TWO-STEP AHEAD PREDICTION FOR ELASTIC CLOUD CONTROL SYSTEM

PREPARED BY: DESSIE TEKA

ADVISOR: SURAFEL LEMMA (PHD)

A Thesis submitted to

School of Electrical and Computer Engineering

Addis Ababa Institute of Technology

in

Partial Fulfillment of the Requirements for the

Degree of Master of Science in Telecommunication Engineering

Addis Ababa University

Addis Ababa, Ethiopia

February 05, 2020

Abstract

Elasticity property of cloud computing enables dynamic provisioning of computing resources in order to match changes in application demand at run time. Cloud elasticity solution acquires or releases cloud resources automatically using either proactive or reactive mechanism. Hybrid of the two approaches has been proposed in previous researches to take their advantages and mitigate their drawbacks; however, proposed approaches do not address proactive model inaccuracy before resource deployment and after system load behavior changes. In this research, threshold-based deviation between two-step ahead recursive predictions and observed values is proposed for decision making before scaling actions in combining proactive and reactive models. In addition, Stacked Long-Short Term Memory Recurrent Neural Network has been implemented as predictive model that can learn linear and nonlinear relationships of time series. The predictive model is retrained automatically with new observations at run time to adapt system load changes. To evaluate the proposed approach, experiments are conducted using CloudSim Plus with real system CPU usage. Empirical analysis of experimental simulation revealed significance of proposed solution in alleviating proactive model inaccuracy, deployment delay and oscillation to improve utilization efficiency and application performance. Moreover, performance of online retraining demonstrated that predictive model can learn new changes at run time to enrich proactive mechanism.

Keywords

Cloud Computing, Cloud Elasticity, Recurrent Neural Network, Long-Short Term Memory

Declaration

I, the undersigned, declare that the thesis comprises my own work in compliance with internationally accepted practices; I have fully acknowledged and referred all materials used in this thesis work.

February 2020

Dessie Teka _____

Name Signature

Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

This is to certify that the thesis prepared by Dessie Teka, entitled with *Two-step Ahead Prediction for Elastic Cloud Control System* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Telecommunication Engineering complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

Examiner 1	_____	Signature _____	Date _____
Examiner 2	_____	Signature _____	Date _____
Adviser	Surafel Lemma (PHD)	Signature _____	Date _____
Director of Post-graduate Program	_____	Signature _____	Date _____

_____

Dean, School of Electrical and Computer Engineering

Acknowledgments

First and foremost, praises and thanks to God, the Alpha and Omega, for His everlasting mercy and compassion besides allowing and helping me to complete this study!

I would like to express my special appreciation and thanks to my adviser Dr. Surafel Lemma, for his priceless advice, comments, guidance and encouragement. I also would like to express my deepest gratitude to my brothers Birhane W/Gebriel, Mesfin Alemu and Seid Degu, and all my friends for their invaluable comments, suggestions, encouragement, direct and indirect support.

My special thanks to all staff members in School of Electrical and Computer Engineering at AAiT for providing such a conducive environment during my study. I also would like to thank ethio telecom for giving me the opportunity to attend my postgraduate program in Addis Ababa University.

Last but not least, I want to express my deepest gratitude to my beloved Sisters, Brothers and all my families for their love, support and patience.

Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
ARIMA	Auto-Regressive Integrated Moving Average
ARMA	Auto-Regressive Moving Average
BPTT	Back Propagation Through Time
CC	Cloud Computing
CPU	Central Processing Unit
ECCS	Elastic Cloud Control System
FNN	Feed forward Neural Network
IaaS	Infrastructure as a Service
IT	Information Technology
KVM	Kernel Virtual Machine
LSTM	Long-Short Term Memory
MAPE-K	Monitor, Analyze, Plan and Execute with Knowledge-based system
MASE	Mean Absolute Scaled Error
MSE	Mean Square Error
NIST	National Institute of Standards and Technology
NN	Neural Network
OS	Operating System

PaaS	Platform as a Service
PM	Physical Machine
QoS	Quality of Service
RAM	Random Access Memory
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RT	Response Time
SaaS	Service as a Service
SARIMA	Seasonal Auto-Regressive Integrated Moving Average
SLO	Service Level Objective
SLSTM	Stacked LSTM
SVR	Support Vector Regression
TCP	Transport Control Protocol
vCPU	Virtual Central Processing Unit
VM	Virtual Machine
VMM	Virtual Machine Monitor

1 Introduction

Cloud computing (CC) is a rapidly growing Information Technology(IT) model that allows to deliver or acquire computing services on the basis of pay-as-you-go pricing model [1, 2, 3]. Cloud technology brings a paradigm shift in resource provisioning to access reliable, scalable and inexpensive computing resources over the Internet [3, 4]. Unlike in traditional computing model, users can purchase IT services from cloud providers instead of building their own IT infrastructures [2, 5]. Computing resources such as compute, storage and network in cloud datacenter can be shared between multiple users or applications by pooling physical machines into virtual resources [3, 6]. CC not only avoids investment cost of software applications and hardware devices from end-users' perspective but also maximizes resource efficiency and profitability from the perspective of infrastructure providers [7, 8, 9].

Currently, cloud technology has played crucial role in the development of enterprises and IT industries [5, 10]. Cloud computing is able to partition hardware capabilities into virtual resources to manage resource efficiency keeping performance requirements. According to Gartner: "*Cloud computing is a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies.*" [11]. In this regard, scalability and elasticity property leverages cloud computing ability to deliver flexible and on demand IT resources [1, 12]. Hence, cloud systems rely on elasticity solution to provision computing resources according the dynamic load variations of app [3, 13]. Cloud elasticity solution can be either horizontal or vertical to adjust computing resources automatically when application load runs beyond or below planned capacity [7, 9]. Horizontal elasticity is the automation of adding or releasing virtual machines (VMs) while vertical elasticity auto-scales computing resources such as CPU, RAM and storage. However, decision making and capacity planning are challenging tasks in modeling cloud elasticity solution [8]. With the growth of cloud computing technology, the need for efficient and autonomic provisioning of computing resources requires continual investigation of elasticity solution in cloud environment.

1.1 Motivation

In traditional computing technologies, server side applications require capacity planning in order to implement reliable and cost effective services. However, applications demand may not match with static capacity of planned resources during system operation due to dynamic variation in application load behavior. Unbalanced planning of computing resources to application requirement results in either over provisioning or under provisioning. This effect will bring low quality of application service or inefficient resource utilization with high power consumption.

The desire for IT service model to achieve better application performance with efficient resource utilization drives different enterprises and IT industries to adopt cloud computing technology [1, 5]. Cloud computing is a forward-looking IT service delivery model to harness solutions for challenges of traditional computing. Cloud technology improves IT service delivery, maximizes resource utilization efficiency, enhances massive innovations and increases profitability [5]. However, lack of system to provision IT resources in cloud environment with constantly changing demand automatically results in inefficient resource utilization and violates performance requirement. Thus, cloud systems require effective and efficient elasticity controlling mechanism for performance analysis, scaling decision and capacity planning.

1.2 Problem Statement

Modeling cloud elasticity solution requires either reactive or proactive mechanisms to manage on demand provisioning of computing resources [7, 12]. However, newly allocated resources cannot be ready on time in reactive elasticity to handle application demand due to reconfiguration delay [14]. The deployment process requires additional time to prepare and startup VM instances in horizontal cloud elasticity [7, 15, 16]. VM startup delay and oscillation problem in reactive elasticity violate user or application performance requirements [15]. Consequently, proactive elasticity mechanisms have been studied in previous research works to mitigate the time delay for instantaneous deployment of VM instances with predictive techniques [14]. In spite of allocating computing resources before actual load is observed, proactive elasticity cannot predict unexpected load in the prediction control interval [7].

To get the best of reactive and proactive approaches and address their limitations, investigating hybrid of the two approaches is proposed by several researches [8, 12, 14]. The combined approach minimizes VM deployment delay, prediction inaccuracy and oscillation problems [17, 18]. However, the proposed approaches use linear and nonlinear techniques in parallel and do not address proactive model inaccuracy before resource deployment and after system load changes from the original pattern. In line with previous works, this research investigates horizontal cloud elasticity using time series predictive technique in proactive model combined with reactive method. We hypothesize that threshold-based deviation between two-step ahead recursive predictions and run time observations improves proactive model accuracy and oscillation problem in the combined approach. Online retraining of predictive technique enables to adapt new changes and improve prediction performance over time.

