An event-driven and lightweight proactive auto-scaling architecture for cloud applications

Uttom Akash, Partha Protim Paul and Ahsan Habib*

Institute of Information and Communication Technology,
Shahjalal University of Science and Technology,
Kumargaon, Sylhet, Bangladesh
Email: uttom34@student.sust.edu
Email: partha-iict@sust.edu
Email: ahabib-iict@sust.edu
*Corresponding author

Abstract: The cloud environment is used by the application providers (APs) to host their applications in order to reduce procurement and management costs of the cloud resources. Moreover, the variation in the traffic load of the client applications and the appealing auto-scaling capability of the cloud resources have prompted application providers to seek ways to reduce the cost of their rented services. This paper describes a constructive auto-scaling mechanism based on the events in cloud systems fitted with heuristic predictors. The predictor examines historical data using these approaches: (1) Double Exponential Smoothing (DES), (2) Triple Exponential Smoothing (TES), (3) Weighted Moving Average (WMA) and (4) WMA with Fibonacci numbers. The outcomes of this model simulation in CloudSim indicate that the model can decrease the application provider's cost while preserving application user satisfaction.

Keywords: cloud computing; autoscaling; elastic computing; resource provisioning; exponential smoothing; triple exponential smoothing.

Reference to this paper should be made as follows: Akash, U., Paul, P.P. and Habib, A. (2023) 'An event-driven and lightweight proactive auto-scaling architecture for cloud applications', *Int. J. Grid and Utility Computing*, Vol. 14, No. 5, pp.539–551.

Biographical notes: Uttom Akash received his BSc Engineering degree in Software Engineering from Shahjalal University of Science and Technology, Sylhet, Bangladesh. He is working as a Software Engineer concerning event-driven architectures, microservices, distributed transactions and blockchain in Cefalo. His research interests include cloud computing, distributed computing, blockchain, scalability, resource provisioning and event-driven architecture.

Partha Protim Paul is working as a Lecturer in the IICT at the Shahjalal University of Science and Technology (SUST), Sylhet, Bangladesh. He has a Bachelor of Science in Software Engineering degree from the same university. His primary focus in research is automated program repair, software testing and cloud applications.

Ahsan Habib received his BSc Engineering in Computer Science and Engineering from Shahjalal University of Science and Technology, Sylhet, Bangladesh in 2004 and completed his MSc degree in Information and Communication Technology in 2012 from Bangladesh University of Engineering and Technology, Bangladesh. He received his PhD degree in Computer Science and Engineering in 2019 from Shahjalal University of Science and Technology, Bangladesh. Currently, he is working as an Assistant Professor in the Institute of Information and Communication Technology of Shahjalal University of Science and Technology, Bangladesh. He published more than 30 papers in various reputed international journals and conferences. His research interests include data compression, data management, image processing, machine learning, cloud computing and e-governance.

1 Introduction

The Internet business becomes more appealing with the introduction of cloud computing. The customers in a competitive market rent computing resources on a pay-per-use basis from Cloud Providers (CP). For example, Amazon provides virtual computers in their Elastic Compute Cloud – EC2 service. The present cloud technologies enable application vendors to rent virtual computers. These virtual machines are hosted by cloud providers. This lowers the expense of managing computing resources for application providers. Because of the difference between the frequency with which end-user requests arrive and the lack of awareness of the available resources, the expectation of the application providers is for the cloud providers to do more than just host applications. For this reason, the cloud provider wants to minimise under and over provisioning of resources.

Over-provisioning raises costs of the application providers while under-provisioning creates consumer frustration. One practical answer to this problem is to lower the cost of renting virtual machines as long as the machine can follow the Quality-of-Service (QoS) restrictions. This QoS constraint is covered by a Service-Level Agreement (SLA) contract for a cloud environment. After the QoS has been finalised, the service provider and client sign this SLA contract. A major challenge in the internet business is finding a balance between cost and performance (Jiang et al., 2017). An algorithm based on the flexibility of cloud resources can be developed to automate scale up or down resources to meet this issue (Veni and Saira Bhanu, 2016).

Based on how they handle the decision-making parameters, scaling algorithms are divided into two categories: i) Reactive and ii) Proactive.

Reactive scaling is a rule-based (threshold-based) technique that calculates the penalty for violating a service level agreement and then scales resources in response to the violation. However, there are several drawbacks to this approach. For example, in Amazon's EC2 service, an on-demand virtual machine takes 7 to 15 minutes to boot up (Islam et al., 2012). As a consequence, when the threshold is exceeded, the boot-up time for the new virtual machine delays serving the web traffic. Decision-making following a rule violation is linked to an SLA violation penalty for cloud providers (Qu et al., 2018). Some threshold violations may be caused by transient fluctuations, in which case scaling choices may not be required (Xiao et al., 2013).

The proactive method appears to be capable of covering up the flaws of the reactive approach by looking back in time and predicting future resource usages. This approach can forecast future resource requirements before crisis situations and SLA breaches arise. Furthermore, it can address the problem of delay of VM to being functional by anticipating the future.

On the other hand, researchers are generally attempting to demonstrate which scaling factors are more useful in scaling decision-making. Aslanpour and Dashti (2016) addressed how to choose resource usage factors and Response Time (RT) in their research. Nonetheless, it indicates that knowledge discovery and examination of each parameter's history can help achieve effective auto-scaling.

The aim of this research is to present an event-based auto-scaling approach for discovering and analysing information from the background of the decision-making parameters. The main objectives of this research are to develop:

- a lightweight and adaptive model;
- a forecasting algorithm that observes and predicts resource usage based on the decision-making parameters' past behaviour;
- an event-driven auto-scaling architecture based on Monitor, Analysis, Planning, Execution (MAPE) loop;
- SLA aware and Resource aware auto-scaling in cloud system.

In summary, this research presents an 'Event-Based Proactive Auto-Scaling' model to be used in cloud applications that minimises the service level agreement violation and at the same time minimises the renting cost for the application providers. The proposed model is mainly designed focusing the public and private cloud.

2 Problem analysis

As of the fourth quarter of 2020, there are 4.66 billion active internet users (KEMP, 2020) in the world. This indicates a 321 million increase compared to 2019's fourth quarter (growth rate 7.4%). As of the vast user traffic, if the auto-scaling is not focused on, it will lead to not only huge power consumption but also poor user experience. In this section we analyse the problems that auto-scaling needs to resolve.

2.1 Web traffic pattern

There are some researches which have shown that web usage behaviour follows a specific pattern (Gebert et al., 2012) which is described below.

2.1.1 Trend

The use case of a successful website such as Netflix can be taken which was gaining subscribers in every quarter till 2021 which is shown in the Figure 1. As a consequence of increasing subscribers, there were more users engagements and noticeable increasing trends in web traffic, whereas Netflix is experiencing decreasing trends from 2021 (Netflix, 2022).

2.1.2 Seasonality

Gebert et al. (2012) shown that the server handles higher load of user requests for some period. Figure 2 shows the user traffic in the interval of 5-minute over 14 days for a network supplied in the certain smaller households. The lowest user engagement is recorded from 5:00 to 6:00 a.m., when most individuals are usually asleep. Then the user engagement grows continuously throughout the day and hits the peak at about 22:00 (Gebert et al., 2012). Hence, there is seasonality in web traffic as well.

