Journal of King Saud University - Computer and Information Sciences 36 (2024) 101924

HOSTED BY

Contents lists available at ScienceDirect

Journal of King Saud University - Computer and Information Sciences

journal homepage: www.sciencedirect.com

Full length article

Automatic data featurization for enhanced proactive service auto-scaling: Boosting forecasting accuracy and mitigating oscillation

Ahmed Bali [a,*], Yassine El Houm [a], Abdelouahed Gherbi [a], Mohamed Cheriet [b]

[a] Department of Software and Information Technology Engineering, École de Technologie Supérieure (ETS), 1100 Notre-Dame St W, Montreal, Canada, Quebec, Canada

[b] Department of Systems Engineering, École de Technologie Supérieure (ETS), 1100 Notre-Dame St W, Montreal, Canada, Quebec, Canada

ARTICLE INFO

Keywords:
Container
Auto-scaling
LSTM
Oscillation mitigation
Data featurization
Time-series forecasting

ABSTRACT

Edge computing has gained widespread adoption for time-sensitive applications by offloading a portion of IoT system workloads from the cloud to edge nodes. However, the limited resources of IoT edge devices hinder service deployment, making auto-scaling crucial for improving resource utilization in response to dynamic workloads. Recent solutions aim to make auto-scaling proactive by predicting future workloads and overcoming the limitations of reactive approaches. These proactive solutions often rely on time-series data analysis and machine learning techniques, especially Long Short-Term Memory (LSTM), thanks to its accuracy and prediction speed. However, existing auto-scaling solutions often suffer from oscillation issues, even when using a cooling-down strategy. Consequently, the efficiency of proactive auto-scaling depends on the prediction model accuracy and the degree of oscillation in the scaling actions.

This paper proposes a novel approach to improve prediction accuracy and deal with oscillation issues. Our approach involves an automatic featurization phase that extracts features from time-series workload data, improving the prediction's accuracy. These extracted features also serve as a grid for controlling oscillation in generated scaling actions. Our experimental results demonstrate the effectiveness of our approach in improving prediction accuracy, mitigating oscillation phenomena, and enhancing the overall auto-scaling performance.

1. Introduction

The Internet of Things (IoT), which promotes integration between objects in the real world and services in the digital world, is influencing many aspects of our lives, such as health, education, and construction. These uses rely primarily on networks composed of a large number of tiny devices immersed in our environment, for example, in the form of environmental and health sensors. The use of these devices is constantly growing, leading to a massive number of devices connected to the internet (Evans, 2011). The massive data generated by IoT devices and their limitations in computing and connectivity capabilities may increase the latency of services.

Edge computing, which offloads the workload of IoT systems from the cloud to edge nodes, improves system responsiveness by minimizing latency. However, IoT devices at the edge network are typically resource-constrained and heterogeneous, which can hinder the ability to deploy services to IoT devices.

To address the heterogeneity of IoT technologies, lightweight virtualization technologies, such as containers, have been extensively employed to facilitate the deployment and management of microservices on edge IoT devices (Ahmed et al., 2019). The container can package the service program with all its dependencies into a single module. Thus, services can be run stably and faster, regardless of the operating environment. Additionally, IoT edge devices within the same cluster can share resources and communicate with each other via virtual networks thanks to container orchestration techniques such as Swarm (2022). Furthermore, kubernetes (2022) considers the concept of a pod, which is a collection of one or more containers that share

* Corresponding author.
E-mail addresses: ahmed.bali.1@ens.etsmtl.ca (A. Bali), yassine.el-houm.1@ens.etsmtl.ca (Y.E. Houm), abdelouahed.gherbi@etsmtl.ca (A. Gherbi), mohamed.cheriet@etsmtl.ca (M. Cheriet).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2024.101924
Received 7 April 2023; Received in revised form 6 January 2024; Accepted 9 January 2024
Available online 18 January 2024

network and storage resources and adhere to certain operating rules. A Pod presents the deployable primitive unit of computation.

In this context, the service deployment is presented by deploying a set of replicas (i.e., containers or pods) on the available machines that are grouped into clusters, where each replica presents an instance of the microservice to be deployed. Increasing the number of replicas improves the service's responsiveness (i.e., reduces latency) but consequently increases the usage of computation resources.

Resource usage should be further considered at the edge level, where devices are often limited regarding resources. This requires a reasonable use of computing resources while meeting user requirements (e.g., response time). Therefore, it is necessary to have, dynamically and continuously, sufficient replicas that do not exceed the need (i.e., over-provisioning) and that are not less than the need (i.e., under-provisioning).

However, current container tools (e.g., Swarm (2022) and kubernetes (2022)) need to be more elastic to automatically scale deployed services. Therefore, this reduces their ability to continuously adapt in response to the operational environment, such as the frequent changes in the requested workload (e.g., HTTP requests). In addition, existing approaches lack the proactivity aspect, which limits the system's ability to adapt appropriately to the operational environment.

Proactive auto-scaling plays a pivotal role in adapting systems to future workload demands, contrasting with reactive auto-scalers that respond to workload changes as they occur. The proactive behavior lies in accurately forecasting future workloads, enabling timely system adaptations (Lorido-Botran et al., 2014). Proactive auto-scaling employs algorithms to predict future workloads by analyzing historical data, thus setting the stage for anticipatory system adjustments.

Nonetheless, the effectiveness of proactive auto-scalers, particularly those employing time-series data analysis, depends considerably on prediction accuracy (Doan et al., 2019). Various factors influence the accuracy of the prediction, including workload patterns, sliding window sizes (Lorido-Botran et al., 2014), machine-learning models, and prediction horizons. Therefore, enhancing the performance of auto-scalers necessitates the development of solutions that enhance prediction accuracy, thereby empowering auto-scalers to orchestrate more appropriate scaling actions (e.g., scaling up or down) in alignment with actual workload dynamics.

Container-based auto-scaling solutions are still an open question that needs to be addressed, as highlighted in Qu et al. (2018) and Dang-Quang and Yoo (2021). Designing and implementing efficient auto-scalers for containerized services involve numerous challenges related to dynamic workload characteristics, resource constraints, and the inherently distributed nature of IoT nodes at the edge. Overcoming these challenges is imperative to unlock the full potential of container-based auto-scaling solutions, especially in edge computing environments where resource optimization and timely scaling actions are paramount.

Moreover, the continuous generation of actions by the auto-scaler leads to a dynamic change in the number of replicas, which generates an oscillation issue. An example of an oscillation source could be a scenario where a sudden surge in user requests prompts the auto-scaler to increase the number of replicas at a time $t-1$. However, at time t, the auto-scaler reduces the replicas as the workload momentarily decreases. Shortly after that, for time $t+1$, the load surges again, prompting the auto-scaler to increase the replicas once again. This cycle of continuous scaling up and down leads to oscillation. The oscillation of the number of replicas has considerable consequences on the system's performance, such as increased costs in resource usage in the case of over-provisioning and performance declines in the case of under-provisioning. Moreover, reactive adjustments to the number of replicas can induce latency in the system.

Related work generally adopts the solution based on the cooling down strategy, which introduces a delay before carrying out a scale-down request following a decrease in the workload volume. However, this strategy is demanding since it depends on optimizing the cooling down period parameter. If the period is long, it generates more over-provisioning; if it is short, that reduces the efficiency of the oscillation mitigation.

To address the abovementioned issues, this work presents a novel service auto-scaling approach to sustain desired performance levels while optimizing resource utilization in the face of dynamic workload changes (i.e., increase/decrease). The core of our approach is based on several key contributions, outlined as follows:

- **Adoption of the MAPE-K framework:** We have effectuated the different phases of the MAPE-K (Monitor, Analyze, Plan, and Execute) controlling loop, providing proficient automatic system monitoring, data analyzing, planning adaptation actions, and executing auto-scaling operations.
- **Data Featurization Approach:** Our proposed data featurization mechanism significantly improves the accuracy of the forecasting model by transforming univariate time-series data, containing only the workload data, into a multivariate format enriched by automatically extracting relevant features. Inspired by the technique of Japanese candlesticks prevalent in trading, this transformation captured additional information, thereby augmenting the model's predictive performance. The experiments show that the featurization process led to a significant relative improvement, with gains exceeding 70%, highlighting its effectiveness in enhancing forecasting accuracy.
- **Grid-based Oscillation Mitigation Strategy:** We introduced an original grid-based approach for mitigating oscillations during system auto-scaling. Leveraging the data and features harnessed for workload prediction enhancement, this mechanism, rooted in the economic grid technique concept, effectively curtails oscillation issues. Notably, our strategy is parameterless, delineating an essential departure from existing techniques in related works.
- **Comprehensive Evaluation:** Validating the efficacy of our proposed approach, we conducted exhaustive evaluations employing widely recognized datasets prevalently used in auto-scaling literature. Our approach demonstrated a marked improvement in workload forecasting and auto-scaling performance metrics, substantiating the merits of our contributions.

The remainder of the paper is organized as follows. Section 2 reviews the literature, highlighting research issues and gaps, and positions our work relative to existing studies. Then, Section 3 shows the overall architecture of our auto-scaling process, followed by the presentation of the collection and pre-processing of data in Section 4. Afterward, Section 5 presents our featurization approach. Our LSTM model for forecasting the future workload is presented in Section 6. Section 7 presents our oscillation mitigation approach. Section 8 discusses the experiments to evaluate our contributions. Finally, we conclude this study and highlight our directions for future work in Section 9.

2. Literature review

This section outlines a literature review fundamental to our research contributions, covering various facets of IoT system deployment in edge environments: virtualization, auto-scaling, workload forecasting, and oscillation mitigation. Each subsection includes a brief background, reviews existing work, and highlights the gaps our research addresses, thereby elucidating the originality and relevance of our proposed approach.

2.1. Virtualization and autoscaling

Many works have been performed on auto-scaling at the cloud level in the literature. In Kovács (2019), auto-scaling is defined as a method used in distributed computing, especially in the cloud, to dynamically and automatically adjust the computing resources in a

set of servers based on traffic workload. Additionally, auto-scaling is a crucial component of orchestration regarding policy and flexibility for cloud containers and virtual machines. To explain the auto-scaling feature, we take as an example (similar to that presented in Mishra et al. (2020)) the number of servers running behind a web application that can be automatically increased or decreased depending on the number of active users. Since these measurements vary considerably during the day and servers are a limited resource, operating a sufficient number of servers to support the current workload is often worthwhile. Auto-scaling is very useful for meeting customer service requirements. It reduces the number of active servers when activity is low and launches new servers when activity is high.

The cloud is based on virtualization technology, which allows for executing multiple work environments on the same server. In this regard, Varghese and Buyya (2018) and Pahl et al. (2017) explain, in a simple and detailed way, the virtualization architecture, presenting the trends and technologies involved in the cloud. Containers are a fundamental virtualization property, allowing for deploying microservices on the cloud server. Several domains increasingly use containers, including service meshes, edge and fog computing, IoT, smart cars, and smart cities, as in Ahmed et al. (2019), Jamshidi et al. (2018), Khazaei et al. (2017) and Morabito et al. (2017).

There are two types of autoscaling, vertical autoscaling and horizontal autoscaling (Al-Dhuraibi et al., 2017). The distinction between these two types of auto-scaling stems from how computing resources are added to the infrastructure. In vertical autoscaling, computing power is added to existing replicas/nodes. In contrast, horizontal autoscaling increases a system's capacity by adding more replicas (e.g., containers) to the environment, allowing for processing and memory load sharing across multiple devices.

In the edge computing context, resources are often limited (e.g., Gateway devices), which makes having a mechanism for increasing and decreasing computational resources (i.e., vertical auto-scaling) on the same node less valuable or even unrealistic. Therefore, horizontal auto-scaling becomes more suitable by dynamically changing the number of replicas (e.g., containers or pods) to distribute the processing load among devices that constitute a so-called cluster.