1.3 Objectives

General objective:

- The purpose of this study is to model Elastic Cloud Control System (ECCS) using predictive technique that supports linear and nonlinear time series combined with reactive method for horizontal cloud elasticity.

Specific objectives:

- Model and develop a predictive method with online retraining mechanism for both linear and nonlinear time series prediction
- Obtain trained model with its parameters using training and testing for online prediction in cloud elasticity
- Propose combining mechanism for proactive and reactive elasticity policies to address proactive inaccuracy before resource deployment
- Develop Elasticity Controller to coordinate and manage elasticity processes of the combined approach in ECCS
- Prepare simulation based cloud environment controlled by ECCS to conduct experiments

- Evaluate experimental results to validate performance of the proposed ECCS approach

1.4 Scope

The scope of this research is to model and develop controlling mechanism for horizontal cloud elasticity. The proposed solution combines proactive and reactive approach to manage dynamic provisioning of virtual machines clustered by load balancing method. However, the solution does not include strategies for load balancing and virtual machine allocation. The control system employs machine learning predictive technique in proactive model for time series analysis of CPU Utilization. The predictive model performs online prediction and online retraining. We have also used queue network theory only to calculate utilization threshold without implementing the queue network model for request analysis.

1.5 Methodology

In order to achieve the objectives of this research, we have started by looking for predictive techniques and developing system model for Elastic Cloud Control System. Modeling a predictive technique includes collecting time series dataset from real system, preparing the dataset, building and training predictive model. Trained model parameters are saved in file for further use in cloud elasticity actions. In this research, Python and Java programming languages are employed to develop and implement Elastic Cloud Control System. Predictive model for online predictions and retraining is developed in python while Elasticity Controller in Java.

The Elasticity Controller and predictive model are deployed to work on the basis of client server architecture. The controller is implemented as a client to communicate with predictive model and initiate two-step ahead prediction and online retraining. The predictive model uploads trained model from file for online prediction and retraining. Predicted data values are stored in data store (file) to utilized by the controller during elasticity actions. In addition, predictive model is retrained with historical observations to improve prediction accuracy at run time.

Experiments are conducted to empirically analyze and validate the proposed cloud elasticity solution. Simulation is used to experiment the proposed Elastic Cloud Control System. CloudSim Plus simulator tool is deployed and integrated with Elastic Cloud Con-

trol System. Finally, we used Python to analyze and visualize results of the experimental simulation.

1.6 Significance of The Study

The proposed cloud elasticity solution enables provisioning of IT resource capabilities in cloud environment automatically depending on application system load. In addition, online retraining of predictive model allows elasticity solution to adapt system load changes to enhance decision making in proactive cloud elasticity. Therefore, this study can be useful for small and large enterprises, telecommunications and IT industries which are utilizing and adopting cloud computing technologies.

1.7 Thesis Organization

The rest of this document is organized as follows: Chapter 2 introduces an overview of cloud computing, cloud elasticity types and models, basics of Artificial Neural Network and concept of autonomic computing. Chapter 3 presents literature review on cloud elasticity mechanisms, the relevance of Long-Short Term memory(LSTM) in time series analysis, and related works. The proposed cloud elasticity solution, and its components and functionalities are discussed in Chapter 4. Chapter 5 explains experimental setup, implementation and results. Finally, conclusion, future works and limitations of the study are presented in Chapter 6.

2 Background

2.1 Cloud Computing

Cloud computing is a technology model to provide IT services over the Internet combining distributed and utility computing with virtualization technology [2, 10]. Various definitions of CC are provided by different IT industries and researchers [4, 10]. National Institute of Standards and Technology (NIST) stated broadly adopted definition of CC as *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [6]. NIST cloud model consists of three cloud service delivery models, four cloud deployment models and five essential characteristics (see Figure 2.1).

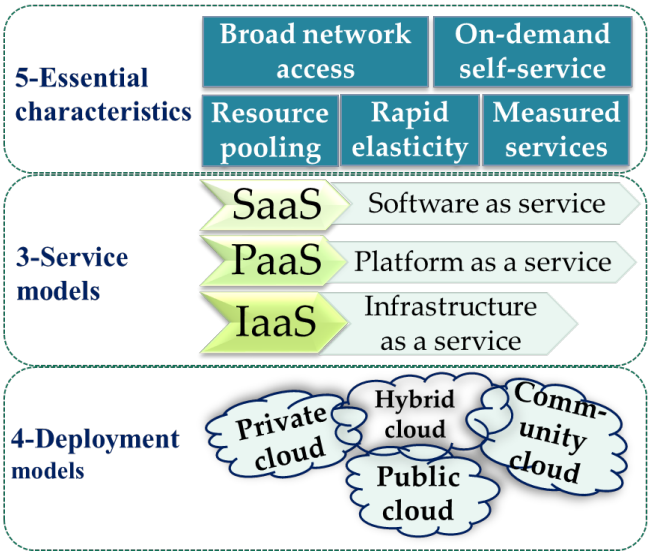

Figure 2.1: NIST cloud computing model

Cloud service models are software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) which are categorized based on type of service delivery [2, 6]. SaaS allows users to access ready-to-use application and storage services over the internet. Clients get direct access to the services using end-user devices without designing, developing or hosting applications. While PaaS enables users to rent platforms and deploy their own software applications on virtualized computing environment. IaaS provides

complete virtualized infrastructures to users in the form of virtual machines, storage and networks to implement both SaaS and PaaS services. As a result, cloud business model incorporates cloud provider, application provider and consumer depending on the service model.

Datacenter infrastructures for the three service delivery models can be deployed and provisioned in either public, private, community or hybrid models based on their locations [6]. In public cloud model, computing resources are provisioned to open use on pay-per-use model similar to utility services such as water and electricity. Single organization owns the cloud infrastructures in private cloud model. Infrastructures in cloud datacenter can be also provisioned to specific community from different organizations having related organizational units in the case of community cloud model. Hybrid cloud model comprises combination of two or more of private, public or community cloud models. Therefore, deployment model identifies cloud computing environment either to implement or access cloud services.

Cloud computing technology offers main features to provision and meter computing resources automatically [2, 6]. The five essential characteristics of cloud computing:

- **Broad network access**: cloud computing resources can be accessed over the network using different types of end-users devices such as smart phones, tablets and computers.
- **On-demand self-service**: computing resources can be provisioned based on user requirement without providers intervention.
- **Resource pooling**: physical and virtual computing resources are pooled together to be shared by multiple users.
- **Rapid elasticity**: computing resources can be acquired and released automatically according to the dynamic variation of user or application demand.
- **Measured services**: provisioned computing resources by users are managed to enable pay-per-use basis.

CC includes additional characteristics besides its five essential characteristics such as service oriented, loose coupling, fault tolerant and virtualization [19]. Virtualization tech-

nology is the main enabler of cloud computing which hides the underlying complexity of physical servers to create a shared pools of logical servers called virtual machines (VMs) [10, 12]. Each VMs run their own operating systems (OSs) as if the OSs are running directly on physical servers. Virtualization isolates applications running on the same physical server by multiplexing its computing capabilities into different virtualized resources such as VM, number of virtual CPU (vCPU), amount of RAM or storage [15].

Figure 2.2 depicts the building blocks of cloud datacenter that incorporates infrastructure layer, virtual layer and application layer. Infrastructure layer hosts physical machines such as servers, storage arrays, and network devices in physical layer. The virtualization layer resides between application and hardware layers which is managed by Virtual machine monitor (VMM) or hypervisor. VMM is a software layer responsible for abstracting and sharing the resources of physical machines among virtual resources. Different types of hypervisors can be used to abstract physical infrastructures which are either open source or commercial [10]. For example, Xen and KVM are open source while VMWare and hyper-V are commercial hypervisors [12, 20]. Application layer is implemented on top of virtualization layer to install operating systems (OSs) and user applications.