Figure 1 Number of Netflix global paid streaming subscribers by quarter

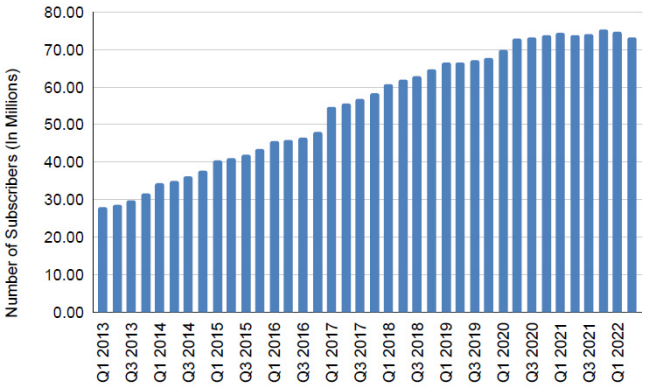
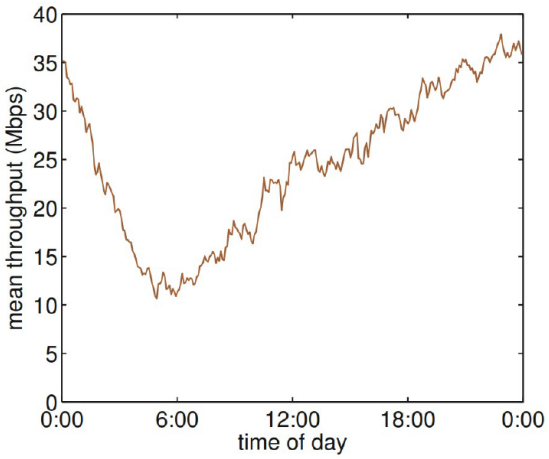


Figure 2 User traffic pattern in a day



2.2 Resource usage fluctuations

There is always a case that server might face web traffic fluctuation for some period of time. If the model is too sensitive to these fluctuations, then there will be more contradictory decisions which will lead more power consumption. A standard model should have some mechanisms to mitigate this situation.

2.3 Startup time of VM

The Startup time of VM is an another problem for autoscaling in cloud (AWS VMs take about 10 minutes to be functional). As a consequence prediction should be made at least 10 minutes before starting of VM.

2.4 Lightweight model

The proposed model should have less memory footprint (RAM usage) and low CPU usage so that it does not become a burden to cloud provider. At the same time, it should also be an adaptive model.

2.5 User experience versus cost

Generally, cloud provider always wants to keep some surplus resources to meet service level agreement and try to provide better user experience. But surplus resources should not be so much that it becomes a burden considering power consumption of data centre and cost. Hence, we may summarise the following listed issues and constraints which should be considered while designing the proposed model.

- Seasonality and trend in web traffic
- Lightweight and adaptive model
- Resource usage fluctuations
- Startup time of VMs
- Balance between user experience and cost

3 Related works

A lot of study has been done in the auto-scaling arena in recent years. The auto-scaling problem was addressed in a number of methods in these works. The majority of these researches focus on a single aspect of the subject. One field of study is devoted only to workload forecasting. These efforts are aimed at developing a method for accurately forecasting future workload for each service. Other studies aim to forecast future resource needs, while others focus on the decision-making aspect of the auto-scaling problem in order to arrive at the best scaling solution. Such judgments take into account the service's present condition as well as its resource requirements.

Islam et al. (2012) conducted research in cloud and they proposed a proactive method. In their algorithm, they used an artificial neural network. The authors claim that their algorithm is capable of scaling based on resource utilisation awareness. They measured the performance of neural networks. But they did not investigate their algorithm's efficiency in comparison with that of the reactive method.

A resource-aware solution was proposed in CloudSim by Huang et al. for a real-world application (Huang et al., 2012). However, they did not include pricing, which is considered one of the most significant application demands.

Mohamed et al. (2014) studied SLA-aware resource scaling using a reactive approach. However, in this study, the resource status was not taken into account. They were unable to study the criteria of SLA violations, costs and resources usage.

García et al. (2014) also proposed a method based on reactive technique. In their research, they suggested a resource management architecture using a horizontal scaling algorithm. They focused on cloud cost to assess their algorithm, and while calculating cost, they did not consider the SLA violation penalty.

Qavami et al. (2014) presented an approach for predicting resources for application using resource aware proactive methods. As part of their decision-making process, they use data from the state of resource usage of cloud applications. In this case, they calculated the cost without considering the penalty for SLA violations.

De Assuncao et al. (2015) evaluated the SLA-focused work in form of user patience in their work on response time

from applications. They employed a proactive strategy called weighted exponential smoothing in their approach. This strategy assisted in improving the research result by proactively keeping track of the response time and the forecast of resource usage.

Gill and Chana (2015) presented a proactive approach based on QoS constraints. The impact of the SLA breach penalty on the cost was not taken into account in this method. The focus of their research was supplying of the heterogeneous resources.

Moltío et al. (2016) examined the use of cloud computing. They provided a reactive resource management architecture in their research. This auto-scaling technique takes a resource conscious approach. In this design, the algorithm is given decision-making capabilities but it is merely aware of resource (memory) usage. Furthermore, the recommended strategy was not contrasted with proactive alternatives. Again, they did not compare the approaches in terms of the criterion for SLA violation cost and SLA awareness.

Ajila and Bankole (2016) proposed a SLA-aware and resource-aware algorithm. However, they solely assessed the outcomes of time series-related criteria in their research and did not include cost or QoS factors.

Ghobaei-Arani et al. (2016) proposed a proactive method that uses resource utilisation parameters for autoscaling. The use of linear regression to analyse the arrival rate of user requests is also useful in their study. Their method reduces costs and SLA violations, and they use Learning Automata to make scaling decisions.

Aslanpour and Dashti (2017) proposed a proactive autoscaling algorithm for cloud application. In their research they tried to minimise the overall cloud cost while maintaining user satisfaction. But in their research, they could have reduced the cost more if they had use a different approach.

Ye et al. (2017) suggested a combined technique for anticipating the required resources in order to maintain the SLA even though the workload of the system varies substantially. In this study, the scaler unit determines required resources needed for the next time interval based on the projected value. In this scenario, the method used to forecast time series is ARMA. In addition, another quick provisioning algorithm is also used in this research. The technique assumes that vertical scaling should be employed to enable rapid resource provisioning in the face of severe workload variances in order to fulfil SLA requirements. In this study, the seasonality of data is not taken into consideration. Another disadvantage of this method is that some of the threshold values must be established by a human operator. In other works, the ARMA model has been used to predict time series. Roy et al. (2011) used this method to forecast the workload of the service. The proposed method in this paper is less accurate than many existing predictive algorithms, and it performs poorly when it comes to remembering the workload from previous timesteps.

