As can be seen in Table 4, surprisingly, the scaling heat algorithm shows a decent performance compared to Fixed PI/PID and NN-PID in terms of the number of SLA violations. However, adopting NN-PI decreased the SLA violations by 27.11%, compared to the scaling heat algorithm. We found that since in control-theoretical approaches, "controller gains" decide about the magnitude of change in the number of containers, non-adaptive controllers are not suitable for slowly varying workloads. In the absence of a system model, these gains cannot be found accurately. And even if we have a model, we need to adapt the model to the environment changes. This fact highlights the need for leveraging an adaptive controller.

Table 4 suggests an inferior performance for the adaptive PID controller. This result could be because the auto-scaler has to deal with the cloud software system's stochastic behaviour, the presence of disturbances, noisy measurements, and delay in the actuator. Having a derivative term would increase the controller's sensitivity to these artifacts, leading to system instability.

According to Table 4, all controllers tend to use the same number of containers on average except NN-PID, which used 23.13% more containers. Besides, according to Figure 10a, this controller applies a greater change in the number of containers compared to other controllers with the expense of degraded performance (e.g., SLA). The reason for this is that creating/removing containers will add a computational overhead to the system. This influences the response time of the system for a while, and if this happens a lot, it will drastically deteriorate the overall performance. This behaviour contributes to the reason why 80% of the controllers prefer not to use the derivative term in the controller [3].

Mean Squared Error: Mean squared error plays a crucial role in designing the adaptive neural network-based PI/PID controller. The adaptive behaviour of the controllers is achieved by minimizing the MSE. According to Table 5 and Table 4, the optimization task is carried out successfully by causing 56.27% and 35.81% drop in MSE compared to the scaling heat algorithm, for NN-PI and NN-PID, respectively. This improvement can be observed in Fixed PI/PID as well. The reason is that the control-theoretical approaches' main objective is to decrease the system error with respect to its magnitude. On the contrary, the scaling heat algorithm ignores the magnitude of the error.

Mean Absolute Error: In addition to MSE, Mean Absolute Error (MAE) was reduced by 42.72% and 14.10% for NN-PI and NN-PID, respectively.

Note that MSE denotes the controller's performance in regulating the response time of the system and shows whether these regulation errors are significant or not in magnitude, having more emphasis on larger errors. As opposed to MSE, MAE looks at the errors with similar weights and ignores the magnitude.

Table 4. The average and standard deviation of the performance metrics for three conducted experiments for FIFA workload.

	Controller Type				
Performance Metrics	Heat	F-PI	NN-PI	F-PID	NN-PID
avg. RT[STD] (ms)	674.28[11.21]	693.54[19.13]	680.76[6.26]	684.62[1.05]	698.14[5.01]
avg. Containers[STD]	10.46[0.058]	11.33[1.61]	10.51[0.37]	10.35[1.12]	12.88[0.82]
MSE[STD] ($\times 10^4$)	11.12[1.30]	6.40[0.40]	4.86[0.48]	6.20[1.08]	7.14[1.10]
MAE[STD]	68.79[1.65]	76.06[3.16]	39.40[2.36]	72.89[4.86]	59.09[7.10]
Over-Provisioning[STD] %	16.04[0.50]	13.29[0.94]	5.66[0.20]	12.73[0.81]	8.87[1.84]
Efficient Provisioning[STD] %	60.94[0.63]	62.29[3.87]	74.42[0.25]	64.25[1.76]	69.60[1.31]
Under-Provisioning[STD] %	15.19[0.96]	14.99[2.70]	16.76[0.37]	13.75[0.53]	15.28[2.84]
SLA Violations[STD] (s)	393.33[57.74]	1,860.00[336.45]	286.67[50.33]	1,873.33[323.31]	873.33[133.16]

Table 5. Improvements in the second experiment, compared to the scaling heat algorithm.