Cloud operating systems such as OpenStack, OpenNebula and CloudStack monitor and

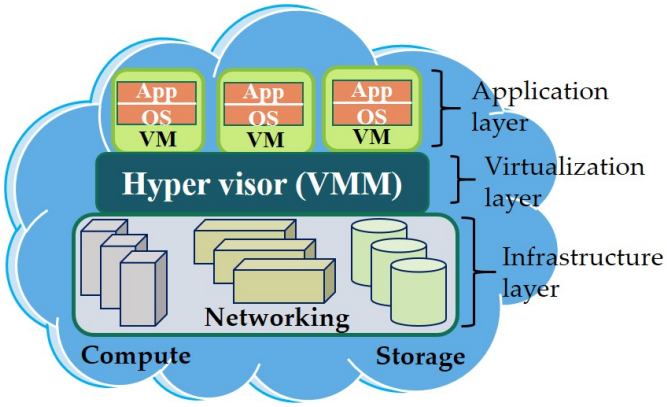

Figure 2.2: Cloud system implementation layers

orchestrate cloud environments on top of virtualization [20]. OpenStack and CloudStack are open source platforms to manage cloud resources [21, 22]. The monitoring functionality of cloud operating system supports to automate scalability and elasticity [20]. Qu et al. [8] described that elasticity is a feature of cloud computing to automate auto-scaling

processes while auto-scaling is a mechanism to acquire or release computing resources dynamically.

2.2 Cloud Elasticity

Elasticity is principal property of cloud computing to enable flexible and autonomic provisioning of computing resources to meet the dynamic requirements of applications or users [7, 8]. Herbst et al. [13] surveyed various definitions of elasticity to propose comprehensible definitions of elasticity in CC. According to the authors, *"Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible"*. Computing resources of elastic cloud system expands and contracts automatically in order to adapt current application demand to avoid under utilization or over-utilization of resources. Thus, cloud elasticity is responsible to scale optimum number of virtual infrastructures with the aim of resource auto-scaling, application performance, service availability and cost reduction [12]. In addition, optimal computing resources reduce energy consumption and CO_2 footprint while maximizing economical profit [5]. Elasticity solutions can be categorized into elasticity types depending on virtual resources and elasticity model grouped based on elasticity mechanisms [12].

2.2.1 Elasticity Types

Cloud elasticity types can be either horizontal or vertical which represents the type of computing resources to be implemented during auto-scaling action. Horizontal cloud elasticity acquires (scales out) or releases (scales in) VM instances on the shared pool physical machines [8, 15] whereas vertical elasticity scales up or scales down computing capabilities of a single VM instance [9]. Figure 2.3 depicts a generic architecture of elasticity that comprises cloud systems with cloud elasticity management. End-users or client applications generate requests and dispatch those requests to be processed by computing resources in the datacenter. In horizontal cloud elasticity, number of VM servers are clustered using load balancing method to aggregate VMs in cluster as a single unit; hence, the load balancer is the entry point to receive end-users requests and distribute them to available VMs in the cluster [23, 24]. On the other hand, vertical elasticity

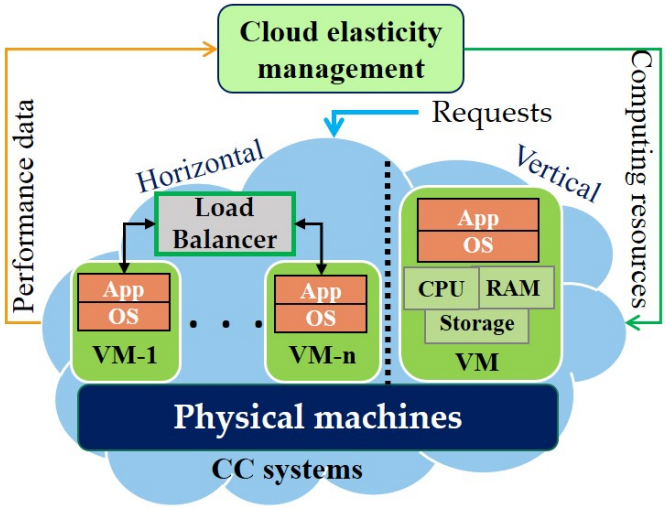

Figure 2.3: Elastic cloud generic architecture

increases or decreases computing capabilities of VM instances such as number of vCPU cores, amount of memory or storage within a single VM server [9, 12]. Depending on the type of elasticity, computing resources can be added or removed dynamically based on workload fluctuations to guarantee the application performance and optimal resource utilization.

Even though vertical elasticity provides fine grained solution to increase or decrease VM resources, it is limited to single physical server [12]. Vertical elasticity may require VM instance migration with from one physical machine to another with higher capacity when resources are not sufficient to support incoming load. Thus, vertical cloud elasticity is restricted to small enterprises with single high performance physical server. However, horizontal elasticity can span large number of physical servers to allocate multiple VM instances. Elasticity models, which are also referred to as elasticity mechanisms or policies, are required to model horizontal or vertical cloud elasticity solutions [7, 12].

2.2.2 Elasticity Models

Elasticity models are mechanisms for decision making and resource auto-scaling that can be either in reactive or proactive approach [8, 12]. Reactive mechanism relies on observed performance metrics to provision cloud resources [25]. For instance, add computing resources if observation is above upper threshold or remove computing resources if observation is below lower threshold for a specific period of time. In reactive method,

performance of application service is degraded due to VM deployment delay depending on the type of operating system (OS) used and location of VM images [7, 15]. The deployment delay is the time taken by VMM to allocate new VMs ready for accepting and processing application requests. Rule-based threshold in reactive model introduces a trade-off between over provisioning and performance degradation for scaling-out action while scaling-in is prone to oscillation [8]. Oscillation problem causes frequent scale-in/out operations for short period of time in successive elasticity control interval that degrades application performance [8, 23]. Whereas, proactive elasticity can mitigate provisioning lags and oscillations that are major challenges of reactive elasticity [23, 15]. Proactive approach uses predictive techniques to forecast future values of performance metrics in order to provision and allocate computing resources ahead of actual observations of the predicted values.

Moreover, auto-scaling techniques in elasticity models can be grouped into five classes: rule-based threshold, control theory, queuing theory, reinforcement learning, and time series analysis [23, 18, 17]. Rule-based threshold is mainly used in reactive model while time series analysis is used in proactive model [12]. Queuing theory, control theory and reinforcement learning can be used in both approaches. Each technique have their own advantages and draw backs to build elasticity solution. In control theory, setting gain parameters can be a difficult task while queuing method are rigid due to fixed parameters [23]. In addition, complexity of algorithmic process limits the practical use of control theory and machine learning methods [26]. Reinforcement learning takes long time to converge. Deciding threshold values in rule-based strategy is a difficult task to match with application performance. Even though time series is not viable in accuracy over time due to the dynamic nature of cloud environment, it is popular and effective in proactive cloud elasticity solution [7, 23, 27]. The accuracy depends on the selected prediction model. Hence, proactive model can be built using predictive techniques for time series analysis of collected performance data from cloud systems.

2.3 Time Series Prediction

Time series is a collection of an orderly sequence of historical observations with temporal dependence in specific interval. The order complexity or time dependency between the

data makes time series prediction more difficult that requires proper selection of predictive techniques [28]. Time series prediction is analysis of timely ordered sequential dataset to understand its patterns in the past and estimate expected future values in advance [29, 30, 31]. The popular methods for time series analysis are statistical linear regression and machine learning techniques [30].

Auto-regressive Integrated Moving Average (ARIMA) model is a commonly used statistical model that follows Box-Jenkins method if observations have linear relationships with static temporal dependence [14, 31]. However, it does not handle observations with nonlinear relationship or different set of lags [30, 31]. In addition, ARIMA model assumes residual noises as random component; but random error can also exist in other components in time series decomposition [17]. Recently, machine learning techniques with their growing prediction accuracy outperforms classical models in complex time series analysis [32]. For instance, Artificial Neural Network (ANN) in machine learning can support linear or nonlinear relationships, different set of lag observation and longtime forecasting horizon [32]. ANN algorithms can identify complicated patterns of time series data and predict future values [33]. ANN is also preferable for model retraining at run time to adapt changes in dynamic systems using new observations [34].

2.4 Artificial Neural Network

Artificial Neural Network works on the basis of biological neural networks inspired by learning and processing way of human brain [27]. Neurons are basic components of ANN which can be also named as units or nodes. Artificial neurons or nodes are responsible to perform mathematical operations analogues to operations in biological neurons [30]. Nodes are also connected in layers to build ANN or simply Neural Network (NN) architecture. Fully arranged layer in NN comprises input layer, hidden layer and output layer. Neurons between layers are interconnected using weight values with a single bias value in each layer along the network structure [35]. The network learning process will be either supervised learning or unsupervised learning. Supervised learning uses labeled training data as inputs along with reference outputs [32, 35]. Then, weights are updated based on errors calculated between reference outputs and predicted outputs using optimization algorithm during training process. Unsupervised learning approach explores hidden structures

in training data without labeling the data in advance for fast learning [29]. The learning process can be feed forward or recurrent based. ANN algorithms are commonly used in different domains such as economics, finance and health [23].