Martinez et al. (2017) provided a technique for learning the service's workload patterns, which are repeated on a regular or seasonal basis, and these patterns were utilised to estimate workload of the service. A datastore is used in this method to store all relevant workload patterns. In the datastore, some patterns are preloaded. Based on the manually acquired information, further patterns may be incorporated to the system (e.g., with the help of a human operator). Each pattern has a number of features that can be used to compare a real workload to previously identified patterns. During runtime, this function attempts to match the current workload pattern with a known pattern in the datastore. The authors utilise the pattern's form and related attributes to match the current data with one of the patterns stored in the datastore. The pattern is denormalised using the actual workload values seen in runtime to anticipate future workload. The actual benefit of this method is that it anticipates service demand based on seasonal and periodic trends, making it ideal for systems with predictable workloads. However, this method is inapplicable to many current systems with changing workloads, resulting in poor job performance.

In the two different researches done by Velásquez et al. (2013) and Makridakis and Spiliotis (2018), it is observed that the computational requirements of machine learning are far greater than statistical approaches and exponential smoothing has a better fit over seasonal data. But ML has better accuracy in predicting data for unknown circumstances. The main focus of this research is to design a lightweight model. The lightweight exponential smoothing is used with improved adaptability and accuracy in the proposed approach.

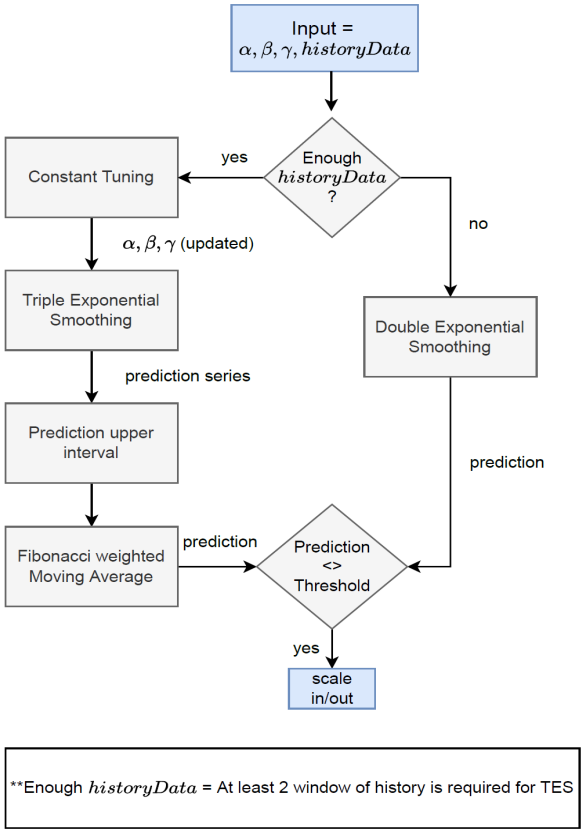
4 The proposed approach

The proposed model is designed considering the issues from the previous Section 2. The model is designed using heuristic called Triple Exponential Smoothing (TES). Firstly, TES can smooth trends and seasonality in time series (Billah et al., 2006; Singh et al., 2019a). Secondly, TES is lightweight which is one of the constraints and TES can also be highly adaptive by tuning its constants. Upper interval prediction is used to keep surplus resources and Fibonacci weighted moving average of upper interval of predictions is used to deal with traffic fluctuations.

4.1 Switching of exponential smoothing heuristics

The proposed model chooses Triple Exponential Smoothing or Double Exponential Smoothing depending on availability of web traffic data. The model's first choice is Triple Exponential Smoothing which can smooth trends as well as seasonality whereas Double Exponential smoothing can smooth trends only. But Triple exponential smoothing requires a minimum of two seasons of data. If enough data are not available, the proposed model chooses Double Exponential Smoothing. If the proposed model chooses Triple Exponential Smoothing, the model pre-processes the constants of TES and post processes prediction, which is shown in Figure 3. TES along with pre-process and post-process is discussed below.

Figure 3 Data flow diagram of the proposed model



4.2 *Initialisation of triple exponential smoothing value*

Triple Exponential Smoothing calculates overall smoothed value l_t, smoothed trend b_t, smoothed seasonality s_t and m-step ahead forecast f_{t+m} at time t using equations (1), (2), (3) and (4), respectively.

$$l_t = \alpha(x_t - s_{t-L}) + (1-\alpha)(l_{t-1} + b_{t-1}) \quad (1)$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \quad (2)$$

$$s_t = \gamma(x_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-L} \quad (3)$$

$$f_{t+m} = l_t + m * b_t + s_{t-L+1+(m-1) \bmod L} \quad (4)$$

where α is the overall smoothing factor, β is the trend smoothing factor and γ is the seasonal change smoothing factor. And x_t is the record in the time series at time t and L is the number of period in one season.

At the initial stage, the initial value for the smoothed trend b_0 and the smoothed seasonality s_t is calculated using equations (5) and (6), respectively.

$$b_0 = \frac{1}{L}\left(\frac{x_{L+1} - x_1}{L} + \frac{x_{L+2} - x_2}{L} + \ldots + \frac{x_{L+L} - x_L}{L}\right) \quad (5)$$

$$s_t = \frac{1}{N}\sum_{j=1}^{N}\frac{x_{L*(j-1)+t}}{A_j} \quad (6)$$

$$A_j = \frac{\sum_{j=1}^{N} x_{L*(j-1)+t}}{L}$$

where $j = 1, 2, \ldots, N$ and N = Number of seasons. If the constants are initialised as $\alpha = 0, \beta = 0, \gamma = 0$, then we get the value of equations (1), (2) and (3) as follows:

$$l_t = \left(l_{t-1} + b_{t-1}\right)$$
$$b_t = b_{t-1}$$
$$s_t = s_{t-L}$$

In this case, the model doesn't give any priority to current records in history data and this can cause following problems:

- Current traffic history values have no impact on the proposed model.
- Increasing or decreasing traffic won't contribute to the estimations.

Hence, the model has to tune its constants automatically to adapt to the web traffic. Initially, the proposed model initialises it's constants with some static values $(\alpha = 0.4, \beta = 0.1, \gamma = 0.5)$. The adaptation process of proposed model has been discussed in the next section.

4.3 *Tuning of constants of triple exponential smoothing*

An adaptive approach is used in the proposed model to adjust the constants for triple exponential smoothing. The steps of this approach are described below:

- One of the three constants (α, β, γ) is randomly selected.
- A little bias is selected randomly.
- Upper interval of the selected constant is calculated by adding the bias to the selected constant.
- Lower interval of the selected constant is calculated by subtracting the bias from the selected constant.
- MSE is calculated using TES with calculated constants.
- One of the constant out of three (current constant, upper interval, lower interval) is selected for which MSE is minimum.

In this way, one constant is adjusted with history data in one iteration.