Furthermore, auto-scaling approaches are classified into two types: reactive and proactive. As in our previous work (Bali et al., 2020), the reactive auto-scaler reacts to the current workload or resource utilization according to predefined rules and thresholds. In proactive auto-scaling, an algorithm forecasts future workload based on historical data (Lorido-Botran et al., 2014). The difference is that, in proactive mode, the auto-scaler must predict workloads to adapt the system to future needs, while in reactive mode, the system reacts to current workload changes (Lorido-Botran et al., 2014).

Due to the ease of implementation, most current auto-scalers use reactive threshold-based approaches, as used in Kubernetes HPA, Google Cloud Platform, Amazon EC2, and Oracle Cloud. For this purpose, many studies (Klinaku et al., 2018; Taherizadeh and Stankovski, 2019; Nguyen et al., 2020) suggest using the reactive auto-scaling functionality offered by cloud servers. However, selecting appropriate thresholds is difficult, especially when dealing with complex workloads (Imdoukh et al., 2019). To optimize the configuration of thresholds, auto-scalers can use static heuristic techniques offline according to predefined workloads (Zhong and Buyya, 2020). These strategies are unable to cope with highly dynamic workloads in which applications must scale at runtime (Zhong et al., 2022).

In addition, although this improved reactive approach is simple, it is less efficient since it generates oscillations due to sudden and unpredictable workload changes. As a result, the reactive approach results in waste due to the over-provisioning of resources and degradation of the system's performance when releasing the resources that the system needs.

Our work proposes a proactive auto-scaling approach that enhances the accuracy of workload forecasting and overall performance.

2.2. Workload forecasting techniques

In Dang-Quang and Yoo (2021), the authors demonstrate that their proactive auto-scaler outperforms Kubernetes' default horizontal autoscaling pod (HPA) regarding accuracy and speed when provisioning and de-provisioning resources.

In proactive auto-scaling as in Sangpetch et al. (2017), Imdoukh et al. (2019) and Dang-Quang and Yoo (2021), machine learning algorithms are often applied in time-series analysis for workload forecasting. Different ML algorithms are used to predict the future from historical data (Lorido-Botran et al., 2014). The time-series-based forecasting approaches offer more performance than regular regression approaches specifically designed for forecasting tasks. These approaches explicitly consider the sequential nature of the data and incorporate past observations to make predictions, taking into account temporal patterns. In contrast, regular regression models, such as Linear and Polynomial Regression, typically assume a constant relationship between the input and target variables without explicitly accounting for these temporal patterns. The literature uses two common categories of time-series data analysis and forecasting methods.

First, algorithms based on statistical time-series analysis (e.g., ARIMA) are widely used, such as those presented in Lorido-Botran et al. (2014), Sangpetch et al. (2017), Calheiros et al. (2014), Roy et al. (2011), Kan (2016), Li and Xia (2016), Ciptaningtyas et al. (2017) and Meng et al. (2016). These statistical approaches, while effective, are slower in dynamic workload environments and often result in resource overuse (Imdoukh et al., 2019). Most of these works are primarily intended for cloud environments, so the application of these techniques in edge computing is limited due to resource constraints.

Second, there are solutions based on advanced machine learning, such as neural networks (ANN) and LSTM algorithms have been explored, for instance, in Calheiros et al. (2014), Goli et al. (2021), Zhu et al. (2019), Saxena and Singh (2022), Kumar et al. (2021), Imdoukh et al. (2019) and Dang-Quang and Yoo (2021). The study in Imdoukh et al. (2019) shows that the LSTM model not only matches the accuracy of the ARIMA model, but also offers faster prediction speeds. Additionally, recent advances include the utilization of bidirectional LSTM networks (BiLSTMs), gated recurrent units (GRUs), and convolutional neural networks (CNNs) for workload forecasting. Using Bi-LSTM for predicting future HTTP workloads is examined in Dang-Quang and Yoo (2021). In contrast, Mozo et al. (2018) introduces a short-term network traffic forecasting method using CNNs, which incorporates a multiresolution input strategy and separate convolutional channels for different data granularities.

Other studies have combined various prediction algorithms to enhance generalization capabilities and employed attention mechanisms to refine prediction accuracy. For example, Wang et al. (2020) proposes a probabilistic method using Gaussian process regression (GPR), complemented by a stack of three forecasting models: random forest (RF), LSTM, and linear regression (LR). Recent studies have explored innovative architectures and deep learning techniques for workload forecasting. For instance, Patel and Bedi (2023) introduces MAG-D, a complex architecture incorporating Bi-GRU layers, a Bi-LSTM layer, and a multivariate attention layer for cloud workload forecasting. Similarly, Dogani et al. (2023) uses CNN and GRU networks to extract spatial and temporal features, thereby aiming to improve forecasting accuracy.

However, the effectiveness of proactive auto-scalers is highly dependent on prediction accuracy (Doan et al., 2019), which in turn depends on factors such as workload patterns, history windows (Lorido-Botran et al., 2014), the machine learning model, and the prediction horizon. Therefore, enhancing prediction accuracy remains a crucial and open issue in container-based autoscaling.

To address this challenge, our approach introduces an automated solution based on feature extraction (i.e., featurization) from data during the data processing phase. We extract features (such as maximum and

minimum values) to provide a comprehensive description of the data window, which then serves as input for the forecasting model, LSTM, to predict future workloads. Including these features significantly enhances the prediction model's accuracy, as detailed in the evaluation section (Section 8). In contrast to solutions like Imdoukh et al. (2019) and Dang-Quang and Yoo (2021), our method enhances the workload data with automatically generated features. This automated feature generation process simplifies our model's data preparation requirements compared to traditional multivariate approaches, which often necessitate collecting additional features, as we will discuss in the subsequent subsection.

2.3. Time-series featurization

While many auto-scaling studies have traditionally focused on univariate time-series data, such as workload, it has been widely recognized in the literature that incorporating multivariate data (i.e., features) can significantly improve forecasting accuracy. Cetinski and Juric (2015) demonstrated the importance of extending training data with relevant features, such as the time of day and weekends.

In the context of auto-scaling, LSTM models have been shown to effectively capture complex non-linear feature interactions when applied to multivariate data with numerous dimensions and a substantial volume of data (Ogunmolu et al., 2016). Laptev et al. (2017) proposed a novel LSTM architecture that leverages an autoencoder for feature extraction, achieving superior performance compared to the vanilla LSTM model. In their data preparation process, they incorporated additional specific features such as weather information (e.g., precipitation, wind speed, temperature) and city-level information (e.g., current trips, current users, local holidays). However, most of these additional features cannot be automatically extracted and need to be logged during data collection.

Various classical statistical time-series features have been considered in the literature to improve forecasting accuracy. Hyndman et al. (2015) explored features such as mean, variance, ACF (Auto-correlation Function), trend strength, linearity, peak, and season. Di et al. (2012) focused on important and predictive statistical properties of host load, including mean load, load fairness index, noise-decreased fairness index, and N-segment pattern. However, these derived features, particularly those related to trend and seasonality, usually require manual analysis to identify their parameters.

To further illustrate the significance of incorporating relevant features, Wang et al. (2021) established a dataset by collecting features of complex system simulation to improve the performance of the resource prediction of simulation applications in the cloud. These features include average, maximum, and minimum values of usage metrics such as CPU, memory, file system, network (receive and send bytes), communication delay, and execution time. Similarly, Kao et al. (2020) focused on communication metrics, specifically incoming traffic, outgoing traffic, number of connections, and network traffic load (per day). These features need to be obtained during data logging since they are not derived automatically.

In our approach, instead of relying on pre-existing collected multivariate (i.e., features) such as CPU usage and number of HTTP requests, we propose an automatic extraction of new features from each collected data. Specifically, in this work, we showcase the effectiveness of our automatic feature extraction method, which enriches univariate data, such as HTTP requests, by adding new features using non-linear functions. Drawing inspiration from Japanese Candlesticks, a technique widely used in the trading domain, we apply this featurization technique to each data window. This automatic generation of features eliminates the need for manual analysis of statistical properties, particularly trend and seasonality parameters, thereby streamlining the data preparation process and enhancing the accuracy of time-series forecasting.

2.4. Oscillation mitigation

Another essential aspect to consider is the continuous action generation by the auto-scaler, which can lead to frequent changes in the number of replicas, resulting in oscillation issues that waste resources. For example, at time t, the auto-scaler may reclaim a resource just released at time $t-1$, then consider re-releasing it at time $t+1$. This frequent toggling cannot only waste resources, but also introduce response lags to the system. Such continuous action changes might arise due to workload fluctuations or inaccurate predictions. Oscillation mitigation aims to reduce the number of unnecessary changes in the number of replicas, thereby improving system performance and reducing resource wastage.

Unfortunately, oscillation mitigation has not received sufficient attention in the literature. To overcome this limitation, Imdoukh et al. (2019) and Dang-Quang and Yoo (2021) have integrated the oscillation mitigation technique into the self-adaptive and autonomous MAPE-K loop system (Arcaini et al., 2015). Therefore, we compared the results in our study with those in Imdoukh et al. (2019) and Dang-Quang and Yoo (2021), as they had good results on the data analysis and implementation of an auto-scaling system, as well as they considered the oscillation mitigation issue.

Our grid-based oscillation mitigation approach benefits from the generated features. The feature values form a value grid, where each line (i.e., value) represents a reference action. This grid enables matching the actions generated by our auto-scaling system to the reference actions to reduce the oscillation issue. We have called this original method of handling oscillation 'Grid-based oscillation mitigation'. Our approach has a further advantage, as the grid values change dynamically according to the historical data window used. Approaches in the literature often use the cooldown timer (CDT) principle (Imdoukh et al., 2019), which delays the execution of a scaling-down request due to the decreased workload volume. However, this CDT solution requires finding an optimized delay timer value. In contrast, our oscillation processing approach is less demanding since it is a parameterless mechanism, and it could improve the oscillation mitigation compared to the related work approaches. Moreover, combining our grid-based approach with the CDT mechanism significantly improves the oscillation mitigation.

Finally, it is worth noting that our approaches of featurization improving the forecasting accuracy and the oscillation mitigation can be used in other scientific fields where there is a need to make more accurate time-series forecasting, such as the transportation domain (Nguyen et al., 2018), where there is a need for traffic flow prediction and the networks for autonomous and proactive resource management.

3. Overall architecture

Our approach adheres to the MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) framework (Computing et al., 2006) to ensure effective auto-scaling. As depicted in Fig. 1, MAPE-K outlines the general auto-scaling process in our context. Initially, the auto-scaler monitors the system by logging relevant measurement data (e.g., number of HTTP requests and CPU usage) during the monitoring phase. These historical data constitute the time series data, crucial for model training and forecasting.

In the analysis phase, the auto-scaler evaluates the system state and forecasts future workload using our LSTM model. This forecasting model capitalizes time-series workload data, such as the number of HTTP requests, collected in real-time from the deployed system. A key component of this analysis phase is our unique featurization approach to time-series data, designed to enhance the accuracy of workload prediction. The output of the model is a prediction of future workload, for example, the estimated number of HTTP requests for the upcoming minute.

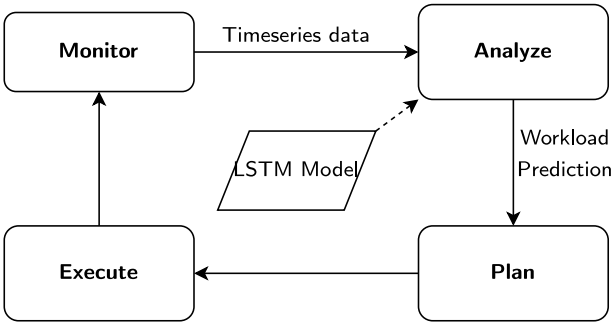


Fig. 1. General auto-scaling process.