2.4.1 Feed forward Neural Network

Feed forward Neueal Network (FNN) is also called Multilayer Perceptron (MLP) [36]. FNN performs computational processes starting from input information to produce an output without feedback connection.

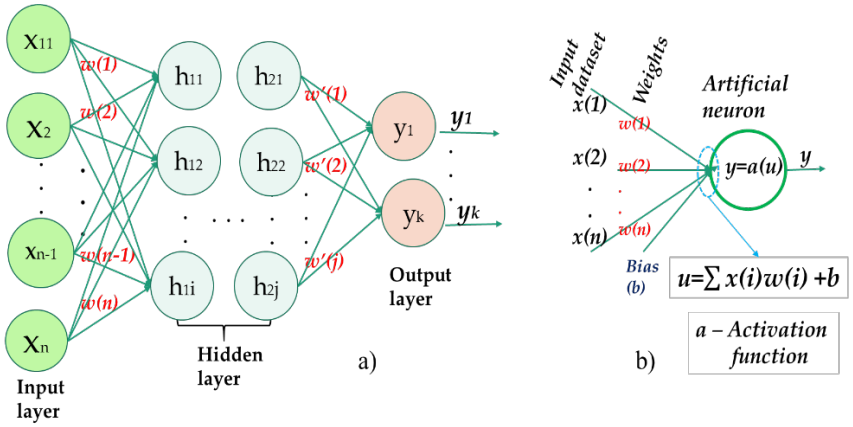

Figure 2.4: (a) FNN layers, (b) Artificial neuron

Where x – inputs data, w – NN assigned weights, h - hidden layer data, u – weighted sum of input data, $a(u)$ – activation function, y – output of artificial neuron and i, j, n – number of neurons.

Figure 2.4 (a) depicts Feed forward Neural Network architecture where input layer receives original data and transfer it to the first hidden layer. In Figure 2.4 (b), each neuron in the hidden layer receive weighted sum of input data and process on it using activation function such as sigmoid or hyperbolic tangent function. The activation function introduces non-linearity on the input data from the previous layer before passing its output to the next layer [29]. Finally, output layer apply aggregation and activation function on received data from last hidden layer to provide final NN output. The learning process implements gradient algorithm to train hidden layer neurons [36]. However, NN may incur either exploding gradient problem when weights are very large or vanishing gradient problem when weights are too small (less than 1) [36]. Even though deep FNN avoids gradient problems using different weights in each time step, it cannot hold long-term temporal dependencies in sequential data.

2.4.2 Recurrent Neural Network

Recent developments of ANN encourage to analyze sequential data such as time series forecasting, image processing and language translation [36]. Unlike Feed forward Neural Network, Recurrent Neural Network (RNN) uses recurrent connections between hidden layer neurons to predict sequential data (See Figure 2.5). Output of node at each time step will be an input to another node in next time step in the same hidden layer [36]. Thus, RNN model adds memory block to capture periodic components of long-term dependencies in sequences [33, 37, 34]. The memory block stores output of RNN units at time t-1 which will be feedback to another at time t in the same hidden layer. RNN training uses back

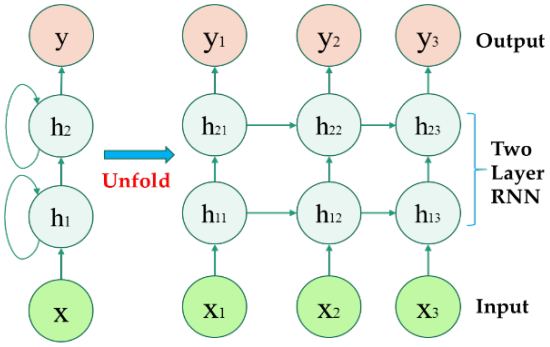

Figure 2.5: Two layers of Recurrent NN [36]

Where x – inputs data, h - hidden layer data, y – output of RNN

propagation through time (BPTT) learning algorithm which is a gradient based technique. In RNN, weights are multiplied many times over several time steps that causes exploding or vanishing gradient problem during training process [33, 36]. Hence, challenges of learning long-term dependencies are also main problems in traditional RNN. Long-Short Term Memory (LSTM) is special type of avant-garde RNN model to resolve vanishing or exploding gradient problems of classical RNNs [34, 38]. Classical RNN memory blocks overwrite their contents in each training time step whereas LSTM memory blocks contain memory cells to store temporal state of its network [37]. Besides, LSTM block consists of different nonlinear gating units to regulate the in and out flow of information in its memory cell [39]. Figure 2.6 shows input layer, three layers of LSTM network blocks in the hidden layer and output layer. When hidden layer comprises of multiple LSTM is said to be Stacked LSTM (SLST) or Deep LSTM [40]. Internal gates in the LSTM block enable whether to update, keep or reset memory cell state [31]. The three internal gates of LSTM

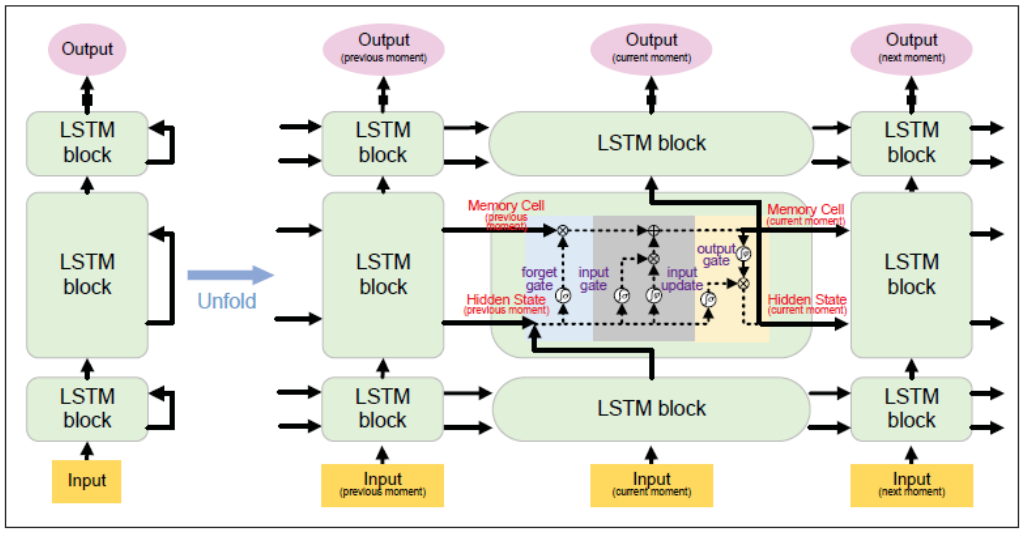

Figure 2.6: Three layers of LSTM network blocks [31]

block are input gate, output gate and forget gate [37]. Input gate controls the flow of new information to update the memory cell while output gate controls the flow of LSTM block information into the rest of LSTM network block. Forget gate resets the old information of memory cell during training process [31, 39]. The internal gates and memory cell in LSTM block enables to comprehend long term time dependencies [33]. Therefore, LSTM RNN is more efficient than basic RNNs or traditional NNs for sequential data analysis such as time series data, language, audio and video processing [37].

2.5 Autonomic Computing

Cloud utilizes autonomic computing to facilitate on-demand and flexible provisioning of computing resources. According to [41] "*Autonomic computing system senses its operating environment, models its behavior in that environment, and takes action to change the environment or its behavior*". Originally, IBM proposed autonomic computing frame work that comprises Monitoring, Analysis, Planning and Execution phases with knowledge-based system called MAPE-K loop architecture to automate self-management systems [41, 42]. Figure 2.7 depicts the four phases of MAPE loop architecture with a knowledge-based system to control automatic cloud resource provisioning. Monitoring phase of MAPE loop gathers performance data that will be consumed in the analysis phase for decision making and capacity estimation in planning phase. Execution phase of MAPE loop applies reconfiguration action. The knowledge-based system usually

maintains configurations and analytical information about managed systems shared by all MAPE loop phases.