4.4 *Prediction upper interval*

The model keeps some surplus resources to provide better user experience according to QoS as defined in the SLA. At the same time, it tries to minimise the cost. Hence, the model selects the surplus resources in a systematic way by using the prediction upper interval. The upper interval of prediction $upperF_m$ is calculated using the following equation with 95% confidence interval:

$$upperF_m = f_{t+m} + z * \sigma_{t+m} + c \quad (7)$$

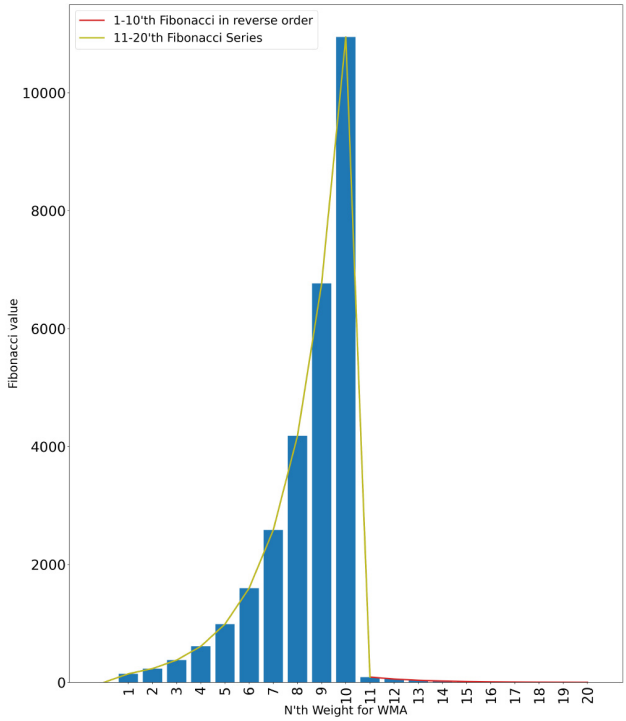
where f_{t+m} = m-step ahead forecast at time t, z = quantile, σ = standard deviations using well-known Brutlag's algorithm, c = offset according to QoS.

During planned unusual high traffics, the QoS parameters z and c can be specified in the SLA which will be used to calculate the prediction upper interval. Moreover, these parameters can be specified independently for a particular period of time. To tackle suspicious unusual high traffic, cloud providers generally do have another shield mechanism such as the AWS Shield (AWS, 2022). Those traffics cannot enter through the shield and are not monitored by model.

4.5 *Fibonacci weighted moving average (WMA)*

The model uses the weighted moving average of the upper interval of the predictions of 20 minutes as final prediction. In the proposed approach, the model uses Fibonacci series as weights.

Figure 4 Proposed order of Fibonacci series



Firstly, The Fibonacci series is used in this model because the increasing ratio in Fibonacci series is greater than the ordinal series. Secondly, the model rearranges the order of Fibonacci series. Here, the model takes the first 20 numbers from the Fibonacci series, breaks the series into two equal parts, reverse the first part and then swaps the two parts. The final Fibonacci series is shown in the Figure 4. It is rearranged in this way to give more weights to the prediction point and recent values in predictions. So, the prediction $f_t + 10$ at 10 minutes will get the most weight and $f_t + 20$ at 20 minutes will get the least weight. The value of wma is calculated using following equation:

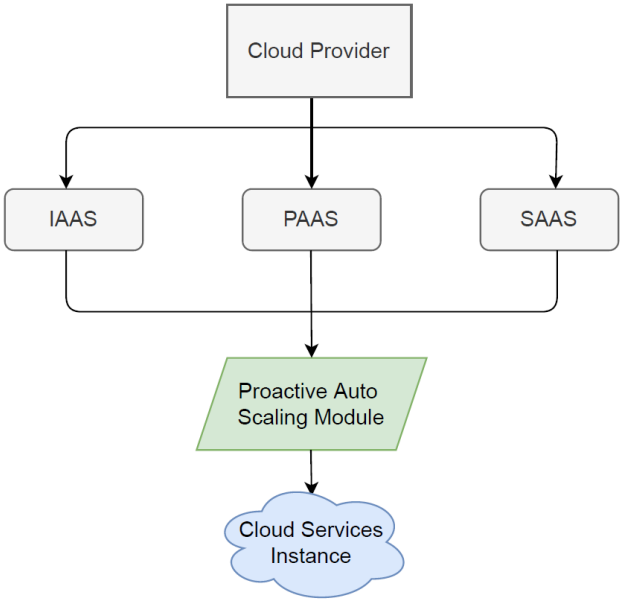
$$wma = \frac{\sum_{m=1}^{n} (fiboW_m * upperF_m)}{\sum_{m=1}^{n} (fiboW_m)} \tag{8}$$

where $fiboW_m$ = m'th Fibonacci number from the Figure 4.

5 Auto-scaling module

The cloud providers provide IaaS, PaaS and SaaS services to their customers. The cloud providers would use proposed Proactive Auto Scaling Component (PASC) and scale their service instances for better user experience. The Figure 5 indicates where the proposed proactive auto-scaling module is used in cloud architecture.

Figure 5 PASC integration in cloud environment



The proposed module regionally maintains the Service Level Agreement (SLA) and performs the scaling actions regionally as well. The Load Balancer (LB) will be responsible for distributing the requests to the appropriate server based on the region of the request initiator and traffic load.

5.1 *Monitor component*

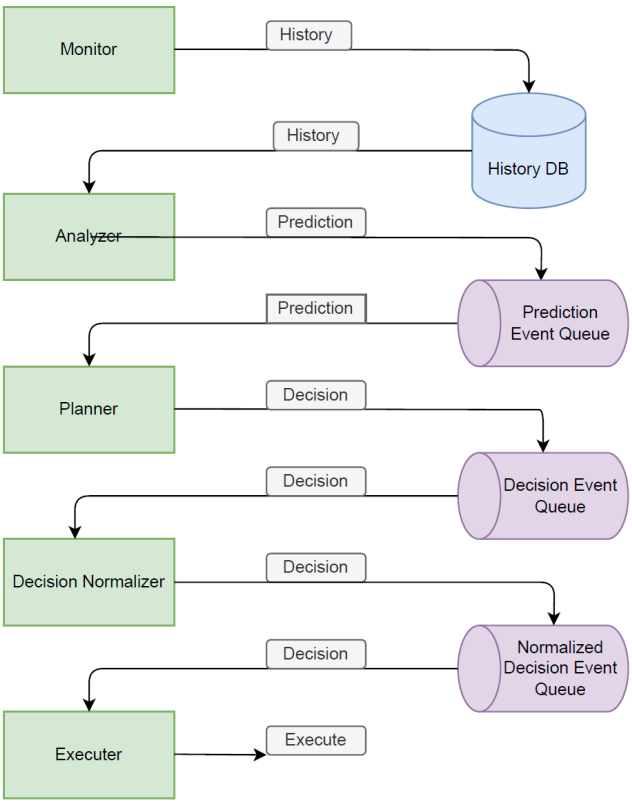
Monitor component monitors prediction parameters from the system. These prediction parameters are CPU utilisation, VMs Count, Throughput, Response Time, Delay Time, SLA violation count, SLA violation percentage, SLA violation time, Failed Cloudlets and Workload. These prediction parameters are collected from the system and added to our history database by this component which is shown in Figure 6.

CPU utilisation is the percentage for serving all the requests for a time period. The average CPU utilisation $\overline{CpuUtil_t}$ during time t is calculated using equations (9) and (10):

$$\overline{CpuUtil_t} = \frac{\sum_{i=1}^{VM_{online}} VmUtil_i}{VM_{online}} \tag{9}$$

$$VmUtil_i = \frac{\sum_{j=1}^{R_i} (P_j * L_i)}{PES_i * MIPS} \tag{10}$$

Figure 6 Proactive auto-scaling module



where

$VmUtil_i$ = The CPU efficiency of *i*-th VM

VM_{online} = Number of functional/Online machines.