	Improvement %	
Performance Metrics	NN-PI	NN-PID
avg. Response Time %	-0.96	-3.53
avg. Containers %	-0.47	-23.13
Mean Squared Error %	56.27	35.81
Mean Absolute Error %	42.72	14.1
Over-Provisioning %	64.71	44.70
Efficient Provisioning %	22.12	14.21
Under-Provisioning %	-10.33	-0.59
SLA Violations %	27.11	-122.03

Discussion: Exposing the controllers to a realistic workload for a longer time revealed interesting findings. For **RQ2**, the same behaviour in maintaining the average response time was observed, confirming the results of the previous experiment. Also, better MSE and MAE were achieved by the control-theoretical approaches. However, surprisingly, fixed PI/PID exhibited unacceptable performance in terms of the SLA violations. In contrast, the adaptive PI controller significantly decreased the SLA violations compared to the other controllers.

Regarding the provisioning efficiency (**RQ3**), the control-theoretical approaches present better resource management capabilities in almost all cases. However, the scaling heat algorithm performs better in terms of under-provisioning. An interesting observation is the tendency of the adaptive PID in creating/removing the containers, which causes performance overhead and, consequently, SLA violations.

Comparing both adaptive controllers, we can conclude that *adaptive PI* performs better than *adaptive PID* for real-world workloads and long term control purposes. This is mainly due to the fact that it is less sensitive to the uncertainties and noise in the system. Besides, it provides better resource management utilizing an almost equal number of containers while achieving fewer SLA violations.

6 LIMITATIONS

There are some limitations on this work. First, since we require an initial condition for our proposed controller, finding better initial conditions can lead to better performance. However, in the long-run, the optimization task will push the system to a minimized cost function. Furthermore, considering

online model identification and the highly stochastic behaviour of the cloud software systems, noisy measurements could result in incorrect online system identification or controller divergence. It is better to select the identification and controller training rate as low as possible to address this issue. However, model training and controller optimization will take a longer time.

The second limitation is the extent of the generalizability of our experiments. In this study, we selected the WordPress application as a standard three-tier application for evaluating the adaptive behaviour of our proposed adaptive PID-controller in optimizing the performance of the containerized cloud software systems. The rationale behind this decision is that WordPress is one of the main drivers for many websites nowadays. It worth noting that the focus of this study is to propose an adaptive control theoretical approach for optimizing the auto-scalers at runtime and the way we can benefit from a massive amount of collected performance data to obtain a non-linear model of the cloud software system, which can be used to optimize the performance and the auto-scaler. To the best of our knowledge, our study is the first to adopt neural networks to propose a model suitable for control theoretical auto-scalers, and we aim to pave the road for future studies in the area of cloud software system performance optimization. We expect the same efficiency and the same outcome for other applications. However, future studies may further investigate the impact of selecting different applications on the neural networks' architecture.

The third limitation of this study is the sampling rate selection. The sampling rate plays a crucial role, both in the system identification and the control mechanism. Increasing the sampling rate helps improve the reaction time to any change in the measurements while adding some overhead to the system monitoring and data acquisition. Moreover, a high sampling rate will not necessarily improve the controller's performance due to the presence of uncertainty in the measurements. Selecting different sampling rates results in different system models; therefore, we cannot make a fair comparison between these models. As a result, we relied on a widely used sampling rate, a sampling rate of 20 seconds, in the cloud infrastructures. Also, considering that the proposed controller itself is more complicated than a simple heat-based method, we impose a computation overhead to the system, which is not measured here.

Another limitation of this study is the stability analysis for the proposed controller. According to [55], the stability of a software system guarantees the ability of the system to converge to the objectives. A system can be stable without goals; however, a goal cannot be achieved in an unstable system. In our study, we defined this goal as the controller's ability to maintain the system's response time in a specific range, in other words minimizing the regulation error. For different software qualities, we have different interpretations of stability. Also, to analyze a system's stability mathematically, we require the dynamic model of the system. Since our model is data-driven, we cannot directly use it to find the attraction region of the controller.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed and evaluated various adaptive and non-adaptive controllers for containerized cloud applications. Through extensive experimentation, we identified the best adaptive controller that can optimize the performance and SLA under different parameter settings and configurations. To this end, we leveraged the power of black-box modelling, i.e., neural networks and PID controllers, to make controllers adaptive, scalable, efficient, and extendable to other containerized cloud applications. The key is to use a simple, pure reactive auto-scaler at the beginning of the software system operation and then hand over the auto-scaling to the adaptive controller when it has reached satisfactory accuracy in prediction. Afterward, the software system's performance under control will be kept optimized due to the neural network's online training and modifying the controller's parameters accordingly at run time.