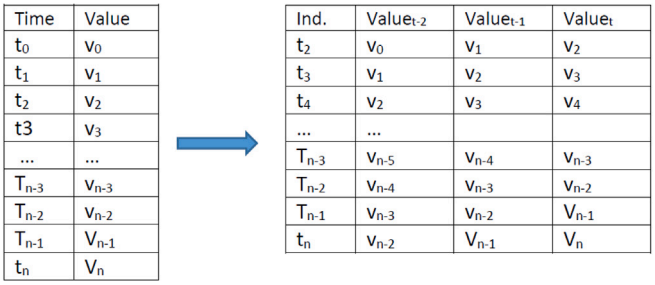


Fig. 2. Time-series reframing, with window size = 2.

Subsequently, the planning phase generates a plan that includes scaling actions, focusing primarily on adjusting the number of service replicas. The generated plan is designed to address the oscillation issue effectively. Finally, the Execute step of the MAPE-K loop executes the generated plan. In our architecture, the LSTM-based prediction model is an integral part of the shared knowledge in the MAPE-K loop, as illustrated in Fig. 1.

4. Data collection and pre-processing

The data collection is based on the monitoring phase. In our case, we used univariate time series data. We organized the dataset to have only two valuable pieces of information: the time (period) and the count of HTTP requests. Thus, the overall dataset is aggregated and transformed such that each record represents the total workload (i.e., number of HTTP requests) per minute.

Time-series data can be collected and stored by the Prometheus tool, which aggregates metrics from monitoring tools such as CAdvisor and Node Exporter.

We use the Worldcup98 Dataset (Arlitt and Jin, 2000) and NASA dataset (Dang-Quang and Yoo, 2021) to evaluate and compare our forecasting approach to those in the literature. The Worldcup98 Dataset contains the HTTP request logs for approximately 1.3 billion total requests made to the FIFA World Cup Website between April 30 and July 26, 1998. In contrast, the NASA'95 dataset contains a two-month log of all HTTP requests made to the Florida NASA Kennedy Space Center Web server. These datasets have been used extensively to evaluate auto-scalers in the cloud computing literature (Imdoukh et al., 2019). We will further present the used datasets in the evaluation section (Section 8).

4.1. Data scaling

Scaling the data can increase the performance of some ML algorithms, such as LSTM, in our case. Scaling involves adjusting the values of numeric variables to achieve a common scale. As a result, data is transformed to be bounded within a newly defined range, such as [0, 1], using the min–max scaling mechanism as presented in Eq. (1).

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1}$$

where x', x, $x_{\min}$, and $x_{\max}$ represent the scaled value, original value, minimum value of the feature in the dataset, and maximum value, respectively.

4.2. Data reframing and horizon of prediction

The data reframing step is essential for transforming the initial time-series workload data, which typically consists of a sequence of logged workload values (such as HTTP requests), into a format suitable for building a supervised learning-based forecasting model. Reframing involves using the workload values from previous time steps as input variables and the workload value from the next time step as the output variable. As illustrated in Fig. 2, the original data is presented as a chronological list of time-series data points. Reframing converts these data into a sliding window format comprising two values. Each sliding window in this format effectively captures a time series segment, providing the necessary context for the forecasting model to predict future values based on observed historical patterns.

The data prepared in the previous step are sufficient for a single-step prediction. For the case of multi-step prediction (e.g., next fifth-time step), we assign the corresponding output value in the training process. However, this approach is not practical, since it may require many models that match the size of the forecasting horizon. For example, five models are needed for a prediction horizon of five-time steps. Another alternative is the recursive multistep prediction (Imdoukh et al., 2019). However, its limitation is the degradation of accuracy as the size of the horizon increases.

5. Our time-series featurization

The analysis phase of the MAPE-K loop (in Fig. 1) is based on forecasting the future workload. Predicting the future workload is essential for making the auto-scaling process proactive. This prediction is based on quantitative forecasting using the collected and pre-processed data from the monitoring step, representing the historical workload data. We propose adding a featurization phase to improve the accuracy of the forecasting algorithm (LSTM in our case). The time-series featurization enriches the time-series data, which initially contains only workload information, by extracting new relevant information from the data, as will be presented in the following subsection.

5.1. General description and motivation of our featurization approach

Our approach adds a step of time-series featurization (i.e., feature extraction) to improve prediction accuracy. The added features are derived from the input data, mainly summarizing the time-series window data.

To formulate the data featurization, we first define the workload data at a specific instant t as d_t. Consequently, time-series data TS can be expressed as n-tuple:
$TS = (d_0, d_1, \dots, d_n)$. The featurization can be modeled as a function:

$$\begin{aligned} &F : D^s \rightarrow D^k \\ &W_t = (d_{i-s}, \dots, d_{t-1}, d_t) \rightarrow F_t = (f_{t1}, f_{t2}, \dots, f_{tk}) \end{aligned} \tag{2}$$

where D represents the domain of the data value (e.g., integer or real), the window at time t, W_t, contains the s previous values, and the function generates F_t of k features.

Building upon our featurization approach, the input to the prediction model, denoted as the function P, is a combination of the data window W_t and its corresponding extracted features F_t, as defined in Eq. (2). Eq. (3) formalizes this integration:

$$\begin{aligned} &P : D^s \times D^k \rightarrow D^h \\ &(W_t, F_t) \rightarrow \bar{Y}_i = (\bar{y}_{t+1}, \bar{y}_{t+2}, \dots, \bar{y}_{t+h}) \end{aligned} \tag{3}$$

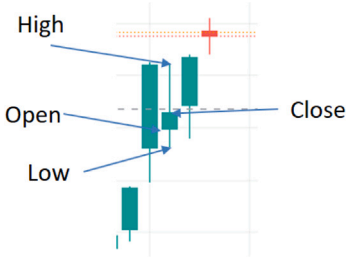


Fig. 3. Japanese candlestick information.

where W_t and F_t represent the data window and the extracted feature, respectively, as outlined in Eq. (2). The function $P(W_t, F_t)$ predictions spanning a horizon of h steps. In scenarios where the prediction is focused on a single step ahead, the function simplifies to $P(W_t, F_t) = \bar{y}_{t+1}$.

Our featurization methodology draws inspiration from the Japanese Candlestick concept (Tam, 2015) commonly utilized in financial trading of different assets such as stocks, providing helpful information like open, low, high, and close, as presented in Fig. 3. Formally, the information of the Japanese candlestick can be represented as a 4-tuple (Open(p), High(p), Low(p), Close(p)), where Open, High, Low, and Close represent the first value, the maximum value, the minimum value, and the final value in the period p, such as a day. The information provided by a sequence of Candlesticks offers an abstraction, allowing traders to comprehend stock evolution better. Fig. 4(a) presents the line representation corresponding to the sequence of Tesla stock price values (i.e., time-series data), while Fig. 4(b) presents the equivalent representation based on the Japanese candlesticks, where each candlestick summarizes a one-day period.

5.2. Time-series featurization algorithm

Algorithm 1 Feature Extraction.

Require:
1: $TS = (d_1, d_2, ..., d_n)$
2: s : window size
Ensure: $reformulatedData$

3: $reformulatedData \leftarrow \{\}$
4: **for** $i \leftarrow 1$ to $n - s$ **do**
5: $w \leftarrow (d_i, ..., d_{i+s})$
6: $open \leftarrow d_i$
7: $close \leftarrow d_{i+s}$
8: $low \leftarrow \min(w)$
9: $high \leftarrow \max(w)$
10: $w \leftarrow w || [open, close, low, high]$
11: reformulatedData.append(w)
12: **end for**

To map the candlestick concept to our context, we consider that the window is the equivalent to the period (ex. 3 m, 15 m, one day, etc.).

Given the window: $w = (d_{t-s}, d_{t-s-1}, \dots, d_t)$, where s is the size of the window, the candlestick features are obtained by the Eqs. (4), (5), (6), and (7).

$$Open = d_{t-s} \tag{4}$$

$$Close = d_t \tag{5}$$

$$Low = \min(w) \tag{6}$$

$$High = \max(w) \tag{7}$$

Fig. 5 presents an extract of time-series data organized in periods with its features.

Other information, such as indicators, can be added like the average (Eq. (8)).

$$Average = \frac{1}{s} \sum_{i=0}^{s-1} d_{t-i} \tag{8}$$

Our time-series featurization process is presented in Algorithm 1. The data sequence is passed in parameters. The window size is used to reformulate the data in Line 5. Lines 6–9 extract the features, which are concatenated to the initial window in Line 10.

The time complexity of the feature extraction part of the algorithm is linear (i.e., $O(n)$) to the input size, which corresponds to $|features| \times w_z$, where $|features|$ is the number of features to extract and w_z represents the window size.

5.3. Rationale behind our feature extraction

The primary aim of drawing inspiration from the Japanese Candlestick is to enhance prediction accuracy by deriving features that encapsulate important information from time-series data. In our approach, the period represents the sliding windows (Fig. 2).

As mentioned in Chernikov et al. (2022), the features can be used as auxiliary information to achieve better accuracy. Our approach has the advantage of automatically generating dynamic features that capture relevant information. This featurization provides time-based features, like open (i.e., the first value of a window) and close (i.e., the last value of a window), as well as value-based features, like maximum and minimum. The extracted features describing the data presented in the window allow for capturing short-term dependencies, while the sequence of these features enables capturing long-term dependencies.

To present the importance of the extracted features by our featurization approach, we analyze their correlation in two datasets, WorldCup'98 and NASA'95, which are used for our experimentation. Fig. 6 illustrates the correlation between various features extracted using our approach and the target variables over a predictive horizon of 15 lag steps. This correlation analysis demonstrates the relationship between extracted features and the target variables, highlighting their potential impact on enhancing prediction accuracy.

On the other hand, our approach avoids generating linear dependences between the extracted features and the input variables (i.e., data window), which could potentially cause instability or overfitting in the models. The non-linear functions (e.g., max and min) applied to the data introduce variations in the derived features, leading to a diversity of information. This benefits the forecasting model as it captures different aspects of the underlying patterns.

6. LSTM model

LSTM is a multi-layer learning model and an advanced type of recurrent neural network (RNN). Recurrent neural networks are distinguished by an ability to memorize from prior inputs to influence the current input and output. Due to the vanishing gradient problem (Hu et al., 2018), RNNs tend to forget what they have seen in previous layers and do not learn appropriately in cases of long-term dependency. LSTM overcomes this drawback of RNN by using gate mechanisms that control the information flow. Thus, LSTM is highly suited for predicting the subsequent sequence in time-series data, such as workload over time. An LSTM network connects many LSTM units (i.e., cells) together.

An LSTM unit comprises an internal memory controlled by three gates: input, forget, and output. The role of each gate is to regulate the volume of data that passes through it. The input gate decides whether the input should modify the cell's content, with its output C'_t obtained by Eqs. (10) and (11). The forget gate determines whether to reset the cell content to 0; its output corresponds to f_t calculated by Eq. (9). The output of these two gates forms the basis for calculating the new

(a) Using line representation

(b) Using Japanese candlesticks

Fig. 4. Representation of Tesla stock.

...	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)	open	low	high	close
...	829.0	837.0	749.0	805.0	874.0	604.0	2.0	1.0	1343.0	604.0
...	837.0	749.0	805.0	874.0	604.0	806.0	11.0	1.0	1343.0	806.0
...	749.0	805.0	874.0	604.0	806.0	731.0	16.0	1.0	1343.0	731.0
...	805.0	874.0	604.0	806.0	731.0	1000.0	7.0	1.0	1343.0	1000.0
...	874.0	604.0	806.0	731.0	1000.0	828.0	3.0	1.0	1343.0	828.0
...	...	...	...	...	...	...	...	...	...	...

Fig. 5. Overview of the proposed multivariate data structure.

cell state, C_t, as presented in Eq. (12). Finally, the output gate decides whether the cell content (i.e., C_t) should impact the output of the cell h_t as presented in Eqs. (13) and (14).