R_i = The number of current requests being served by the *i*-th VM.

P_j = Required number of processing elements for the *j*-th request.

L_i = The number of instructions of the *j*-th request.

PES_i = The number of processing elements of the *i*-th VM.

$MIPS$ = Each processing element's processing power.

5.2 *Analyser component*

Analyser component collects recent data from the history database for three windows. This component analyses data of ten prediction parameters collected by the monitor component in the previous stage. The functions of this component are to: 1) Tune constants of TES, 2) Apply TES on the history data series, 3) Use Fibonacci moving average to make the final prediction.

This component uses an adaptive approach to tune the constants for triple exponential smoothing. The proposed Algorithm 1 uses randomisation to tune constants.

Algorithm 1 illustrates the pseudo-code of constant tuning. At first a little bias is randomly selected at line 5. In line number 6, it selects one of the three constants (α, β, γ) randomly. Then the algorithm takes the upper value by adding bias to the selected constant in line number 8. Then the lower value is calculated by subtracting bias from the selected constant in line number 9. Then the mean squared error is calculated for 3 recent windows in lines 11–17. After that, it picks the minimum MSE_{min} out of the three calculated constants $MSE_{current}$, MSE_{up}, and MSE_{low} in line 19. Finally, the algorithm selects one of the current, upper or lower constant for which Mean Square Error (MSE) is minimum in lines 21–27. In this way every time the model comes closer to the best value for any constant. The complexity of Algorithm 1 is $O(N)$ where N is the length of data series.

Algorithm 1: TuneConstant

```
   input : dataSeries //Records of three
           recent windows
           L //Number of period in season
           c //Offset according to QoS
           z //Quantile in confidence
   interval
           numberOfPredictions
           α, β, γ
   output: const_tunned
   // Initialization
 1 const_current ⟵ {α, β, γ};
 2 const_up ⟵ const_current;
 3 const_low ⟵ const_current;
 4
   // Select bias and indice randomly
   // The range of the bias is 0.0 - 0.02
 5 bias = randDouble(0.02);
   // Indice range will be 0-2
 6 indice = randInteger(3);
 7
 8 const_up[indice] ⟵ const_up[indice] + bias;
 9 const_low[indice] ⟵ const_low[indice] − bias;
10
   // Passing all required parameters and
      modified consts in forecast
11 upperF_current ⟵ forecast(...,const_current);
12 upperF_up ⟵ forecast(...,const_up);
13 upperF_low ⟵ forecast(...,const_low);
14
15 MSE_current ⟵ mse(upperF_current,dataSeries);
16 MSE_up ⟵ mse(upperF_up,dataSeries);
17 MSE_low ⟵ mse(upperF_low,dataSeries);
18
19 MSE_min = min(MSE_current,MSE_up, MSE_low)
20
21 if MSE_min = MSE_current then
22 |  const_tunned ⟵ const_current ;
23 else if MSE_min = MSE_up then
24 |  const_tunned ⟵ const_up;
25 else
26 |  const_tunned ⟵ const_low;
27 end
```

After that, the analyser component applies TES on the history data series to forecast predictions for next twenty minutes using the following forecast Algorithm 2.

Algorithm 2: Forecast

```
   input : dataSeries //Records of three
           recent windows
           L //Number of period in season
           c //Offset according to QoS
           z //Quantile in confidence
   interval
           numberOfPredictions
           α, β, γ
   output: upperF //upper intervals of
           predictions
           wma //Final prediction
   // Initialization
 1 l_0 ⟵ dataSeries[0];
 2 b_0 ⟵ equation 5;
 3 s ⟵ equation 6;
 4
   // forecast
 5 for i ← 1 to length_dataSeries do
 6 |   x_i ⟵ dataSeries[i];
 7 |   l_i ⟵ equation 1;
 8 |   b_i ⟵ equation 2;
 9 |   s_i ⟵ equation 3;
10 |   s[i%L] ⟵ s_i;
11 |   f[i] ⟵ l_i + b_i + s[i%L];
12 |   σ[i] ⟵ γ * abs(x_i − f[i]) + (1 − γ) * σ[i − 1];
13 end
14 for m ← 1 to numberOfPredictions do
15 |   f[i + m] ⟵ equation 4;
16 |   σ[i + m] ⟵ γ * σ[i];
17 |   upperF[m] ⟵ equation 7;
18 end
19
   // Calculate weighted moving average
20 wma ⟵ equation 8;
```

In the Algorithm 2, the model calculates the initial trend, seasonal indices in lines 1–3. Then, the algorithm calculates level (l_i), trend (b_i) and seasonality (s_i) in lines 7–9 from the data series. Then, the model learns forecasting from history data series in line 11. And the model calculates deviations (σ) in line 12. After that, the model forecasts (f) in line 15. Then, the algorithm calculates the upper interval of the forecast $(upperF)$ in line 17. Finally, it calculates the weighted moving average of the upper interval of the forecasts in line 20. Its complexity is $O(N)$ where N is the length of data series.

The calculated weighted moving average (*wma*) pushed into *Prediction Event Queue* which is shown in Figure 6. The planner component retrieves the values from the queue later.

5.3 Planner component

Planner component takes prediction events from queue (*PredictionEventQueue*) as input and uses them to take scaling decisions accordingly. This component first calculates the SLA threshold and then the planner component decides scaling event (scale down or up) using this value.

From the queue (*PredictionEventQueue*) the model gets the event (*predictionEvent*) that is generated in the analyser component. This *predictionEvent* contains information about SLA identifier. After that, the proposed model collects the SLA for that *predictionEvent* using SLA identifier. This SLA contains all the information about the degree of service expected by a consumer from a cloud service provider.

After collecting the SLA for the *predictionEvent*, the comparison between the SLA threshold with the prediction is made. If the prediction is less than threshold, the proposed model calculates the number of scaled-down decision. After that the proposed model generates event (*decisionEvent*) for each scale down decisions. Otherwise, the proposed model generates *decisionEvent* for each scale up decision in the same process. The generated *decisionEvent* events are pushed to the queue (*decisionEventQueue*) to use as input in the later component.

5.4 Normaliser component

Normaliser component takes value from queue (*decisionEventQueue*) as input and normalise them.

At the first step, the proposed model pops the first decision event and then peeks the second decision event from the queue. In the second step, the model checks whether the first and the second decision events are contradictory or not. If they are contradictory decisions and the scaling time difference is less than *contradictoryDET* (contradictory Decision Event Time) then they practically eliminate each other. In this scenario, the component pops second event from the queue and does not push any event to the normalised decision event queue.

If both the decision events are not contradictory or the scaling time difference between them is greater than *contradictoryDET*, then the component pushes the first event to the Normalised Decision Event queue.

5.5 Executor component

The executor component checks the events from the Normalised Decision Event queue. If the event is scale-up, this component performs scale up in the cloud service instance. Otherwise, the component figures out which cloud instance will be taken down. After that, the component scales down that selected cloud instance.