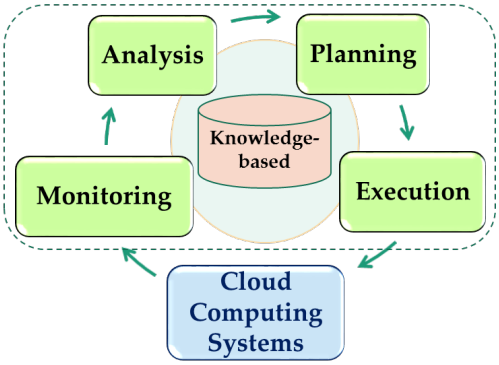

Figure 2.7: Process flow in MAPE-K loop

The introduction of autonomic computing to cloud elasticity solution brings significant benefits in order to manage computing resources automatically [43]. Autonomous system monitors its environment to adapt with dynamic variations of external conditions by adjusting its attributes without intervention. Cloud elasticity managers are categorized in the domain of control systems according to their capability to change virtual components of cloud computing dynamically [8]. Hence, autonomic computing can be applied in elasticity solution to facilitate elasticity processes phase by phase. The MAPE-K loop organizes system monitoring, historical data analysis, capacity planning and resource reconfiguration [8, 23, 24]. Elasticity techniques such as time series analysis and queuing theory can be used in MAPE-K loop analysis phase while control theory and reinforcement learning can be used in both analysis and planning phase [22, 42]. Queuing theory can also be used for resource planning while rule-based thresholds are used for decision making in planning phase of MAPE-K loop.

3 Literature Review

Several researches have been conducted to propose elasticity solutions in order to manage computing resource provisioning in cloud environment automatically. Many of the previous works are based on either reactive or proactive approaches [9, 12, 23]. Recently, researches on elasticity solution combine both methods. In addition to cloud elasticity mechanisms, Maurer et al. in [44] also studied the relevance of autonomic computing for cloud infrastructure self-management.

3.1 Reactive and Proactive Elasticity

Reactive mechanisms depend on real time observations for decision making in auto-scaling cloud resources [27]. The implementation of the decision (e.g., starting a new VM) in reactive approach has a delay. Humphrey et al. [16] examined the effect of VM startup time in cloud auto-scaling across public providers with different characteristics such as OS type, image size and instance type. The paper revealed that VM startup time becomes longer as image size of instances increase. Bikas et al. [7] also described that application performance will be degraded due to the delay in resource readiness when VM resources are provisioned in reactive cloud elasticity.

Consequently, researchers looked into proactive modeling techniques to resolve reactive auto-scaling challenges. The authors in [27] argued that predictive model improves resource efficiency and application performance. Huang et al. [22] proposed proactive auto-scaling that uses queuing model for workload analysis and resource planning to improve performance and efficiency. The authors also proposed an algorithm to optimally allocate resources between horizontal and vertical scaling. However, queuing model is inflexible in analyzing frequently changing application or workload [23]. Roy et al. [25] proposed auto-scaling solution for cloud computing using proactive techniques to improve application quality of service (QoS) and reduce costs. They used second order Auto-Regressive Moving Average (ARMA) model for workload prediction and look-ahead optimizations to provision computing resources. Moreover, Calheiros [14] used ARIMA model. Both ARMA and ARIMA forecasting models are working on time series data having linear relationship. The algorithm for ARIMA model relies on linear functions

which are suitable for time series with linear dependence [30]. Borkowski et.al [27] proposed predictive approach using ANN to predict historical data for resources provisioning in cloud environment. In this research, the proposed solution utilizes combined proactive and reactive elasticity policies based on autonomic computing approach to devise Elastic Cloud Control System.

3.2 LSTM RNN in Time series analysis

RNN techniques are able to learn trends which are in increasing or decreasing level of time series as well as seasonality or consistently repeating patterns of time series over time [27] . Researchers compared LSTM RNN technique with statistical ARIMA model for time series prediction. Deepak and Enda [30] compared LSTM Network with statistical ARIMA model for time series analysis using CPU utilization of servers in data centers. The authors conducted empirical evaluation to investigate LSTM and ARIMA model on the same dataset of CPU usage generated every 5-minutes interval. The observations indicated that ARIMA model could predict short term time series data but unable to predict long term time series data. Finally, results of the study showed that LSTM model learns long term in addition to linear and nonlinear time series better than ARIMA model. Hua et al. [31] investigated performance of LSTM RNN technique to predict long term dependencies of time series using traffic and user mobility in telecommunication networks. They proposed random connectivity to LSTM network to minimize computational complexity. The authors compared their proposed LSTM model with ARIMA, Support Vector Regression (SVR), FNN and other variants of LSTM models.

LSTM can solve time series problems which is able to capture long term temporal dependencies as well as handle order complexities between observations for better predictions [33, 31]. LSTM predicts time series data better than classical NN, SVR and statistical prediction models such as ARIMA model [30, 31]. In this thesis, proactive model uses LSTM RNN having forget gate with bias factor greater or equals to 1 in order to learn long term dependency besides linear and non-linear patterns of time series [45]. Then ECCS uses reactive method if proactive model fails to adjust VM deployment due to prediction inaccuracy. Moreover, LSTM RNN retrains with new generated time series in online manner to improve prediction accuracy.

3.3 Related Works

Liu et al. [17] proposed elasticity controller to provision distributed storage system (state-based service) in order to maintain quality of service (QoS) and reduce cost. The authors proposed cloud elasticity controller using combined proactive and reactive approach to overcome their limitations and take their advantages instead of using them individually. The proposed hybrid elasticity model monitors workload (arrival rate of reads and writes) on each VM for workload preprocessing. Prediction model analyzes and predicts future values at the beginning of each prediction window. Firstly, scaling decision and action are performed using predicted workload within prediction window and VM instances are reconfigured. Then, predicted and observed workloads are compared at the end of each prediction window in the reactive method to adjust incorrect scaling operations. But this causes oscillation when proactive model makes scaling decision based on inaccurate prediction to implement computing resources.

Wiener filter linear technique has been used to estimate one step ahead prediction in proactive model. This method is optimal to minimize Mean Square Error (MSE) for time series having smaller percentage of noise. However, linear filtering method is poor for noisy time series and real application are not linear in practice [30]. The authors suggested online training of predictive model for combined elasticity approach that can tune model parameters and improve prediction accuracy for scaling decision making.

Herbst et al. [18] also proposed cloud auto-scaler using combined Seasonal Auto-regressive Integrated Moving Average (SARIMA) and trigonometric, Box-cox transformed, ARMA errors using Trend and Seasonal components (tBATS) models for multistep ahead time series forecasting. The aim of this approach is to reduce risk of single proactive method and minimize reaction times. Reactive method is also used if predictive model forecasts inaccurately in proactive method. In proactive model, SARIMA and tBATS models are not executed at same time instead predicted vales are selected at a time either from forecasting results based on meta-learning approach automatically. SARIMA is linear prediction model while tBATS can predict nonlinear time series.

The proposed model monitors request arrival rate and stored in time series data storage. The forecast component carried out multiple forecasts ahead of time. Forecasting can be

initiated depending on Mean Absolute Scaled Error (MASE) which is also used to drop or accept proactive or reactive method in each scaling control time. Thus, proactive model is used for scaling decision and action if MASE value is less than some threshold otherwise the controller uses reactive method. MASE is computed with the assumption of taking this time observation as predicted value for the next time-step. To plan resource capacity, system utilization and application response time are calculated using predicted or observed arrival rates based on queue network theory. The calculated utilization and response time are compared with reference values to estimate number of virtual machines.

However, the proposed approaches do not address proactive model inaccuracy before resource scaling operations and adaptability to the changes in system load behavior in the long-run. Besides, predictive techniques are linear in [17] or linear and nonlinear but utilized separately in [18]. In this research, combined proactive and reactive approach is proposed using Stacked LSTM predictive model for two-step ahead prediction to handle proactive model inaccuracy before scaling actions. SLSTM model can learn time series with linear and nonlinear relationships. In addition, predictive model is developed to perform online retraining automatically to adapt system load changes.