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{9}$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{10}$$

$$C'_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{11}$$

$$C_t = f_t \times C_{t-1} + i_t \times C'_t \tag{12}$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{13}$$

$$h_t = o_t \times tanh(C_t) \tag{14}$$

where W and b represent the weights and bias, respectively. In addition, σ (sigmoid) and $tanh$ denote the used activation functions.

Our proposed approach employs a sequential architecture of LSTM units arranged in a pipeline, as depicted in Fig. 7. The size of the input layer is designed to correspond to the sliding window, where each unit sequentially receives a specific input x_i. In contrast, the output layer consists of a single unit, which produces the predicted value x'_{t+1}. The architecture encompasses a hidden layer populated with a set of LSTM units (e.g., ten units). The Adam optimizer is utilized in the optimization process, fine-tuning the model's parameters based on the training data. Table 1 delineates the configurations of our primary LSTM model. In particular, Section 8 evaluates our approach by experimenting with various hyperparameter configurations, such as the number of units and epochs of the hidden layer, to compare robustly with related work.

Table 1
Hyperparameters of our primary LSTM model.

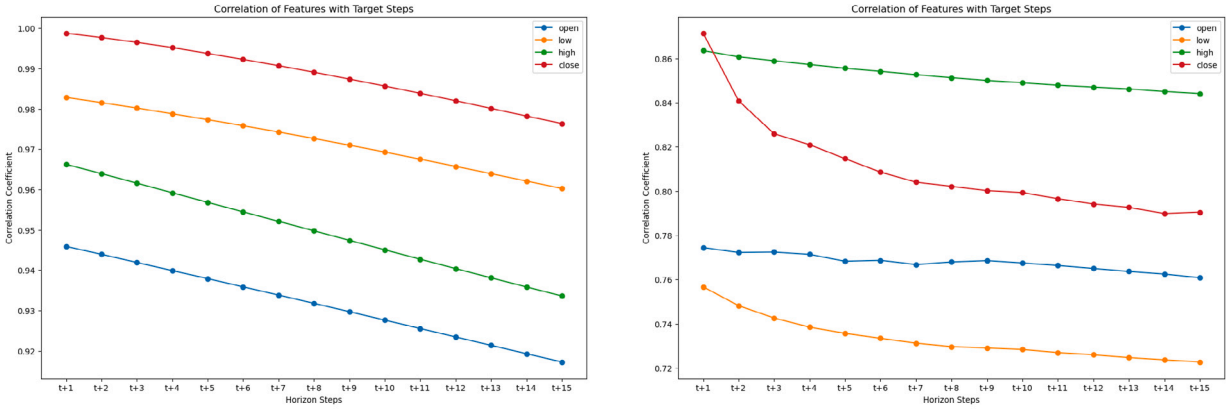
Parameter	Value
Input size	17
Output size	1
Hidden units	30
Loss function	MAE
Optimizer	Adam
Batch size	512
Epochs	50

7. Grid-based oscillation mitigation

Oscillation in auto-scaling arises primarily due to frequent and sudden changes in the number of replicas generated by the auto-scaler in response to workload fluctuations or inaccurate predictions. Such oscillation can lead to resource waste and introduce response lags to the system, affecting overall efficiency and performance. Existing solutions like the cooldown timer (CDT) and scaling down ratio (SDR) used in Dang-Quang and Yoo (2021) and Imdoukh et al. (2019) require parameter tuning, and their performance depends on the precise values of these parameters (i.e., CDT and SDR).

In contrast, our grid-based approach for oscillation mitigation offers a parameterless solution by utilizing features extracted during the forecasting phase. These extracted features form grid lines that aim to reduce action volatility and consequently mitigate oscillation. Fig. 8 shows an illustrative example of a grid of a window of s size, with grid lines corresponding to four Japanese candlestick features: open (first value), low (minimum value), high (maximum value), and close (last value).

Suppose the model predicts a value x'_{t+1} less than the current value x_t, indicating a request for resource reduction (i.e., down-scaling). In that case, our approach selects the value of the nearest higher grid line instead of directly using the predicted value. For instance, if the next



(a) WorldCup Dataset (b) NASA Dataset

Fig. 6. Correlation analysis of extracted features with a 15-Step predictive horizon in WorldCup'98 and NASA'95 datasets.

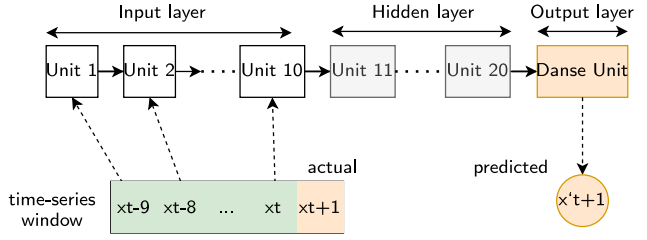


Fig. 7. Illustration of our LSTM architecture, simplified with example values for input and hidden layer sizes.

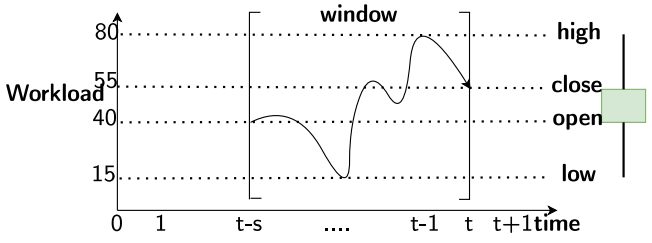


Fig. 8. A grid based on four features with the workload curve.

value $x'_{t+1} = 50$, the Planner selects the line value of 55 (i.e., close line). Similarly, if $x'_{t+1} = 35$, the planner selects the line value of 40 (i.e., open line), thereby ensuring a significant reduction while mitigating the oscillation effect.

Algorithm 2 outlines the steps involved in this grid-based oscillation mitigation approach. At each execution, the algorithm estimates the future workload using a prediction model (e.g., LSTM), as shown in Line 6.

When a system downscale is needed, our approach adjusts the workload value to a value from the feature set, as obtained in Line 2. The feature set values, representing the grid lines, are generated by Algorithm 1. The function 'getNextRoof' (Lines 11–14) returns a value from the grid that is the smallest value greater than the predicted workload value, aiding in a controlled reduction of replicas. Subsequently, the algorithm computes the required number of replicas (Line 10) by dividing the predicted workload value by the replica capacity, which signifies the workload capacity of a replica. Specifically, *replicaCapacity* represents the number of requests a replica can process within a given time frame (e.g., one minute).

Algorithm 2 Grid-based Oscillation Mitigation

Require:
1: $w =< d_{t-s}, ..., d_{t-1}, d_t >$
2: $featureSet = \{low(w), open(w), close(w), high(w)\}$
3: $ForecastingModel$
4: $replicaCapacity$
Ensure: $newReplicas$

5: $currentWorkload \leftarrow d_t$
6: predNextWorkload ← predict(ForecastingModel, w)
7: **if** (predNextWorkload < currentWorkload) **then**
8: predNextWorkload ← $getNextRoof$(predNextWorkload, featureSet)
9: **end if**
10: newReplicas ← predNextWorkload / replicaCapacity

11: **function** GETNEXTROOF(elt, $\{el_1, el_2, ...el_n\}$)
12: $subSet \leftarrow$ inferior(elt, $\{el_1, el_2, ...el_n\}$)
13: roof ← min(subSet)
14: **return** roof
15: **end function**

8. Experiments and results

In this section, the main objective of the evaluation is to present the feasibility and utility of our two contributions, namely time-series featurization, which improves the accuracy of the forecasting model, and our oscillation mitigation approach, which improves the auto-scaling efficiency.

To achieve this objective, each of the following subsections presents an important aspect of the evaluation.

8.1. Simulator

To examine our approach, we developed a simulation program in Python (using NumPy, Pandas, Scikit-learn, and Keras libraries) to test and compare our results with related work (Imdoukh et al., 2019; Dang-Quang and Yoo, 2021). This simulator enables the auto-scaling of replicas according to the prediction data generated by the machine learning model presented in Section 6. As shown in Fig. 9, the simulation tool consists of five phases, following the MAPE-K loop, including monitoring the auto-scaling and evaluating the results to verify the impact of the models on the auto-scaling process.

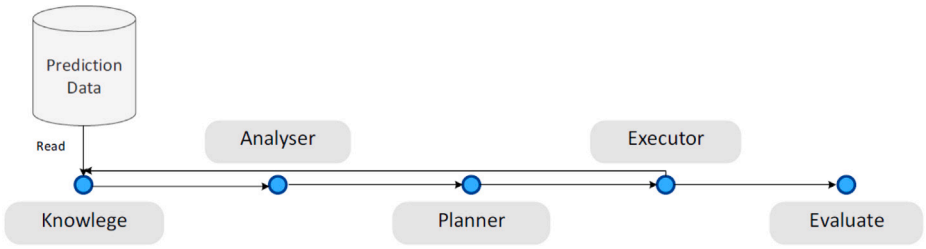


Fig. 9. Architecture of the simulator.

8.2. Performance metrics

Two types of metrics are considered: forecasting model metrics and auto-scaling metrics.

8.2.1. Evaluation metrics for the prediction model

The regression models selected in this study, LSTM and Bi-LSTM, allow the prediction of future sequences of a dataset according to the parameters acquired during its training. The efficiency of these forecasting models is evaluated according to their generalization error rate (i.e., accuracy). The accuracy evaluation in regression analysis consists of comparing the original target with the predicted one. For these measures, we can use different metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2), which help to explain the errors and predictive ability of the model (Chicco et al., 2021). These measures are defined as follows:

- MAE (Mean Absolute Error) represents the difference between the actual and predicted values, calculated by averaging the absolute errors over the dataset (Eq. (15)).
- MSE (Mean Squared Error) represents the average squared difference between the actual and predicted values (Eq. (16)).
- RMSE (Root Mean Squared Error) represents the standard deviation of the prediction by the square root of MSE (Eq. (17)).
- Coefficient of determination (R^2) represents the proportion of the variance in the dependent variable that is predictable from the independent variables in the regression model. The value of R^2, which usually ranges from 0 to 1, indicates the strength of the correlation between the actual and predicted values, with higher values indicating a better model fit (Eq. (18)).

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{15}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{16}$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{17}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{N}\left(y_i - \bar{y}\right)^2} \tag{18}$$

where y, $\hat{y}_i$ and $\bar{y}_i$ represent actual value, predicted value, and mean of actual values y, respectively.

As shown by equations, MSE and MAE are error metrics that quantify the difference between predicted and actual values. A lower value for these metrics indicates better performance, meaning the model's predictions are closer to the actual values. The RMSE, being the square root of MSE, penalizes more significant errors further, giving a better representation of the overall prediction performance. Thus, a model having a lower MSE value implies that it has a lower value of RMSE. Additionally, we included the R-squared (R2) metric to assess the proportion of variance in the predicted values, obtained by forecasting models, compared to actual values. It represents an informative metric of the model's goodness-of-fit, where a higher value indicates a better fit.