6 Experimental result and analysis

To evaluate the proposed event-driven proactive approach, it is compared with another existing heuristic Proactive Auto-Scaling Algorithm (PASA) which is proposed in Aslanpour and Dashti (2017). The PASA model is used in the comparison because this model is based on the lightweight heuristic algorithm and it outperforms the state-of-the-art reactive models. Both of these models are evaluated in the same setup using the CloudSim framework. We have used the NASA-HTTP[1] dataset for the simulation with CloudSim.

At first, a workload imitates the behaviour of web users in the web to test our proposed model. This workload generates 567,783 web user requests. The VM configuration was inspired by Amazon's EC2 service. The configuration of VM is shown in the Table 1.

Table 1 Configuration of VM

Purchase	*CPU*	*Memory*	*Storage*	*Price*
Reserved	2 (2.5 up to 3.3)	2GiB	EBS only	$1030 (1-year term)
On-Demand	2 (2.5 up to 3.3)	4GiB	EBS only	$0.08 (per hour)

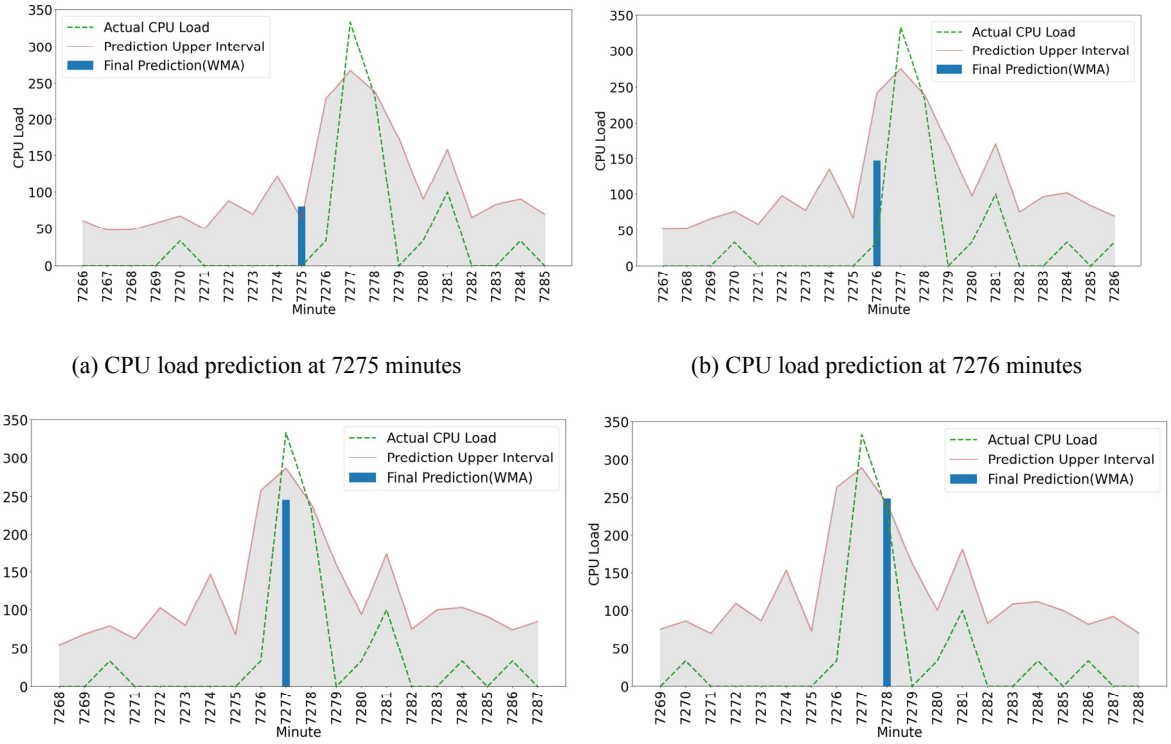
The Monitoring phase of the proposed scaling model is executed in every minute. The mechanism for evaluating resource usage and RT was discussed in the preceding section (Auto Scaling Module). The lower-and upper-scaling threshold for resource utilisation is set to 30% and 70%, respectively. The lower-and upper-scaling threshold is set to 200 ms and 1000 ms for RT, respectively. The SLA for Response Time (RT) is set to 1000 ms. Besides, the delay of VM startup is set to 10 minutes. The initial values for TES constants are $\alpha = 0.6, \beta = 0, \gamma = 0.06$.

6.1 Prediction analysis

The predictions were made using the NASA-HTTP data set as actual CPU Loads. Here one window is equal to one day (1440 minutes). So, one window consists of 1440 records.

In the Figures 7 and 8, the final prediction of the proposed model is shown by the blue bar at 7275 minutes (6th day 1 hr 15 minutes), 7276 minutes (6th day 1 hr 16 minutes), 7277 minutes (6th day 1 hr 17 minutes), 7278 minutes (6th day 1 hr 18 minutes), 7279 minutes (6th day 1hr 19 minutes) and 2781 minutes (6th day 1 hr 21 minutes), respectively and these predictions were made 10 minutes earlier. In these figures, actual CPU Load is indicated by the dotted green line, upper interval of predictions is shown by grey area covered by the red line and the final prediction is shown by the blue bar.

Figure 7 CPU load prediction validation with actual CPU load (see online version for colours)



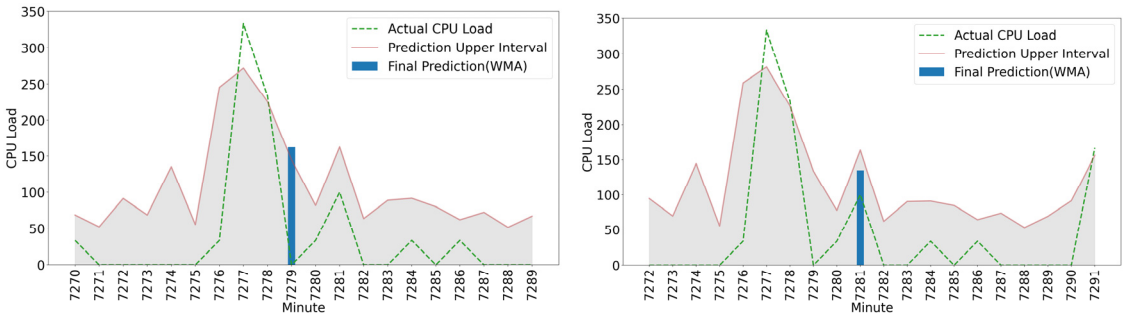
(a) CPU load prediction at 7275 minutes

(b) CPU load prediction at 7276 minutes

(c) CPU load prediction at 7277 minutes

(d) CPU load prediction at 7278 minutes

Figure 8 CPU load prediction validation with actual CPU load (see online version for colours)



(a) CPU load prediction at 7279 minutes

(b) CPU load prediction at 7281 minutes

In Figure 7(a), the proposed model predicted the CPU load at 7275 minutes when current time was 7265 minutes. In this iteration, the model took 3 recent windows of past records. Using these selected records, the model provided the upper interval of predictions for the next 20 minutes which is shown by the red line in the figure from 7266 minutes to 7285 minutes. Finally, the model took the weighted moving average providing the most weight to the prediction at time 7275 minutes and provided the final result of CPU load at 7275 minutes which is indicated by the blue bar.