4 Proposed Approach

4.1 System Model

Cloud elasticity with autonomic computing encompasses mechanisms to monitor system load, analyze for scaling decision, and plan computing resource capacity [23]. Elasticity solution is able to collect system performance from cloud systems and apply auto-scaling action. Hence, controlling system is required to automate the dynamic nature of cloud environment using reactive and proactive mechanisms. In this research, two-step ahead prediction with run time observation to coordinate proactive and reactive elasticity and online retraining of predictive model are proposed to model Elastic Cloud Control System.

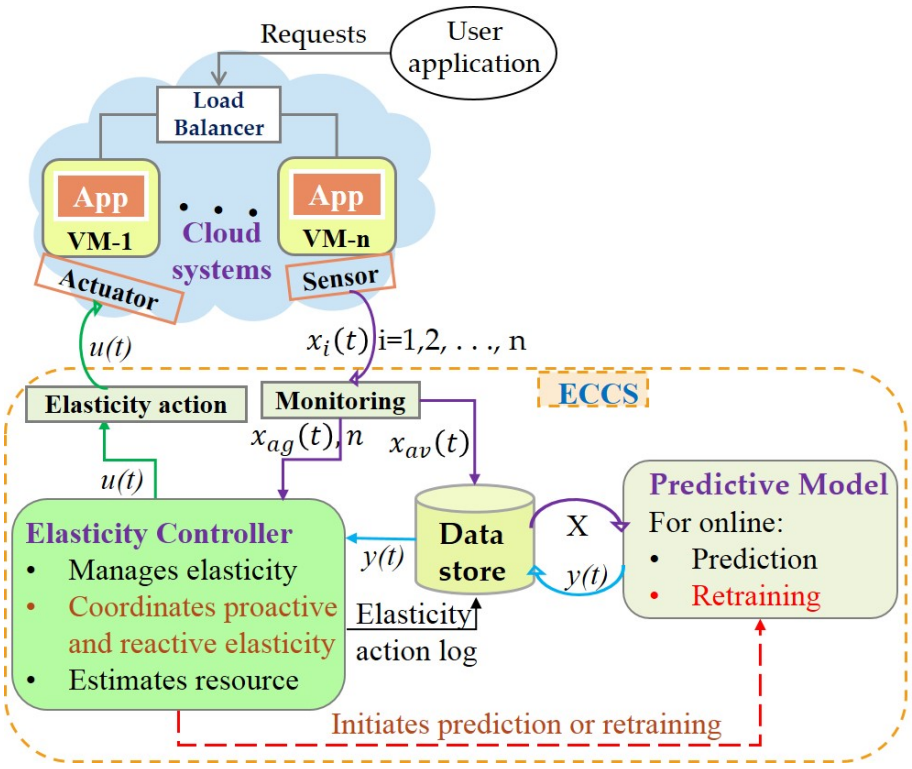

Figure 4.1: System model for proposed approach

Where $x_i(t)$ – observed CPU utilization of VM(i) at time t, n – number of running VMs, $x_{ag}(t) = \sum_{i=1}^{n} x_i$ – aggregated utilization, $x_{av}(t) = \frac{x_{ag}}{n}$ – average utilization, $X = \{x_{av}(t), x_{av}(t-1), x_{av}(t-2), ...\}$ – historical observations, $u(t)$ – number of estimated VMs, and $y(t)$ – predicted CPU utilization for time t, and App - Application.

Figure 4.1 depicts system model of the proposed solution which relies on combined

proactive and reactive approach for horizontal cloud elasticity based on MAPE-K loop architecture described in Section 2.5. The system model incorporates cloud systems which generate monitoring data in handling requests of user applications in addition to Elastic Cloud Control System. ECCS is integrated with cloud systems to gather monitored information and provision VMs automatically in order to alleviate resource over-utilization or under-utilization at run time. The main components of Elastic Cloud Control System are system load monitoring, data store, predictive model, Elasticity Controller and resource reconfiguration.

First, client applications or users are sending their requests to application running on clustered VMs via a load balancer that distributes the requests. VMs are monitored to collect actual observations of system load $x(t)$ from each VMs in constant monitoring time interval. CPU utilization load is used as a performance metric generated by applications running on virtual machines. Time series of CPU usage observations X are stored in the data store. Subsequently, predictive component anticipates future system load $y(t)$ by using the historical data X in the data store while Elasticity Controller manages time and scaling decision, coordinates reactive and proactive models, and estimate resource capacity $u(t)$. Elasticity Controller utilizes outputs from historical data analysis $y(t)$ and actual observations $x_{ag}(t)$ and $x_{av}(t)$ to make scaling decision and trigger elasticity action automatically. Additionally, online retraining enriches proactive model to adapt workload changes using time series data from real time observations X. Elasticity Controller works with predictive model following client-server architecture. Elasticity Controller is running as a client to initiate online prediction and retraining while proactive model is running as a server to handle the requested tasks. The next subsections describe different components of Elastic Cloud Control System.

4.1.1 Monitoring System and Data

Monitoring measures system load (performance metric) to enable cloud elasticity solution. Sensors or agents can be deployed within cloud computing environment to measure the system load [24, 46]. Monitoring system gathers measured system information of computing resources and running applications besides resource configuration from cloud systems. In this research, monitoring tool collects observations of CPU utilization from

each virtual machines continuously sampled at a constant monitoring time interval. In addition, monitoring tool aggregates total CPU utilization and calculates the average for analysis and decision making. The historical data of average CPU utilization and elasticity action log will be stored in data store. Data store can be a database system that supports to manipulate large amount of data.

Historical data is said to be time series data when it is ordered with respect to time sequentially [32]. Proactive model utilizes time series of average CPU utilization measured at two minute interval to forecast future values using predictive technique. Predicted values are stored in data store at each prediction control time that will be used by Elasticity Controller in next scaling control time.

4.1.2 Predictive model

Predictive model is responsible to anticipate future values of system load to make scaling decision ahead of time. We used Long Short Term Memory RNN to develop proactive model and predict CPU usage with similar reasons provided in [33]. The authors mentioned three key reasons to select LSTM model for system load forecasting. LSTM model can learn nonlinear time series, remember short-term and long-term dependencies and extract useful information from data. In addition, "*Long Short-Term Memory (LSTM) is able to solve many time series tasks unsolvable by feed-forward networks using fixed size time windows.*" [28].

LSTM block maintains a separate memory cell which can update and hold its state over time and convey information to the next time steps for long-short term prediction [31].

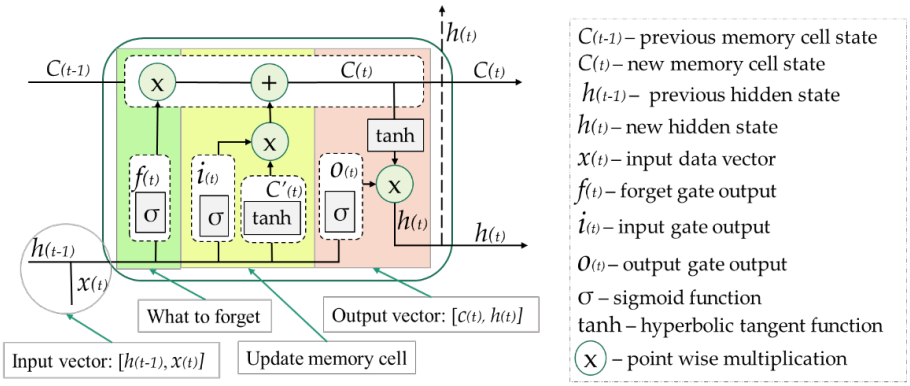

Figure 4.2: Flow of data in LSTM block [39]

Figure 4.2 shows the input-output flow of data in LSTM block where the previous memory cell state $C(t-1)$, previous hidden state $h(t-1)$ and $x(t)$ are inputs and new memory cell state $C(t)$ and $h(t)$ are outputs. LSTM block performs mathematical operations using hyperbolic tangent ($\tanh$) and sigmoid (σ) functions given in Equation 4.1 and 4.2 respectively to determine how much data to be stored in memory cell, update the memory cell and calculate output vector $[C(t), h(t)]$. Hyperbolic tangent function ($\tanh$) squashes input data (u) between an interval $[-1, 1]$ while sigmoid function (σ) gives between $[0, 1]$ [29].

$$\tanh = \frac{1 - e^{(-2u)}}{1 + e^{(2u)}} \tag{4.1}$$

$$\sigma = \frac{1}{1 + e^{(-u)}} \tag{4.2}$$

Mathematical equations of LSTM block are shown in Equation 4.3 [33, 39].

$$\begin{aligned} f(t) &= \sigma([x(t) * w_{fx}, h(t-1) * w_{fh}]) \\ i(t) &= \sigma([x(t) * w_{ix}, h(t-1) * w_{ih}]) \\ o(t) &= \sigma([x(t) * w_{ox}, h(t-1) * w_{oh}]) \\ C'(t) &= \tanh([x(t) * w_{ix}, h(t-1) * w_{ih}]) \\ C(t) &= f(t) * C(t-1) + i(t) * C'(t) \\ h(t) &= \tanh(C(t)) * o(t) \end{aligned} \tag{4.3}$$

Where w_{gx} and w_{gh} are weight values for input data($x(t)$) and previous hidden state ($h(t-1)$) in each gate $g = \{f, i, o\}$ and C' is an intermediate value to compute cell state.