Our experimental results show that the proposed adaptive PID controller works well even without a precise model for updating the controller's parameters. This makes them well suited for controlling cloud software systems.

For future studies, we plan to investigate the performance of different data-driven modelling approaches, such as Long Short Term Memory (LSTM), and introduce other optimization terms in our proposed adaptive controller's cost function,e.g. constraints on the number of containers. Moreover, we aim to study the effect of different sources of uncertainty on the performance of self-adaptive systems such as actuator uncertainties.

ACKNOWLEDGEMENT

We would like to thank Google for supporting this research by providing us the research credit to access the Google Cloud Platform.

REFERENCES

[1] Alma Y. Alanis, Jorge D. Rios, Jorge Rivera, Nancy Arana-Daniel, and Carlos Lopez-Franco. 2015. Real-time discrete neural control applied to a Linear Induction Motor. *Neurocomputing* 164 (2015), 240–251.

[2] Hanieh Alipour and Yan Liu. 2017. Online machine learning for cloud resource provisioning of microservice backend systems. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2433–2441.

[3] Kiam Heong Ang, Gregory Chong, and Yun Li. 2005. PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology* 13, 4 (2005), 559–576.

[4] Hamid Arabnejad, Claus Pahl, Pooyan Jamshidi, and Giovani Estrada. 2017. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 64–73.

[5] Davide Arcelli and Vittorio Cortellessa. 2016. Challenges in Applying Control Theory to Software Performance Engineering for Adaptive Systems. In *Companion Publication for ACM/SPEC on International Conference on Performance Engineering* (Delft, The Netherlands). ACM, New York, NY, USA, 35–40.

[6] D. Arcelli, V. Cortellessa, A. Filieri, and A. Leva. 2015. Control theory for model-based performance-driven software adaptation. In *2015 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. 11–20.

[7] M. Arlitt and T. Jin. 2000. A Workload Characterization Study of the 1998 World Cup Web Site. *Netwrk. Mag. of Global Internetwkg* 14, 3 (2000), 30–37.

[8] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. 2010. A view of cloud computing. *Commun. ACM* 53, 4 (2010), 50–58.

[9] L. Baresi and S. Guinea. 2013. Event-Based Multi-level Service Monitoring. In *2013 IEEE 20th International Conference on Web Services*. 83–90.

[10] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. 2016. A discrete-time feedback controller for containerized cloud applications. (2016), 217–228.

[11] C. Barna, M. Fokaefs, M. Litoiu, M. Shtern, and J. Wigglesworth. 2016. Cloud Adaptation with Control Theory in Industrial Clouds. In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*. 231–238.

[12] Cornel Barna, Hamzeh Khazaei, Marios Fokaefs, and Marin Litoiu. 2017. Delivering Elastic Containerized Cloud Applications to Enable DevOps. *IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2017* (2017), 65–75.

[13] Cornel Barna, Hamzeh Khazaei, Marios Fokaefs, and Marin Litoiu. 2017. A DevOps Architecture for Continuous Delivery of Containerized Big Data Applications. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE.

[14] Cornel Barna, Marin Litoiu, Marios Fokaefs, Mark Shtern, and Joe Wigglesworth. 2018. Runtime Performance Management for Cloud Applications with Adaptive Controllers. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany). ACM, New York, NY, USA, 176–183.

[15] Xiaolin Chang, Ruofan Xia, Jogesh K Muppala, Kishor S Trivedi, and Jiqiang Liu. 2016. Effective modeling approach for IaaS data center performance analysis under heterogeneous workload. *IEEE Transactions on Cloud Computing* 6, 4 (2016), 991–1003.

[16] S. Chen and S. A. Billings. 1992. Neural networks for nonlinear dynamic system modelling and identification. *Internat. J. Control* 56, 2 (1992), 319–346.

[17] Long Cheng, Weichuan Liu, Zeng Guang Hou, Junzhi Yu, and Min Tan. 2015. Neural-Network-Based Nonlinear Model Predictive Control for Piezoelectric Actuators. *IEEE Transactions on Industrial Electronics* 62, 12 (2015), 7717–7727.