In Figure 7(a), the model predicted surplus CPU loads since it already knew there would be more CPU load right after 2775 minutes. After that, in Figure 7(b), the model predicted the required CPU load at 2776 minutes. In Figure 7(c) and 7(d), the model forecast less because there was a fluctuation of traffic for 1 minute. But the handling of the fluctuations is configurable considering how much the model should be smooth. Finally, in the Figure 8(a) and 8(b), the model smoothly predicted the scaling down decision.

6.2 Result evaluation

The effectiveness of an auto-scale model depends on different criteria (Singh et al., 2019b; Lorido-Botran et al., 2013). Some well-known criteria for proving the effectiveness (Singh et al., 2019b) of a scaling algorithm that we considered are: (i) Standard Deviation (SD) of resource utilisation, (ii) Response Time (RT), (iii) SLA violation Percentage, (iv) Time to Adaption, (v) SLA Penalty and (vi) Costs.

Table 2 depicts a comparison of the standard deviation of the resource utilisation of the proposed model with the existing approach PASA. A lower score in this criteria implies better control over fluctuations and balance overload management for cloud applications. Results show that the proposed model has better control over fluctuations than the existing model PASA. This is because the model does not confine itself within the decision-making time period only. The proposed model looks ahead of that time period. Then, it takes advantage of using weighted moving averages on prediction series. Hence, resource usage anomalies do not affect the proposed model. On the contrary, the existing model confines itself to the decision-making time. Therefore, the proposed approach manages to mitigate temporary load fluctuations more effectively.

Table 2 Comparison of fluctuations of CPU utilisation between proposed and PASA model

Model name	*Resource utilisation (SD)*
PASA Model	32.114 %
Proposed Model	27.609 %

Table 3 shows the comparisons of average response and delay time between the proposed model and the existing model PASA. The results show that the proposed model outperformed the existing model PASA. This is because our model predicts the resource usage 10 minutes earlier and activates the scaling decision accordingly. So requests do not need to wait for the execution of scaling decisions. Also, our model assures some surplus resources in a systematic way according to the SLA. Again, our model smoothens the contradictory decision. This reduces our system response time and minimises delay time.

Table 3 Comparison of response time and delay time between proposed & PASA model

Model name	*Response time (Avg)*	*Delay time (Avg)*
PASA Model	0.2948 Sec.	0.0951 Sec.
Proposed Model	0.2394 Sec.	0.0397 Sec.

Table 4 represents a comparison of the adaption of the proposed model with the existing approach PASA. This criterion indicates the sum of the average delay time for a new VM to adapt to the new request. From the results, it is observed that our model takes less time to adapt than the existing model PASA. Firstly, it has already been seen that the proposed model has better control over resource utilisation fluctuations according to Table 2, so our model faces less scaling overheads. Secondly, our model scales the VMs earlier than the requirement of VMs and provides enough time to adapt. This leads to less adaptation time for a request in our model.

Table 4 Comparison of time to adaption between proposed and PASA model

Model name	*Time to adaption*
PASA Model	132.632 sec.
Proposed Model	81.120 sec.

Table 5 illustrates the comparisons of the percentage of SLA Violation between the proposed model and the model PASA. This criterion indicates the percentage of user application requests that are handled with a longer delay than the SLA allows. The higher the proportion, the more dissatisfied the users are. From the results, it is observed that the proposed model breaks minimum number of service level agreements than the existing model PASA. Since the proposed model has less-response time, less-delay time and less-adaption time according to Tables 3 and 4, this lead to less-SLA violation than the other model.

Table 5 Comparison of SLA violation between proposed and PASA model

Model name	*SLA violation (%)*
PASA Model	18.444
Proposed Model	9.512

Table 6 Comparison of costs between proposed and PASA model

Model name	*SLA penalty cost*	*Renting cost*	*Total cost*
PASA Model	5.36 $	32.60 $	37.96 $
Proposed Model	1.84 $	34.28 $	36.12 $

Table 6 depicts a comparison of the total costs, SLA penalty costs and renting costs of the proposed model with the existing model PASA. The costs of on-demand VMs are considered in this experiment. SLA violation penalties are also taken into consideration during the calculation of the cost of VM rent. Total cost is calculated with equation (11).

$$cost_{total} = cost_{rent} + cost_{penalty} \tag{11}$$

where $cost_{rent}$ is the total cost of renting of the VMs which is calculated using equation (12).

$$cost_{rent} = \sum_{i=1}^{VM} h_i * p_i \tag{12}$$

where VM = total number of VM. h_i = active hours of i-th VM. p_i = hourly price of i-th VM as stated by the Table 1.

And SLA Penalty cost is calculated using equation (13).

$$cost_{penalty} = Ceil\left(\frac{\sum_{i=1}^{SLA_{violents}} (RT_i - 1)}{60 * 60}\right) * p \tag{13}$$

where $cost_{penalty}$ = The cost charged on the application provider for SLA contract violation. $SLA_{violents}$ = the requests with a response time of more than $1sec$. RT_i = the response time for i-th request. p = price of per hour of sla violation.

From Table 6 it is observed that the proposed model reduces overall cost compared to the model PASA. From Table 5 it is also observed that our proposed approach minimises SLA violations than other model. This reduces our SLA penalty cost. The renting cost of the proposed model is slightly greater than the existing model PASA. We know that there is a trade-off between renting costs and the cost of SLA penalty (Fé et al., 2022). The proposed model is able to find a position where we can keep this trade-off minimum and reduce total cost.

7 Limitations and threats to validity

Instead of an actual cloud environment, we used the 'CloudSim' simulator to test the validity and impacts of our suggested model. CloudSim is a comprehensive framework for modelling cloud computing systems and measuring application service performance, allowing users to evaluate the performance of their provisioning policies free of cost (Calheiros et al., 2011). If we were to put our model in the real cloud, the most significant difference would be in data centre power costs, which would affect cloud providers' costs, because the socioeconomic and environmental characteristics of the geographical location in which a data centre is located have a direct impact on the total amount of power spent. A data centre, e.g., hosted in a place with low-power costs and less severe weather conditions would have cheaper power expenditures. This type of cost fluctuation has yet to be determined by CloudSim (Buyya et al., 2009). Aside from this, we can consider CloudSim to be a simulation of a real-world cloud environment.

8 Conclusion and future work

Because of the flexibility of cloud computing, many manual activities are no longer necessary and are replaced by automated processes. Auto-scaling reduces the need for continuing service monitoring to increase or decrease the scale, reducing maintenance costs and fines for SLA breaches for businesses. Compared to earlier literature, one of the key aims of the research undertaken in this work was to provide a lightweight method to increase forecasting and decision-making accuracy. An event-based proactive method for cloud instances is provided in this research. This method combines prior studies of the parameter, useful in scaling in decision-making with a heuristic Triple Exponential Smoothing-based strategy. The simulated result of this proposed model which is simulated in CloudSim is summarised as: i) minimised scaling overhead, ii) improved control over variations in resource consumption, iii) improved response time of user requests, iv) minimised SLA violation and v) minimised the overall cost placed on application provider.