8.2.2. Auto-scaler evaluation metrics

The performance evaluation of our auto-scaling simulator is based on two essential aspects: elasticity and provisioning rate. For this purpose, we have used the metrics proposed in Herbst et al. (2016) and Bauer et al. (2018a). These metrics have been used in several literature studies on auto-scaling, such as Imdoukh et al. (2019), Dang-Quang and Yoo (2021) and Bauer et al. (2018b):

- Under-provisioning metric (θ_u) indicates the number of missing replicas (e.g., containers) needed to reach the requested number of replicas in a time interval (Eq. (19)).
- Over-provisioning metric (θ_o) represents the supplied replicas that exceed the demanded number, as shown in Eq. (20).
- Under-provisioning time (T_u) reflects the time during which the simulator was under-provisioning (Eq. (21))
- Over-provisioning time (T_o) reflects the time during which the simulator was over-provisioning (Eq. (22))
- Elasticity speedup (ϵ_n) reveals the performance gain obtained by using a proactive auto-scaler. In this work, the elasticity speedup is calculated by a ratio between two cases: using a proactive auto-scaler and a reactive auto-scaler, which are represented in Eq. (23), by the p and r indices, respectively. In contrast to the previous metrics, the higher the ϵ_n value, the higher the auto-scaler performance. In other words, the best auto-scaler has lower $\theta_u, \theta_o, T_u, T_o$ values and essentially a higher ϵ_n value.

$$\theta_u = \frac{100}{T}\sum_{t=1}^{T}\frac{max(required(t) - provided(t), 0)}{required(t)}\Delta t \tag{19}$$

$$\theta_o = \frac{100}{T}\sum_{t=1}^{T}\frac{max(provided(t) - required(t), 0)}{required(t)}\Delta t \tag{20}$$

$$T_u = \frac{100}{T}\sum_{t=1}^{T}\max(sgn(required(t) - provided(t)), 0)\Delta t \tag{21}$$

$$T_o = \frac{100}{T}\sum_{t=1}^{T}\max(sgn(provided(t) - required(t)), 0)\Delta t \tag{22}$$

$$\epsilon_n = \left(\frac{\theta_{u,r}}{\theta_{u,p}}.\frac{\theta_{o,r}}{\theta_{o,p}}.\frac{T_{u,r}}{T_{u,p}}.\frac{T_{o,r}}{T_{o,p}}\right)^{\frac{1}{4}} \tag{23}$$

where, $required(t)$ represents the correct number of replicas corresponding to the actual workload, and $provided(t)$ represents the number of replicas offered by the auto-scaler according to the predicted value of the workload. Δt corresponds to the time interval (e.g., each 1 min) used to check the change in workload, T represents the entire evaluation period, and $sgn()$ is the sign function.

8.3. Datasets

To validate our findings, we selected the following two publicly available datasets.

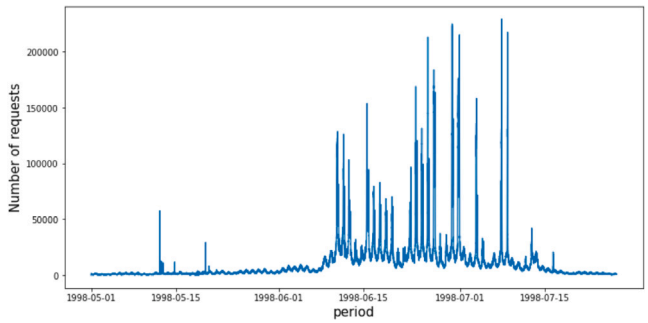


Fig. 10. Representation of Worldcup 98' dataset by number of requests per minute.

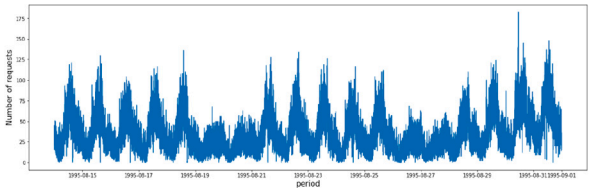


Fig. 11. Representation of NASA 95' dataset by number of requests per minute.

8.3.1. Worldcup'98 dataset

The Worldcup'98 dataset contains the HTTP request logs of more than 1.3 billion requests from the 1998 FIFA World Cup website in France between April 30 and July 26 (Arlitt and Jin, 2000). This high request load is due to the number of spectators worldwide who followed this event. The Worldcup 98' Dataset is often used by researchers working on auto-scaling in the cloud. Moreover, this will allow us to compare our results with other studies such as Imdoukh et al. (2019) and Dang-Quang and Yoo (2021).

The data structure of this dataset contains the following properties: timestamp, clientID, objectID, size, method, status, type, and server (Arlitt and Jin, 2000). To better manage this data, we performed a preprocessing by grouping all logs occurring in the same minute into a single cumulative record, as in Imdoukh et al. (2019). As a result, the information about the number of requests corresponds to the number of received requests in one minute. Fig. 10 plots the dataset obtained.

8.3.2. NASA 95' dataset

The NASA 95' Dataset provides a two-month log of HTTP requests to the NASA Kennedy Space Center Web server in Florida. This dataset contains 3,461,612 requests collected between July 1, 1995, at 00:00:00 and August 31, 1995, at 23:59:59. The timestamps are accurate to one second. It should be noted that no accesses were reported from 01/Aug/1995:14:52:01 to 03/Aug/1995:04:36:13 since the web server was shut down due to Hurricane Erin Dang-Quang and Yoo (2021). Fig. 11 plots a part of the dataset.

8.3.3. Data analysis

In our analysis of the WorldCup and NASA datasets, specifically focusing on the first 80k data points, we observed distinct statistical behaviors in each dataset. For the NASA dataset, as depicted in Fig. 12, there are initial fluctuations in mean and variance values. However, these metrics stabilize after the first 40,000 data points, suggesting a transition to a more consistent workload pattern over time. This observation was substantiated by the Augmented Dickey–Fuller (ADF) test, which indicated stationarity with a near-zero p-value.

In contrast, the WorldCup dataset, illustrated in Fig. 13, demonstrates a different pattern, with both mean and variance increasing over time. This behavior is indicative of a non-stationary time series, as further evidenced by its ADF test result, which yielded a p-value of

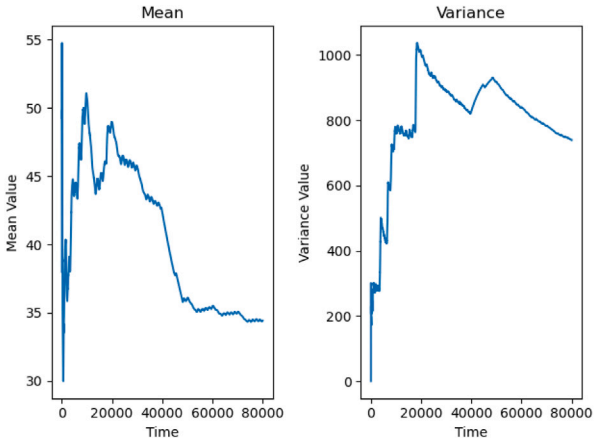


Fig. 12. Data analysis of NASA dataset.

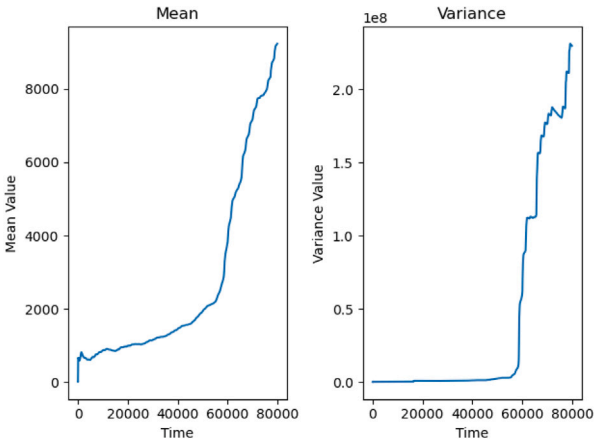


Fig. 13. Data analysis of WorldCup dataset.

0.056638. This value suggests non-stationarity, potentially complicating the modeling process due to the influence of trend and seasonality components.

In light of these findings, some forecasting models like ARIMA necessitate transforming the time series data to achieve stationarity before modeling. In our work, our approach leverages an LSTM model that does not necessitate this transformation thanks to its ability to capture complex patterns in the data effectively.

Additionally, the distribution of the NASA'95 dataset is less complicated than that of Worldcup 98' (Dang-Quang and Yoo, 2021). Fig. 14 showcases the data distributions for both datasets: Worldcup and NASA. The Worldcup dataset is characterized by a higher degree of variability and outliers, as evidenced by the broader spread of data points. This is particularly highlighted by the stark difference between the median value of 3884 and the maximum value, which surpasses 200,000, indicating the presence of extreme values in the Worldcup dataset. In contrast, the NASA dataset demonstrates less variability, as shown by its maximum value of around 300 and a median value of 29, indicating a more concentrated distribution of data points.

The Worldcup'98 dataset, therefore, presents a significant challenge in terms of prediction due to its high variability and complex patterns, featuring unpredictable peaks. This contrasts with the NASA'95 dataset, which exhibits a more stable and predictable pattern.

8.4. Evaluation protocol

First, we have rebuilt the forecasting models proposed in the literature to reproduce these tests in the same operating environment

Table 2
Our featurization-based forecasting approach vs. related work.

Dataset	Approach	RMSE	R^2	MSE	MAE
NASA	LSTM (Imdoukh et al., 2019)	0.0811	0.6430	0.0066	0.0607
	Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0813	0.6409	0.0066	0.0609
	Our model	**0.0016**	**0.9993**	**2.6343e−06**	**0.0015**
Worldcup'98	LSTM (Imdoukh et al., 2019)	0.0023	0.9862	5.1755e−06	0.0016
	Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0020	0.9898	3.8043e−06	0.0013
	Our model	**0.0006**	**0.9990**	**3.5793e−07**	**0.0004**

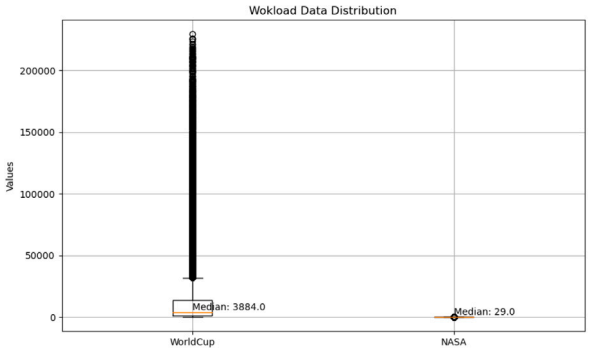


Fig. 14. Data distribution comparison of Worldcup and NASA datasets.

and execution conditions. This approach allows for reproducible and non-biased results. Consequently, we retained the same configuration of hyperparameters proposed in each approach.

Second, the efficiency evaluation of each forecasting algorithm will be based on its performance results. However, the configuration parameters proposed for these algorithms (Imdoukh et al. algorithm in Imdoukh et al. (2019), Dang et al. algorithm in Dang-Quang and Yoo (2021) and our algorithm in Table 1) are not sufficient to cover all our test cases, whether during the prediction stage or the auto-scaling of the system.

Therefore, we needed to test several combinations of the key hyperparameters of these models to verify their impact on the contributions of different approaches by analyzing the change in forecasting results. Additionally, based on multiple combinations of settings and parameters, this study identifies the best model with the optimal configuration.

Third, each dataset selected, namely Worldcup '98 and NASA '95, is partitioned into two subsets: one for training the model and the other for testing purposes. Inspired by the principle "Pareto (80–20)" (Dunford et al., 2014), we allocated 80% of the data for training and the remaining 20% for testing, applying this division to each dataset. The training data is scaled independently of the testing data to prevent potential bias. The results obtained from executing the model on the test data were meticulously analyzed using various performance metrics to assess the accuracy and reliability of the forecasts.

Finally, we experimented with our predicted data as well as those of Imdoukh et al. (2019) and Dang-Quang and Yoo (2021) in our simulator, to choose the best auto-scaling approach, which deals most effectively with oscillation mitigation. These approaches are evaluated according to the metrics presented in Section 8.2.

8.5. Evaluation of our forecasting approach

Initially, we aim to reproduce the forecasting approaches proposed in the literature and test them under similar operating conditions. Specifically, we consider for comparison the two approaches (Imdoukh et al., 2019; Dang-Quang and Yoo, 2021) as they both utilize multilayer learning techniques and address the oscillation issue. Imdoukh et al. (2019) employs an LSTM model, while Dang-Quang and Yoo (2021) utilize a Bi-LSTM model.