[18] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[19] David Desmeurs, Cristian Klein, Alessandro Vittorio Papadopoulos, and Johan Tordsson. 2015. Event-Driven Application Brownout: Reconciling High Utilization and Low Tail Response Times. *International Conference on Cloud and Autonomic Computing* (2015), 1–12.

[20] Docker . 2018. Create a Docker Swarm manager. https://docs.docker.com/swarm/reference/manage/

[21] Jonas Durango, Manfred Dellkrantz, Martina Maggio, Cristian Klein, Alessandro Vittorio Papadopoulos, Francisco Hernandez-Rodriguez, Erik Elmroth, and Karl Erik Arzen. 2014. Control-theoretical load-balancing for cloud applications with brownout. *Proceedings of the IEEE Conference on Decision and Control* 2015-Febru, February (2014), 5320–5327.

[22] S. Farokhi, P. Jamshidi, D. Lucanin, and I. Brandic. 2015. Performance-Based Vertical Memory Elasticity. In *2015 IEEE International Conference on Autonomic Computing*. 151–152.

[23] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro V. Papadopoulos, Suprio Ray, Amir M. Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. 2017. Control strategies for self-adaptive software systems. *ACM Transactions on Autonomous and Adaptive Systems* 11, 4 (2017), 1–31.

[24] M. Fokaefs, Y. Rouf, C. Barna, and M. Litoiu. 2017. Evaluating Adaptation Methods for Cloud Applications: An Empirical Study. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. 632–639.

[25] Ian Gergin, Bradley Simmons, and Marin Litoiu. 2014. A decentralized autonomic architecture for performance control in the cloud. *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014* (2014), 574–579.

[26] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[27] Google. 2020. Benchmarks for Linux VM instances. https://cloud.google.com/compute/docs/benchmarks-linux Last accessed 2020-09-20.

[28] Google, Inc. 2019. Manage a cluster of Linux containers as a single system to accelerate Dev and simplify Ops. http://kubernetes.io/

[29] Beitao Guo, Hongyi Liu, Zhong Luo, and Fei Wang. 2009. Adaptive PID controller based on BP neural network. *IJCAI International Joint Conference on Artificial Intelligence* 2 (2009), 148–150.

[30] Heyman, Jonatan and Bystrom, Carl and Hamren, Joakim and Heyman, Hugo. 2019. Locust - A modern load testing framework. http://locust.io/ Last accessed 2019-10-16.

[31] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[32] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. 2017. Software performance self-adaptation through efficient model predictive control. *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering* (2017), 485–496.

[33] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. 2018. Combined Vertical and Horizontal Autoscaling Through Model Predictive Control. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11014 LNCS (2018), 147–159.

[34] Engin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A McKee. 2005. An approach to performance prediction for parallel applications. In *European Conference on Parallel Processing*. Springer, 196–205.

[35] Pooyan Jamshidi, Amir Sharifloo, Claus Pahl, Hamid Arabnejad, Andreas Metzger, and Giovani Estrada. 2016. Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. IEEE, 70–79.

[36] Hamzeh Khazaei, Jelena Misic, and Vojislav Misic. 2013. A Fine-Grained Performance Model of Cloud Computing Centers. *IEEE Transactions on Parallel and Distributed Systems* 24, 11 (Nov 2013), 2138–2147.

[37] Hamzeh Khazaei, Jelena Misic, Vojislav Misic, and S. Rashwand. 2013. Analysis of a Pool Management Scheme for Cloud Computing Centers. *IEEE Transactions on Parallel and Distributed Systems* 24, 5 (May 2013), 849–861.

[38] Hamzeh Khazaei, Jelena Misic, and Vojislave B Misic. 2011. Modelling of cloud computing centers using M/G/m queues. In *2011 31st International Conference on Distributed Computing Systems Workshops*. IEEE, 87–92.

[39] Hamzeh Khazaei, Jelena Misic, and Vojislav B Misic. 2011. Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems. *IEEE Transactions on parallel and distributed systems* 23, 5 (2011), 936–943.

[40] Hamzeh Khazaei, Jelena Misic, and Vojislav B Misic. 2012. A fine-grained performance model of cloud computing centers. *IEEE Transactions on parallel and distributed systems* 24, 11 (2012), 2138–2147.