In this research, the proposed heuristic parameters measuring technique outperformed the existing method in terms of both resource-aware and SLA-aware. The future work of this research is to incorporate the model in both hybrid and multi-cloud environments.

Acknowledgement

This research was supported by the Research Centre of Shahjalal University of Science and Technology, Sylhet, Bangladesh.

References

Ajila, S. and Bankole, A. (2016) 'Using machine learning algorithms for cloud client prediction models in a web VM resource provisioning environment', *Transactions on Machine Learning and Artificial Intelligence*, Vol. 4, pp.28–51.

Aslanpour, M.S. and Dashti, S. (2017) 'Proactive auto-scaling algorithm (pasa) for cloud application', *International Journal of Grid and High Performance Computing*, Vol. 9, pp.1–16.

Aslanpour, M.S. and Dashti, S.E. (2016) 'Sla-aware resource allocation for application service providers in the cloud', *Proceedings of the 2nd International Conference on Web Research (ICWR)*, pp.31–42.

AWS (2022) *Managed DDoS Protection – AWS Shield – Amazon Web Services*.

Billah, B., King, M., Snyder, R. and Koehler, A. (2006) 'Exponential smoothing model selection for forecasting', *International Journal of Forecasting*, Vol. 22, pp.239–247.

Buyya, R., Ranjan, R. and Calheiros, R.N. (2009) 'Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: challenges and opportunities', *Proceedings of the International Conference on High Performance Computing and Simulation*, IEEE, Germany, pp.1–11.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F. and Buyya, R. (2011) 'Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms', *Software. Practice Experience*, Vol. 41, No. 1, pp.23–50.

De Assuncao, M., Cardonha, C., Netto, M. and Cunha, R. (2015) 'Impact of user patience on auto-scaling resource capacity for cloud services', *Future Generation Computer Systems*, Vol. 55, pp.41–50.

Fé, I., Matos, R., Dantas, J., Melo, C., Nguyen, T. A., Min, D., Choi, E., Silva, F. A. and Maciel, P.R.M. (2022) 'Performance-cost trade-off in auto-scaling mechanisms for cloud computing', *Sensors*, Vol. 22, No. 3. Doi: 10.3390/s22031221.

García, A.G., Espert, I.B. and García, V.H. (2014) 'Sla-driven dynamic cloud resource management', *Future Generation Computer Systems*, Vol. 31, pp.1–11.

Gebert, S., Pries, R., Schlosser, D. and Heck, K. (2012) 'Internet access traffic measurement and analysis', in Pescapíe, A., Salgarelli, L. and Dimitropoulos, X. (eds): *Traffic Monitoring and Analysis*, Springer, Berlin, Heidelberg, pp.29–42.

Ghobaei-Arani, M., Jabbehdari, S. and Pourmina, M. (2016) 'An autonomic approach for resource provisioning of cloud services', *Cluster Computing*, Vol. 19, pp.1017–1036.

Gill, S.S. and Chana, I. (2015) 'Q-aware: quality of service based cloud resource provisioning', *Computers Electrical Engineering*, Vol. 47, pp.138–160.

Huang, J., Li, C. and Yu, J. (2012) 'Resource prediction based on double exponential smoothing in cloud computing', *Proceedings of the 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pp.2056–2060.

Islam, S., Keung, J., Lee, K. and Liu, A. (2012) 'Empirical prediction models for adaptive resource provisioning in the cloud', *Future Generation Computer Systems*, Vol. 28, No. 1, pp.155–162.

Jiang, F., Hsu, C. and Wang, S. (2017) 'Logistic support architecture with petri net design in cloud environment for services and profit optimization', *IEEE Transactions on Services Computing*, Vol. 10, No. 6, pp.879–888.

KEMP, S. (2020) *Digital 2020: Global Digital Overview*.

Lorido-Botran, T., Miguel-Alonso, J. and Lozano, J. (2013) 'Comparison of auto-scaling techniques for cloud environments', in Alberto, A. and Del Barrio, G.B. (eds.): *Actas de las XXIV Jornadas de Paralelismo*, Servicio de Publicaciones, pp.1–7.

Makridakis, S. and Spiliotis, E.A.V. (2018) 'Statistical and machine learning forecasting methods: Concerns and ways forward', PLoS One, Vol. 13, No. Doi: 10.1371/journal.pone.0194889.

Martínez, R.G., Li, Z., Lopes, A. and Rodrigues, L. (2017) 'Augure: proactive reconfiguration of cloud applications using heterogeneous resources', *Proceedings of the IEEE 16th International Symposium on Network Computing and Applications (NCA)*, IEEE, pages 1–8.

Mohamed, M., Amziani, M., Bela¨ıd, D., Tata, S. and Melliti, T. (2014) 'An autonomic approach to manage elasticity of business processes in the cloud', *Future Generation Computer Systems*, Vol. 50, pp.49–61.

Moltío, G., Caballer, M. and De Alfonso, C. (2016) 'Automatic memory-based vertical elasticity and oversubscription on cloud platforms', *Future Generation Computer Systems*, Vol. 56, No, C, pp.1–10.

Netflix (2022) *Number of Netflix Global Paid Streaming Subscribers by Quarter*.

Qavami, H., Jamali, S., Akbari, M. and Javadi, B. (2014) 'Dynamic resource provisioning in cloud computing: a heuristic Markovian approach', *Proceedings of the International Conference on Cloud Computing*, pp.102–111.

Qu, C., Calheiros, R.N. and Buyya, R. (2018) 'Auto-scaling web applications in clouds: A taxonomy and survey', *ACM Computing Surveys*, Vol. 51, No. 4, pp.1–33.

Roy, N., Dubey, A. and Gokhale, A. (2011) 'Efficient autoscaling in the cloud using predictive models for workload forecasting', *Proceedings of the IEEE 4th International Conference on Cloud Computing*, pp.500–507.

Singh, K., Shastri, S., Bhadwal, A., Kour, P., Kumari, M., Sharma, A. and Mansotra, V. (2019a) Implementation of exponential smoothing for forecasting time series data', *International Journal of Scientific Research in Computer Science Applications and Management Studies*, Vol. 8, No. 1, pp.1–5.

Singh, P., Gupta, P., Jyoti, K. and Nayyar, A. (2019b) 'Research on auto-scaling of web applications in cloud: survey, trends and future directions', *Scalable Computing: Practice and Experience*, Vol. 20, pp.399–432.

Velásquez, J., Zambrano, C. and Franco, C. (2013) 'A comparison of exponential smoothing and neural networks in time series prediction', *Dyna (Medellin, Colombia)*, Vol. 80, pp.66–73.

Veni, T. and Saira Bhanu, M. (2016) 'Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach', *International Journal of Big Data Intelligence*, Vol. 3, pp.145–153.

Xiao, Z., Song, W. and Chen, Q. (2013) 'Dynamic resource allocation using virtual machines for cloud computing environment', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, No. 6, pp.1107–1117.

Ye, T., Guangtao, X., Shiyou, Q. and Minglu, L. (2017) 'An auto-scaling framework for containerized elastic applications', *Proceedings of the 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pp.422–430.

Website

1 ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html