Next, we aim to ensure that our featurization-based approach can produce accurate forecasts compared to these existing works. Then, we also aim to assess the impact of featurization on forecasting results.

8.5.1. Comparison to related work

We employed the metrics introduced in Section 8.2, namely RMSE, MSE, MAE, and R^2, to evaluate the regression model. Table 2 presents the evaluation metrics compared to the related work.

During these tests, we noticed that the results of the two related work approaches were consistently close, with occasional variations in performance where one approach outperformed the other. Nevertheless, our approach with the featurization always performed well in these tests and outperformed other approaches.

In the case of the NASA dataset, the improvement rendered by our approach is particularly noteworthy. As explained in Subsubsection Section 8.3.3, the NASA dataset exhibits a more stable pattern, which can positively impact the forecasting error. In contrast, the Worldcup'98 dataset contains numerous workload peaks, leading to more outliers in the differences between predicted and actual data. These substantial and frequent discrepancies obscure the overall enhancement observed in evaluation metrics such as RMSE and MAE.

Our model displayed remarkable performance for the NASA dataset, surpassing the best values reported in the related works. Notably, our model demonstrated approximately relative improvements of 98%, 55%, 99%, and 97% in RMSE, R2, MSE, and MAE, respectively.

This reduction in the error rate has, therefore, impacted the quality and accuracy of the forecasting as presented in Figs. 15(a) and 15(b), which visualize the prediction of our model on Worldcup'98 and NASA dataset respectively. The red color represents the actual dataset, whereas the prediction of our model is plotted using the blue color. It is clear that our model fits well with the test dataset.

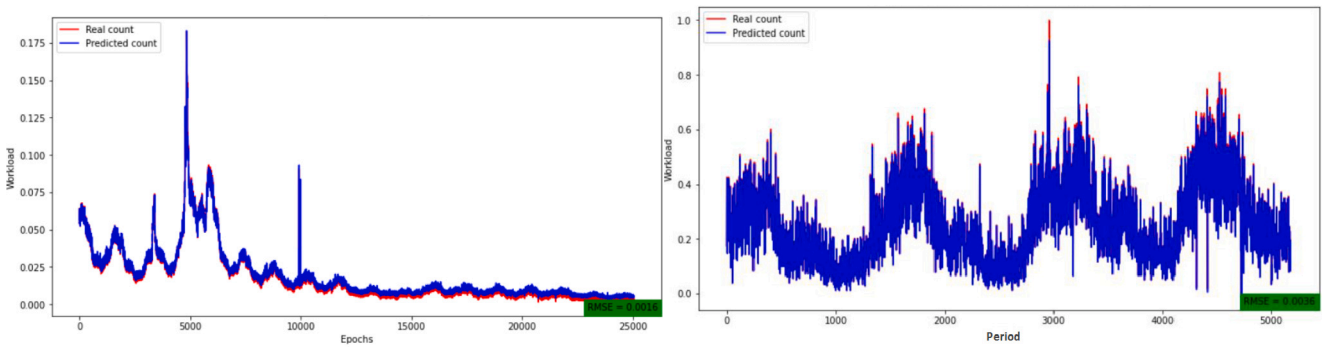
8.5.2. Result of the hyperparameter combination tests

To further evaluate the effect of the featurization mechanism, we decided to change the LSTM hyperparameters of the different approaches to see if that impacts our findings. We have chosen the following hyperparameters: the number of units and epochs. The number of units corresponds to the dimension of the hidden cells. The number of epochs defines the number of times the learning algorithm will change the network's weights.

Table 3 shows that our approach, regardless of the changed hyperparameters, outperforms related work thanks to our featurization mechanism. As in the previous test category, the improvement with the NASA dataset is more apparent and significant.

8.5.3. Evaluation of the impact of our featurization mechanism on the related work approaches

In the previous testing phase, we sought to determine the influence of our featurization mechanism by adjusting two key hyperparameters: the number of units and the number of epochs. In this test category, we consider the overall hyperparameters of related work models by applying our featurization mechanism to the data before using the forecasting algorithms of the related work (i.e., Imdoukh et al. (2019), Dang-Quang and Yoo (2021)). Table 4 showcases that our featurization mechanism enhances the forecasting accuracy of the related work,



(a) Our model prediction with World Cup 98 dataset (b) Our model prediction with NASA dataset

Fig. 15. Our model forecasting.

Table 3
Forecasting results with varying the hyper-parameters: the numbers of units and epochs.

Case	Dataset	Approach	RMSE	R^2	MSE	MAE
30 units with 50 epochs	NASA	LSTM (Imdoukh et al., 2019)	0.0811	0.6430	0.0066	0.0607
		Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0813	0.6409	0.0066	0.0609
		Our model	**0.0024**	**0.9997**	**6.1537−06**	**0.0017**
	Worldcup'98	LSTM (Imdoukh et al., 2019)	0.0023	0.9861	5.1755e−06	0.0016
		Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0020	0.9898	3.8042e−06	0.0013
		Our model	**0.0006**	**0.9990**	**3.5793e−07**	**0.0004**
20 units with 120 epochs	NASA	LSTM (Imdoukh et al., 2019)	0.0807	0.6468	0.0065	0.0606
		Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0817	0.6381	0.0067	0.0610
		Our model	**0.0044**	**0.9989**	**1.9779e−05**	**0.0027**
	Worldcup'98	LSTM (Imdoukh et al., 2019)	0.0038	0.9609	1.4625e−05	0.0035
		Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0021	0.9884	4.3315e−06	0.0017
		Our model	**0.0017**	**0.9916**	**3.1499e−06**	**0.0015**
10 units with 50 epochs	NASA	LSTM (Imdoukh et al., 2019)	0.0816	0.6389	0.0067	0.0614
		Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0823	0.6326	0.0068	0.0614
		Our model	**0.0073**	**0.9971**	**5.2620e−05**	**0.0051**
	Worldcup'98	LSTM (Imdoukh et al., 2019)	0.0044	0.9485	1.9246e−05	0.0040
		Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0023	0.9858	5.3124e−06	0.0018
		Our model	**0.0019**	**0.9907**	**3.4721e−06**	**0.0017**

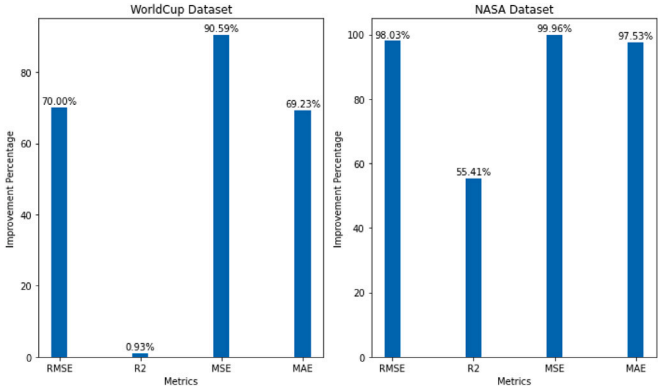


Fig. 16. Relative improvement percentage of metrics: Our approach vs. Related works.

especially evident in the case of the NASA dataset. For example, our approach improves approximately 92% in $RMSE$ and 55% in R^2 for the LSTM model. Additionally, it significantly improves the performance of the Bi-LSTM model with approximately 84% improvement in $RMSE$ and 55% improvement in R^2.

8.5.4. Finding and analysis summary of our featurization approach evaluation

Our featurization approach has exhibited substantial improvements in forecasting accuracy, as substantiated through a comprehensive evaluation encompassing comparisons with related works, testing models with varied hyperparameters, and assessing the impact of our featurization on existing forecasting works.

Fig. 16 visually represents the percentage improvement achieved by comparing our forecasting model relative to the best values of compared related works, which are presented in Table 2.

In both datasets, our approach consistently surpassed other models in accuracy metrics such as RMSE, MSE, and MAE. The relative improvements ranged approximately between 70% and 99%, highlighting the efficacy of our featurization in minimizing forecast errors.

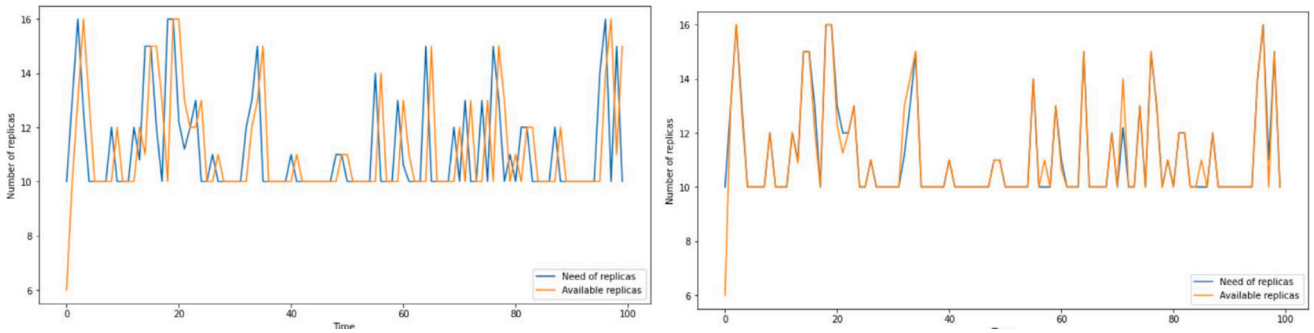
For the WorldCup dataset, the R^2 value of our model was comparable to other models that already exhibited high R^2 values (around 0.99). This similarity indicates that our model captures levels of variance in the data similar to that of the other models while still achieving superior performance in error metrics.

It is important to note that the different metric values are computed based on scaled data. Upon descaling the data (i.e., inverting the scaling process), the metric values increase while maintaining the same relative improvement percentage.

In addition, our work uses an LSTM model that predicts the future workload in order to handle the complexity, the non-stationarity, and the possible uncertainties of the workload data through its ability to learn long-term dependencies and patterns in the data. The LSTM model addresses the non-stationary nature, which has varying means and variances, by learning from sequences of data, effectively capturing trends and seasonal effects, which are common in non-stationary data. Our featurization approach, which automatically extracts features describing the input data, aims to aid the LSTM model in handling

Table 4
The impact of our featurization mechanism on the related work approaches.

Dataset	Approach	RMSE	R^2	MSE	MAE
NASA	LSTM (Imdoukh et al., 2019)	0.0811	0.6430	0.0066	0.0607
	LSTM (Imdoukh et al., 2019) ⊕ our featurization	**0.0056**	**0.9983**	**3.1902e−05**	**0.0043**
	Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0813	0.6409	0.0066	0.0609
	Bi-LSTM (Dang-Quang and Yoo, 2021) ⊕ our featurization	**0.0130**	**0.9908**	**0.0002**	**0.0099**
Worldcup'98	LSTM (Imdoukh et al., 2019)	0.0022	0.9861	5.1755	0.0016
	LSTM (Imdoukh et al., 2019) ⊕ our featurization	**0.0016**	**0.9935**	**2.4442e−06**	**0.0010**
	Bi-LSTM (Dang-Quang and Yoo, 2021)	0.0019	0.9898	3.8043e−06	0.0013
	Bi-LSTM (Dang-Quang and Yoo, 2021) ⊕ our featurization	**0.0052**	**0.9289**	**2.6619e−05**	**0.0051**



(a) Based on a reactive approach (b) Based on our proactive approach

Fig. 17. Auto-scaler behavior using NASA'95 dataset.

uncertainties inherent in time-series data, such as noise and workload fluctuations in the NASA and WorldCup datasets. The data variability (fluctuations) can be well handled if it presents a pattern available or similar to one in the training dataset. While entirely new patterns caused by external uncertainties (e.g., sudden workload spikes due to external events) that were not present in the training data require retraining or updating the model with new data that includes these patterns.