Keras library implements the algorithms of LSTM network model given in Equation 4.3 which was written using Python programming language [37, 47]. In this thesis, Keras library with Theano back-end has been used to perform two-step ahead prediction and model retraining.

4.1.3 Elasticity Controller

Elasticity Controller in ECCS coordinates proactive and reactive elasticity, estimates resources, triggers elasticity actions, and initiates online prediction and retraining of predictive model. It continuously checks control time for scaling, prediction and retraining to manage elasticity processes in constant time intervals. Time series analysis of monitoring data in proactive model enables Elasticity Controller to make scaling decision and plan resource capacity. Proactive elasticity is accompanied by reactive elasticity in a consecutive proactive and reactive control time when predicted value deviates from actual value. The reactive elasticity uses real time observations collected by monitoring. The method used to combine proactive and reactive elasticity recommendations is described in Section 4.3.

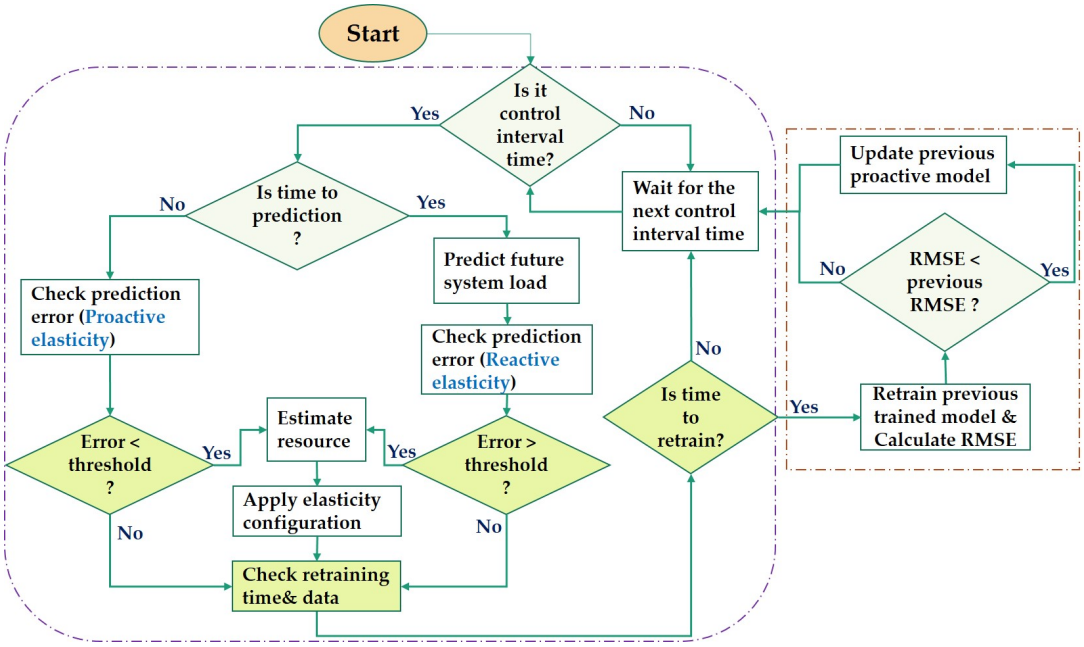

Figure 4.3: Elasticity Controller process flow

Figure 4.3 depicts Elasticity Controller process flow to organize and manage components of ECCS. Elastic Cloud Control System estimates number of VMs required for elasticity operation based on scaling decision. The scaling decision and resource estimation are performed using either observed or predicted values of average CPU utilization and standard thresholds similar to the work in [18].

Thresholds for scaling decisions are set usually based on datacenter trends; for instance, CPU utilization of 80% for upper threshold to scale-out (scale-up) and 20% for lower

threshold to scale-in (scale-down) resources [7, 26]. Moreover, service level objectives (SLO) can use thresholds of response time to guarantee application performance. Utilization can be associated with SLO response time with the concept of queue network theory [48]. In queue network theory, y^{th} percentile of response time can be calculated using Equation 4.4 for $M/M/1$ server.

$$M_{RT}(y) = \frac{T_s}{1-\rho} ln\Big(\frac{100}{100-y}\Big) \tag{4.4}$$

Where "$M_{RT}(y)$ – y^{th} percentile or value of y below which RT (response time) occurs y percent of the time, T_s – mean service time and ρ – utilization"

In this study, Elasticity Controller calculates utilization threshold based on Equation 4.4 using SLO response time and mean service time(T_s). The percentile of response time in Equation 4.4 does not include network transmission delay. By rearranging Equation 4.4 into Equation 4.5 and 4.6, percentile of response time can be translated into utilization. The equations are used to find the range of utilization thresholds satisfying the SLO response time requirements.

- To guarantee application performance, y^{th} percentile of response time should be less than upper threshold value of response time. Thus, we get upper utilization threshold (ρ_{upper}) from Equation 4.5.

$$M_{RT}(y) = \frac{T_s}{1-\rho_{upper}} ln\Big(\frac{100}{100-y}\Big) \leq upperSLO_{RT}$$

$$\rho_{upper} \leq 1 - \frac{T_s}{upperSLO_{RT}} ln\Big(\frac{100}{100-y}\Big) \tag{4.5}$$

- On the other hand, resources could be under utilized when y^{th} percentile of response time drops below the lower threshold value of response time. Thus, we get lower utilization threshold from Equation 4.6. To maximize efficiency, utilization should be greater than the lower threshold value (ρ_{lower}):

$$M_{RT}(y) = \frac{T_s}{1-\rho_{lower}} ln\Big(\frac{100}{100-y}\Big) \leq lowerSLO_{RT}$$

$$\rho_{lower} \leq 1 - \frac{T_s}{lowerSLO_{RT}} ln\Big(\frac{100}{100-y}\Big) \tag{4.6}$$

Where: $M_{RT}(y)$ - y percentile of request response time

T_s - mean service time

ρ - utilization

SLO_{RT} - response defined in service level objective

According to queue network theory for $M/M/c$ system traffic intensity [22, 49]:

$$\rho = \frac{\lambda}{\mu c}$$
$$\rho' = \frac{\lambda}{\mu}$$

Where ρ - single server utilization, ρ' - total utilization, λ - request arrival rate, μ - service rate and c - number of servers.

Assuming evenly distributed of incoming loads to all running servers, we can get average server utilization using total utilization and number of servers:

$$\rho_{avg} = \frac{\rho'}{c} \tag{4.7}$$

For $M/M/1$ server, $\rho' = \rho_{avg}$.

Elasticity Controller estimates required number of VMs using total utilization intensity in Equation 4.7 and utilization thresholds in Equation 4.5 and 4.6. 95^{th} percentile of response time is assumed to compute upper and lower threshold values of utilization in this study. Thus, ECCS acquire or release VMs when average CPU utilization deviates from upper threshold or lower threshold according to Algorithm 1 in Section A.1 of Appendix A.

4.1.4 Elasticity Action

Elasticity action calls Application Programming Interface (API) from cloud system management to apply VM reconfiguration (elasticity operation). Elasticity action can be either scale-out to add VMs or scale-in to remove VMs. No scaling operation is applied when average utilization is in the range of upper and lower threshold. Scale-out operation create VM instances and startup their OS and application. Conversely, scale-in operation stops

application running on VMs and shut down their OS before VMs are released. In addition, ECCS interacts with load balancer to update number of VMs in its cluster pool continuously in each scale-out or scale-in operations. Then load balancer distributes incoming requests to all available VMs. The load balancer also discontinues sending requests to VMs that will be released during scale-in operation.