[41] Hamzeh Khazaei, Jelena Misic, and Vojislav B Misic. 2012. Performance of cloud centers with high degree of virtualization under batch task arrivals. *IEEE Transactions on Parallel and Distributed Systems* 24, 12 (2012), 2429–2438.

[42] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. arXiv:cs.LG/1412.6980

[43] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. 2014. Brownout: building more robust cloud applications. *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014* (2014), 700–711.

[44] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. 2009. Automated Control in Cloud Computing: Challenges and Opportunities. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds* (Barcelona, Spain) *(ACDC '09)*. ACM, New York, NY, USA, 13–18.

[45] Martina Maggio, Cristian Klein, and Karl Erik Årzén. 2014. Control strategies for predictable brownouts in cloud computing. *IFAC Proceedings Volumes (IFAC-PapersOnline)* 19 (2014), 689–694.

[46] Saif UR Malik, Samee U Khan, and Sudarshan K Srinivasan. 2013. Modeling and analysis of state-of-the-art VM-based cloud management platforms. *IEEE Transactions on Cloud Computing* 1, 1 (2013), 1–1.

[47] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[48] Mesosphere, Inc. 2018. Marathon Recipes. https://mesosphere.github.io. https://mesosphere.github.io/marathon/docs/recipes

[49] Tommi Nylander, Cristian Klein, Karl Erik Årzén, and Martina Maggio. 2018. BrownoutCC: Cascaded Control for Bounding the Response Times of Cloud Applications. *Proceedings of the American Control Conference* 2018-June (2018), 3354–3361.

[50] Fernando Ornelas-Tellez, J. Jesus Rico-Melgoza, Angel E. Villafuerte, and Febe J. Zavala-Mendoza. 2019. *Neural Networks: A Methodology for Modeling and Control Design of Dynamical Systems.* Elsevier Inc. 21–38 pages.

[51] Alessandro Vittorio Papadopoulos, Cristian Klein, Martina Maggio, Jonas Dürango, Manfred Dellkrantz, Francisco Hernández-Rodriguez, Erik Elmroth, and Karl Erik Årzén. 2016. Control-based load-balancing techniques: Analysis and performance evaluation via a randomized optimization approach. *Control Engineering Practice* 52 (2016), 24–34.

[52] Haiyang Qian, Deep Medhi, and Kishor Trivedi. 2011. A hierarchical model to evaluate quality of experience of online services hosted by cloud computing. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops.* IEEE, 105–112.

[53] Shashank Shekhar, Hamzah Abdel-Aziz, Anirban Bhattacharjee, Aniruddha Gokhale, and Xenofon Koutsoukos. 2018. Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD).* IEEE, 82–89.

[54] Stepan Shevtsov, Mihaly Berekmeri, Danny Weyns, and Martina Maggio. 2017. Control-theoretical software adaptation: A systematic literature review. *IEEE Transactions on Software Engineering* 44, 8 (2017), 784–810.

[55] Stepan Shevtsov, Mihaly Berekmeri, Danny Weyns, and Martina Maggio. 2018. Control-theoretical software adaptation: A systematic literature review. *IEEE Transactions on Software Engineering* 44, 8 (2018), 784–810.

[56] Bartlomiej Sniezynski, Piotr Nawrocki, Michal Wilk, Marcin Jarzab, and Krzysztof Zielinski. 2019. VM Reservation Plan Adaptation Using Machine Learning in Cloud Computing. *Journal of Grid Computing* (2019), 1–16.

[57] Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Josef Spillner, and Thomas Michael Bohnert. 2017. Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems* 72 (2017), 165–179.

[58] Amjad Ullah, Jingpeng Li, Yindong Shen, and Amir Hussain. 2018. A control theoretical view of cloud elasticity: taxonomy, survey and challenges. *Cluster Computing* 21, 4 (2018), 1735–1764.

[59] W3Techs. 2019. *Usage statistics and market share of WordPress.* https://w3techs.com/technologies/details/cm-wordpress/all/all

[60] Kaiqi Xiong and Harry Perros. 2009. Service performance and analysis in cloud computing. In *2009 Congress on Services-I.* IEEE, 693–700.