8.6. Evaluation of the auto-scaler

This subsection primarily evaluates our oscillation mitigation approach by assessing the auto-scaler performance. Therefore, we will discuss our strategies and the results of the tests conducted using the auto-scaling simulator. We have implemented our auto-scaling simulator in two modes: the reactive and the proactive. The reactive mode, common in most commercial solutions, operates without utilizing prediction functionality. The reactive approach serves as a baseline to compare the enhancement offered by the proactive approaches and plays a role in calculating auto-scaling performance metrics, notably the ϵ_n metric (Elasticity speedup in Eq. (23)).

The proactive mode facilitates the analysis of the performance of our approach and its comparison with related works, such as those by Imdoukh et al. (2019) and Dang-Quang and Yoo (2021).

To allow a fair comparison, we ran our auto-scaling simulator under the same conditions proposed in existing literature, using predicted data collected from previous tests.

In these tests, we considered five auto-scaling approaches:

- The reactive approach: Auto-scaling is based on current monitoring data, lacking forecasting capability. This simplistic category represents most industrial auto-scaling solutions.
- Our proactive approach ($OurProactive$): This represents our proactive approach that uses the data generated from our featurization-based forecasting algorithm without processing the oscillation issue.
- Our Proactive & Grid approach ($OurProWithGrid$): This is our proactive approach combined with our grid-based oscillation mitigation approach.
- Work1 : An approach proposed in literature (Imdoukh et al., 2019), applying our predicted data generated by our featurization-based approach. It suggested a cooldown timer (CDT) of 10 s and a scaling down ratio (SDR) of 40%.
- Work2 : the approach proposed in the literature (Dang-Quang and Yoo, 2021) employing our predicted data. It selects 60-second CDT and 60% SDR.

8.6.1. Reactive vs. our proactive approach

In this test category, we demonstrate the enhancements brought by our proactive approach, based on the featurization forecasting mechanism.

Comparing two Figs. 17(a) and 17(b), we observe that our proactive approach shortens response times due to its predictive capabilities regarding system resource requirement changes. Moreover, our forecasting approach has notably improved test result metrics values across both datasets (NASA and WorldCup), as presented in Table 5.

Thanks to the forecasting mechanism, our proactive approach reduces the variation between the current and the needed resources (i.e., replicas), thus improving auto-scaling performance. For instance, in the case of Worldcup dataset, our model improves the metric values of over-provisioning (θ_o), under-provisioning (θ_u), over-provisioning time (T_o), under-provisioning time (T_u) and elasticity speedup (ϵ_n) by about 55%, 76%, 55%, 33%, and 140%, respectively. Also, for the NASA dataset, our proactive approach enhances the overall auto-scaling performance (i.e., ϵ_n) by about 206%. This significant improvement demonstrates the importance of the proactive behavior of our approach, which uses our featurization-based forecasting approach.

8.6.2. Our approach with vs. without oscillation mitigation

To evaluate the impact of our oscillation mitigation approach on the auto-scaling behavior, we compare our proactive approach with and without the oscillation mitigation. Table 6 summarizes the test results. Our oscillation mitigation approach was able to improve the performance of the auto-scaler enormously. It improved the overall Elasticity Speedup metric (ϵ_n) by about 400% with the NASA'95 dataset and 108% with the WorldCup'98 dataset.

Table 5
Reactive vs. our Proactive auto-scaling approaches with Worldcup 98' and NASA Datasets.

Dataset	Metric	Reactive	Our proactive
NASA	θ_o	2.8139	**0.0493**
	θ_u	**1.1794**	1.1993
	T_o	13.3070	**9.5947**
	T_u	13.2231	**11.6587**
	ϵ_n	1	**3.0659**
Worldcup'98	θ_o	10.1007	**4.4949**
	θ_u	4.6103	**1.0628**
	T_o	20.2554	**9.0138**
	T_u	20.4295	**13.5262**
	ϵ_n	1	**2.3983**

Table 6
Our proactive approach with vs. without oscillation mitigation with Worldcup 98' and NASA Datasets.

Dataset	Metric	OurProactive	OurProWithGrid
NASA	θ_o	**0.0493**	0.0722
	θ_u	1.1993	**0.2543**
	T_o	9.5947	**0.2916**
	T_u	11.6587	**2.0095**
	ϵ_n	3.0659	**15.2634**
Worldcup'98	θ_o	4.4949	**0.6567**
	θ_u	1.0628	**1.0508**
	T_o	9.0138	**4.5799**
	T_u	13.5262	**9.8195**
	ϵ_n	2.3983	**4.9919**

Our grid-based mechanism can increase the over-provisioning metric, θ_o, as it can choose a higher value than predicted in the case of downscaling needs. For example, in the case of the NASA dataset, the θ_o metric is increased by about 46%. Moreover, against expectation, our approach has reduced the time when the system is over-provisioning (T_o metric). The dynamics of our grid and the predicted workload explain this improvement. As a result, the overall performance of auto-scaling is greatly improved.

8.6.3. Comparison of our oscillation mitigation approach with related work

This test category aims to compare our oscillation mitigation mechanism with the related work (Imdoukh et al., 2019; Dang-Quang and Yoo, 2021) that both proposed oscillation mitigation strategies by restricting the periodicity and rate of change. Work1 (Imdoukh et al., 2019) used a cooldown timer (CDT) of 10 s with a scaling down ratio (SDR) of 40 percent, while Work2 (Dang-Quang and Yoo, 2021) selects 60 s for the CDT and 60 percent for the SDR. To neutralize the forecasting effects of each approach, we utilized the same forecasting data generated by our forecasting approach.

Interestingly, unlike other related work mechanisms, our oscillation mitigation approach introduces a unique mechanism that is agnostic to any parameters that need optimization (i.e., parameterless). Considering this originality, our approach slightly improved the overall auto-scaling performance compared to established and widely used mechanisms, as presented in Table 7.

These findings led us to explore the combination of these approaches for potential improvements, discussed in the subsequent evaluation section.

8.6.4. Combination of oscillation solutions

This test category investigates the feasibility and effectiveness of combining our grid-based oscillation mitigation approach with the cooldown time (CDT), commonly used in related work. The CDT introduces a delay when a down scaling request is received to ensure the persistence of the need.

Table 7
Our proactive approach with our grid-based oscillation mitigation vs. related work.

Dataset	Metric	OurProWithGrid	Work1 (Imdoukh et al., 2019)	Work2 (Dang-Quang and Yoo, 2021)
NASA	θ_o	**0.0722**	0.0828	0.0732
	θ_u	0.2543	0.2565	**0.2501**
	T_o	**0.2916**	0.2954	0.2954
	T_u	2.0095	2.0202	**2.0082**
	ϵ_n	**15.2625**	14.6498	15.2237
Worldcup'98	θ_o	**0.6567**	0.9125	0.8357
	θ_u	1.0508	1.3457	**0.9761**
	T_o	**4.5799**	4.3529	4.6044
	T_u	**9.8195**	10.0019	9.8665
	ϵ_n	**4.9919**	4.3572	4.7753

Table 8
Using our predicted data, a comparison of our grid-based approach ($OurProWithGrid$) to its combination with CDT mechanism ($OurProWithGrid \oplus CDT$).

Case	Metric	OurProWithGrid	OurProWithGrid ⊕ CDT
Case1: CDT = 10 s	θ_o	**0.0722**	0.4526
	θ_u	0.2543	**0.0366**
	T_o	**0.2916**	2.0167
	T_u	2.0095	**0.5605**
	ϵ_n	**15.2625**	13.2899
Case1: CDT = 60 s	θ_o	**0.0722**	0.1539
	θ_u	0.2543	**0.0061**
	T_o	**0.2916**	0.4268
	T_u	2.0095	**0.0694**
	ϵ_n	15.2625	**67.6469**

Table 9
Using the predicted data of Imdoukh et al. (2019) approach, a comparison of our grid-based approach ($OurProWithGrid$) to its combination with CDT mechanism ($OurProWithGrid \oplus CDT$).

Case	Metric	OurProWithGrid	OurProWithGrid ⊕ CDT
Case1: CDT = 10 s	θ_o	0.0722	**0.0336**
	θ_u	**0.2543**	0.5066
	T_o	0.2916	**0.1973**
	T_u	**2.0095**	7.0110
	ϵ_n	**15.2625**	12.5497
Case1: CDT = 60 s	θ_o	0.0722	**0.0119**
	θ_u	0.2543	**0.1280**
	T_o	0.2916	**0.0480**
	T_u	2.0095	**2.2090**
	ϵ_n	15.2625	**43.6019**

Table 10
Using the predicted data of Dang-Quang and Yoo (2021) approach, a comparison of our grid-based approach ($OurProWithGrid$) to its combination with CDT mechanism ($OurProWithGrid \oplus CDT$).

Case	Metric	OurProWithGrid	OurProWithGrid ⊕ CDT
Case1: CDT = 10 s	θ_o	0.0722	**0.0337**
	θ_u	**0.2543**	0.4981
	T_o	0.2916	**0.1973**
	T_u	**2.0095**	6.9737
	ϵ_n	**15.2625**	12.6064
Case1: CDT = 60 s	θ_o	0.0722	**0.0119**
	θ_u	0.2543	**0.1261**
	T_o	0.2916	**0.0480**
	T_u	**2.0095**	2.1983
	ϵ_n	15.2625	**43.8206**

To deduce general insights into the performance of the combination of mechanisms, we conducted extensive experiments considering different contextual parameters, such as the predicted data and the CDI values, as presented in Tables 8–10.

Two CDT values were considered: 10 ms and 60 ms. In the case of CDT = 10 ms, the combination is less efficient due to a small value of CDT, leading to more auto-scaling operations exacerbating the oscillation issue.

In contrast, the combination with the CDT = 60 case significantly improved the performance of the auto-scaling. This improvement was reflected in the overall performance metric (ϵ_n), which increased by approximately 343%, 185%, and 187% considering the predicted data used in Tables 8–10, respectively.

In addition, this fairly high value of CDT (i.e., 60 ms) can increase the over-provisioning metrics, T_o and θ_o, as the case in Table 8. Since the CDT mechanism adds a delay time, our approach reduces the number of resources to shrink. However, despite this possible increase in over-provisioning metrics, the combination significantly improves the under-provisioning metrics, which is reflected in the overall performance of the auto-scaling represented by Elasticity speedup (ϵ_n).

8.6.5. Finding and analysis summary of our oscillation mitigation approach evaluation

Initially, our proactive approach, leveraging our featurization technique for improved workload forecasting, significantly enhanced the auto-scaling performance compared to the reactive approach. The proactive nature allows for better handling of workload fluctuations, thereby reducing the likelihood of over or under-provisioning resources.

Furthermore, the performance improvement is further amplified by combining our featurization approach with our proposed oscillation mitigation mechanism, referred to as OurProWithGrid. The overall auto-scaling performance of our approach surpasses that of the related works for both datasets. Notably, our grid-based oscillation mitigation approach offers the advantage of being parameterless, eliminating the need for tuning, in contrast to the common mechanisms employed in related works.

Finally, we explored combining our grid-based oscillation mitigation approach with established mechanisms, particularly the Cool Down Timer (CDT). With an optimized CDT value, this combination leads to enhanced auto-scaling performance compared to using only the CDT mechanism. However, it requires the CDT parameter, making it not parameterless. Therefore, we plan to explore the utilization of optimization techniques to determine the optimal CDT value as part of our future work, as discussed in the conclusion section.

9. Conclusion

In this work, we addressed the challenge of resource management in IoT systems, focusing on service auto-scaling in the edge computing environment. Given the capabilities of edge devices, optimizing their resource management is essential for enhancing service performance and ensuring efficient resource utilization.

Our contributions are twofold. Initially, we employed the LSTM machine learning model to predict upcoming resource change requests, enabling a proactive adaptation to fluctuating resource needs. We further enhanced our LSTM prediction model through a data featurization technique inspired by the Japanese Candlestick concept. This enhancement allowed for more accurate extraction of correlations within the input data, significantly improving prediction accuracy compared to existing literature.

Subsequently, we addressed the challenge of oscillation mitigation inherent to auto-scaling. We introduced a novel grid concept in the action generation phase, which effectively leverages historical workload features to select the number of replicas efficiently during the planning phase. In contrast to related works, our approach is parameterless, offering the advantage of eliminating the need for parameter optimization and demonstrating improved performance.

Combining these mechanisms considerably enhances the performance of service auto-scaling, representing a significant advance over the existing literature. The versatility of our solution also suggests potential applicability across various domains (e.g., networks), fostering improved autonomous and self-adaptive proactive resource management.

In terms of future work, the promising results from our featurization approach inspire further exploration into utilizing candlestick patterns within our feature grid to refine our forecasting model. The growing field of graph theory-based parametric machine learning algorithms (Tutsoy, 2023) presents an attractive prospect. We intend to incorporate the derived features into multi-dimensional data modeled as a graph, aiming to enhance the understanding and modeling of complex relationships, which might improve further prediction accuracy. Furthermore, we plan to apply optimization techniques to enhance the combination of our grid-based oscillation mitigation mechanism with the commonly used CDT mechanism, as our experiments suggest that such a combination holds substantial potential for performance improvement.

Declaration of competing interest

All authors confirm that:

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

Acknowledgment

This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Ahmed, B., Seghir, B., Al-Osta, M., Abdelouahed, G., 2019. Container based resource management for data processing on IoT gateways. Procedia Comput. Sci. 155, 234–241.

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P., 2017. Elasticity in cloud computing: state of the art and research challenges. IEEE Trans. Serv. Comput. 11 (2), 430–447.

Arcaini, P., Riccobene, E., Scandurra, P., 2015. Modeling and analyzing MAPE-K feedback loops for self-adaptation. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE, pp. 13–23.

Arlitt, M., Jin, T., 2000. A workload characterization study of the 1998 world cup web site. IEEE Netw. 14 (3), 30–37.

Bali, A., Al-Osta, M., Ben Dahsen, S., Gherbi, A., 2020. Rule based auto-scalability of IoT services for efficient edge device resource utilization. J. Ambient Intell. Humaniz. Comput. 1–18.

Bauer, A., Grohmann, J., Herbst, N., Kounev, S., 2018a. On the value of service demand estimation for auto-scaling. In: International Conference on Measurement, Modelling and Evaluation of Computing Systems. Springer, pp. 142–156.

Bauer, A., Herbst, N., Spinner, S., Ali-Eldin, A., Kounev, S., 2018b. Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field. IEEE Trans. Parallel Distrib. Syst. 30 (4), 800–813.

Calheiros, R.N., Masoumi, E., Ranjan, R., Buyya, R., 2014. Workload prediction using ARIMA model and its impact on cloud applications' QoS. IEEE Trans. Cloud Comput. 3 (4), 449–458.

Cetinski, K., Juric, M.B., 2015. AME-WPC: Advanced model for efficient workload prediction in the cloud. J. Netw. Comput. Appl. 55, 191–201.

Chernikov, A., Tan, C.W., Montero-Manso, P., Bergmeir, C., 2022. FRANS: Automatic feature extraction for time series forecasting. arXiv preprint arXiv:2209.07018.

Chicco, D., Warrens, M.J., Jurman, G., 2021. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. PeerJ Comput. Sci. 7, e623.

Ciptaningtyas, H.T., Santoso, B.J., Razi, M.F., 2017. Resource elasticity controller for Docker-based web applications. In: 2017 11th International Conference on Information & Communication Technology and System. ICTS, IEEE, pp. 193–196.

Computing, A., et al., 2006. An Architectural Blueprint for Autonomic Computing. IBM White Paper 31, pp. 1–6.

Dang-Quang, N.-M., Yoo, M., 2021. Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. Appl. Sci. 11 (9), 3835.

Di, S., Kondo, D., Cirne, W., 2012. Host load prediction in a Google compute cloud with a Bayesian model. In: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, pp. 1–11.

Doan, D.N., Zaharie, D., Petcu, D., 2019. Auto-scaling for a streaming architecture with fuzzy deep reinforcement learning. In: European Conference on Parallel Processing. Springer, pp. 476–488.

Dogani, J., Khunjush, F., Mahmoudi, M.R., Seydali, M., 2023. Multivariate workload and resource prediction in cloud computing using CNN and GRU by attention mechanism. J. Supercomput. 79 (3), 3437–3470.

Dunford, R., Su, Q., Tamang, E., 2014. The pareto principle.

Evans, D., 2011. The Internet of Things: How the Next Evolution of the Internet is Changing Everything. CISCO white paper 1, pp. 1–11.

Goli, A., Mahmoudi, N., Khazaei, H., Ardakanian, O., 2021. A holistic machine learning-based autoscaling approach for microservice applications. In: CLOSER. pp. 190–198.

Herbst, N., Krebs, R., Oikonomou, G., Kousiouris, G., Evangelinou, A., Iosup, A., Kounev, S., 2016. Ready for rain? A view from SPEC research on the future of cloud metrics. arXiv preprint arXiv:1604.03470.

Hu, Y., Huber, A., Anumula, J., Liu, S.-C., 2018. Overcoming the vanishing gradient problem in plain recurrent networks. arXiv preprint arXiv:1801.06105.

Hyndman, R.J., Wang, E., Laptev, N., 2015. Large-scale unusual time series detection. In: 2015 IEEE International Conference on Data Mining Workshop. ICDMW, IEEE, pp. 1616–1619.

Imdoukh, M., Ahmad, I., Alfailakawi, M.G., 2019. Machine learning-based auto-scaling for containerized applications. Neural Comput. Appl. 1–16.

Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J., Tilkov, S., 2018. Microservices: The journey so far and challenges ahead. IEEE Softw. 35 (3), 24–35.

Kan, C., 2016. DoCloud: An elastic cloud platform for Web applications based on Docker. In: 2016 18th International Conference on Advanced Communication Technology. ICACT, IEEE, pp. 478–483.

Kao, C.-C., Chang, C.-W., Cho, C.-P., Shun, J.-Y., 2020. Deep learning and ensemble learning for traffic load prediction in real network. In: 2020 IEEE Eurasia Conference on IOT, Communication and Engineering. ECICE, IEEE, pp. 36–39.

Khazaei, H., Bannazadeh, H., Leon-Garcia, A., 2017. Savi-iot: A self-managing containerized iot platform. In: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud. FiCloud, IEEE, pp. 227–234.

Klinaku, F., Frank, M., Becker, S., 2018. CAUS: an elasticity controller for a containerized microservice. In: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. pp. 93–98.

Kovács, J., 2019. Supporting programmable autoscaling rules for containers and virtual machines on clouds. J. Grid Comput. 17 (4), 813–829.

kubernetes, 2022. What is kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

Kumar, J., Singh, A.K., Buyya, R., 2021. Self directed learning based workload forecasting model for cloud resource management. Inform. Sci. 543, 345–366.

Laptev, N., Yosinski, J., Li, L.E., Smyl, S., 2017. Time-series extreme event forecasting with neural networks at uber. In: International Conference on Machine Learning. sn, pp. 1–5.

Li, Y., Xia, Y., 2016. Auto-scaling web applications in hybrid cloud based on docker. In: 2016 5th International Conference on Computer Science and Network Technology. ICCSNT, IEEE, pp. 75–79.

Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A., 2014. A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. 12 (4), 559–592.

Meng, Y., Rao, R., Zhang, X., Hong, P., 2016. CRUPA: A container resource utilization prediction algorithm for auto-scaling based on time series analysis. In: 2016 International Conference on Progress in Informatics and Computing. PIC, IEEE, pp. 468–472.

Mishra, S.K., Sahoo, B., Parida, P.P., 2020. Load balancing in cloud computing: a big picture. J. King Saud Univ.-Comput. Inf. Sci. 32 (2), 149–158.

Morabito, R., Petrolo, R., Loscrì, V., Mitton, N., Ruggeri, G., Molinaro, A., 2017. Lightweight virtualization as enabling technology for future smart cars. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management. IM, IEEE, pp. 1238–1245.

Mozo, A., Ordozgoiti, B., Gomez-Canaval, S., 2018. Forecasting short-term data center network traffic load with convolutional neural networks. PLoS One 13 (2), e0191939.

Nguyen, H., Kieu, L.-M., Wen, T., Cai, C., 2018. Deep learning methods in transportation domain: a review. IET Intell. Transp. Syst. 12 (9), 998–1004.

Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., Kim, S., 2020. Horizontal pod autoscaling in Kubernetes for elastic container orchestration. Sensors 20 (16), 4621.

Ogunmolu, O., Gu, X., Jiang, S., Gans, N., 2016. Nonlinear systems identification using deep dynamic neural networks. arXiv preprint arXiv:1610.01439.

Pahl, C., Brogi, A., Soldani, J., Jamshidi, P., 2017. Cloud container technologies: a state-of-the-art review. IEEE Trans. Cloud Comput. 7 (3), 677–692.

Patel, Y.S., Bedi, J., 2023. MAG-D: A multivariate attention network based approach for cloud workload forecasting. Future Gener. Comput. Syst. 142, 376–392.

Qu, C., Calheiros, R.N., Buyya, R., 2018. Auto-scaling web applications in clouds: A taxonomy and survey. ACM Comput. Surv. 51 (4), 1–33.

Roy, N., Dubey, A., Gokhale, A., 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In: 2011 IEEE 4th International Conference on Cloud Computing. IEEE, pp. 500–507.

Sangpetch, A., Sangpetch, O., Juangmarisakul, N., Warodom, S., 2017. Thoth: Automatic resource management with machine learning for container-based cloud platform. In: CLOSER. pp. 75–83.

Saxena, D., Singh, A.K., 2022. Auto-adaptive learning-based workload forecasting in dynamic cloud environment. Int. J. Comput. Appl. 44, 541–551.

Swarm, 2022. Swarm mode overview. https://docs.docker.com/engine/swarm/.

Taherizadeh, S., Stankovski, V., 2019. Dynamic multi-level auto-scaling rules for containerized applications. Comput. J. 62 (2), 174–197.

Tam, F.K., 2015. The Power of Japanese Candlestick Charts: Advanced Filtering Techniques for Trading Stocks, Futures, and Forex. John Wiley & Sons.

Tutsoy, O., 2023. Graph theory based large-scale machine learning with multi-dimensional constrained optimization approaches for exact epidemiological modelling of pandemic diseases. IEEE Trans. Pattern Anal. Mach. Intell..

Varghese, B., Buyya, R., 2018. Next generation cloud computing: New trends and research directions. Future Gener. Comput. Syst. 79, 849–861.

Wang, S., Yao, Y., Xiao, Y., Chen, H., 2020. Dynamic resource prediction in cloud computing for complex system simulatiuon: A probabilistic approach using stacking ensemble learning. In: 2020 International Conference on Intelligent Computing and Human-Computer Interaction. ICHCI, IEEE, pp. 198–201.

Wang, S., Zhu, F., Yao, Y., Tang, W., Xiao, Y., Xiong, S., 2021. A computing resources prediction approach based on ensemble learning for complex system simulation in cloud environment. Simul. Model. Pract. Theory 107, 102202.

Zhong, Z., Buyya, R., 2020. A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. ACM Trans. Internet Technol. (TOIT) 20 (2), 1–24.

Zhong, Z., Xu, M., Rodriguez, M.A., Xu, C., Buyya, R., 2022. Machine learning-based orchestration of containers: A taxonomy and future directions. ACM Comput. Surv..

Zhu, Y., Zhang, W., Chen, Y., Gao, H., 2019. A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. Eurasip J. Wirel. Commun. Netw. 2019 (1), 1–15.