Even though scale-in operation is easy compared to scale-out operation, it may have risk of stopping current connections to applications running on VMs that will be shut down before processing them. One solution for this problem is shutdown idle VMs if any or wait VMs until they are idle [15]. But, this is not efficient way to release VMs. TCP based application such as web application does not have risk of data loss; because, it can reestablish connection if the first connection is disconnected. In this research, TCP based applications, which run as a front-end servers, are considered to be controlled by Elastic Cloud Control System.

4.2 Two-step Ahead Prediction

Multi-step ahead time series prediction can be performed either directly from given observations or in recursive way where first predicted value need to be incorporated in predicting second value [32]. The prediction method provides multiple outputs in one step directly or predicts one out at a time and incorporate previous predictions to predict the next output one by one recursively. However, multi-step ahead prediction using a direct way does not keep sequence dependency in time series [32]. Recursive approach keeps the dependency of time series sequence but prediction inaccuracy rises as number of multi-step ahead predictions increases [32]. For example, in four-step ahead recursive predictions: a prediction value $y(t+4)$ at time $t+4$ in the future accumulates prediction errors at time $t+1$, $t+2$ and $t+3$. Thus, it is preferable to use small number of prediction steps to get more accurate values using recursive method.

Proactive model in our ECCS conducts continuous predictions in every proactive control interval. It loads actual observations on the basis of walk-forward sliding window method which adds new observations to the sliding window by replacing equal size of observations with older timestamp. The predictive model adds data points at timestamp $t+1$ and $t+2$ into the sliding window of n historical time steps and removes two data points at timestamp

$t-(n-1)$ and $t-(n-2)$ from the sliding window to keeping its size constant. In this research, recursive way of two-step ahead prediction is used where second step prediction depends on first step prediction. Figure 4.4 depicts walk-forward method for two-step ahead recursive predictions. First, predictive model performs one-step ahead prediction and include the value into sliding window to get second value recursively.Algorithm 2 in Section A.2 of Appendix A implements two-step ahead prediction in our Elastic Cloud Control System.

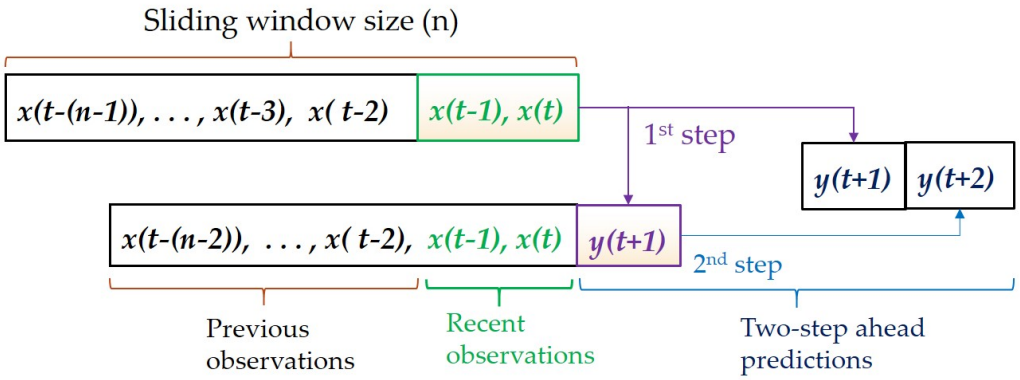

Figure 4.4: Sliding window based recursive prediction

Where $[x(t-(n-1)), x(t-(n-2)), ..., x(t-2), x(t-1), x(t)]$ *are vectors of* n *historical dataset in the past and* $[y(t+1), y(t+2)]$ *are vectors of future values of two-step ahead prediction.*

4.3 Combined Elasticity Approach

Elasticity Controller interacts with proactive model to forecast two-step ahead future values from historical VM CPU utilization data in each prediction control time. First value of the predictions is used to control uncertainty of proactive model while second value is used for scaling decision and resource estimation in proactive control time. Prediction accuracy of first-step prediction controls error propagation to second-step prediction in two-step ahead recursive prediction. In Figure 4.5, LSTM predicts average CPU usage P_{t+1} for time $t+1$ and P_{t+2} for time $t+2$ at time t using recent data points from the data store. LSTM starts online prediction by adding actual observations of VMs to the sliding window at time $t+2$ according to procedures given in Section 4.2. Then, predictions are performed every four minute interval (at $t+4, t+6, t+8, t+10, ...$). The time interval between $t+2$ and $t+3$ or $t+3$ and $t+4$ is two minute while between $t+2$ and $t+4$ or $t+4$ and $t+6$ is four

minute.

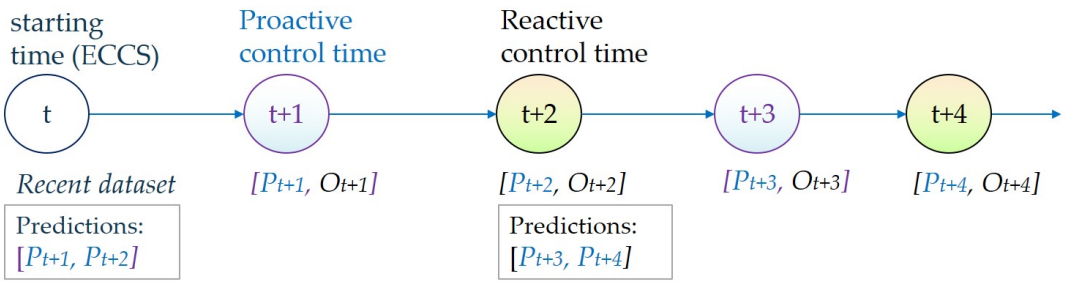

Figure 4.5: Elasticity controlling time

Where t, $t+1$, $t+2$ and $t+3$ are scaling control time, P_{t+1} is first-step prediction, P_{t+2} is second-step prediction at time t and O_{t+1} is observation at $t+1$, O_{t+2} is observation at $t+2$.

The error difference between the first prediction value P_{t+1} and actual observation O_{t+1} is calculated to evaluate prediction accuracy. The lower error value indicates better approximation of actual value. We used threshold to determine between proactive or reactive models for scaling decision in each consecutive control interval (See Figure 4.5) according to Equation 4.8 and 4.9. The use of proactive model for decision making depends on magnitude of the error value in Equation 4.8. If absolute value of the error is less than or equal to the threshold value at time $t+1$, then proactive model can perform scaling decision and resource estimation using second step prediction value (P_{t+2}).

$$|P_{t+1} - O_{t+1}| \leq threshold \tag{4.8}$$

If absolute value of the error is greater than the threshold value, Elasticity Controller skip decision making based on proactive method and uses reactive method in the next control time. The deviation between second-step prediction (P_{t+2}) with its corresponding observed value (O_{t+2}) will be evaluated in addition to first-step prediction error at time $t+2$. Reactive method is used for scaling decision and resource estimation when the absolute value of the error in first step prediction on Equation 4.8 or second prediction on Equation 4.9 are greater than the threshold.

$$|P_{t+2} - O_{t+2}| > threshold \tag{4.9}$$

Reactive model can not be applied when absolute value of errors in both first and second step predictions are less than or equal to the threshold value. Algorithm 3 in Section A.3 of Appendix A denotes elasticity method based on combined proactive and reactive approach for decision making and capacity planning in cloud environment. Scaling action is executed after system load observation in reactive model and before the next control interval in proactive model. Initially, Elasticity Controller interacts with cloud systems to obtain monitoring CPU utilization and active VM servers. It connects to proactive system to initiate two step ahead predictions on every prediction control time. The algorithm coordinates proactive and reactive elasticity based on prediction errors in different control time consecutively. Capacity planning is performed with the assumption of queue network theory according to Equation 4.5, 4.6 and 4.7. Predicted average CPU utilization will be multiplied with number of running VMs to get the total utilization intensity in proactive elasticity. CPU usage of individual VM instances will be aggregated in reactive elasticity. Finally, the algorithm estimate resources and execute auto-scaling action.

4.4 Predictive Model Retraining

System load can be changed due to the dynamic property of cloud environment. Thus, Elasticity Controller initiates online retraining at run time to adapt system load behavior. The online retraining updates weights of LSTM model using new time series observations following the same way as offline training. First predictive model uploads previously trained model and its corresponding weights and compile them. Then, it retrains the model with newly observed time series data at run time. The time series data obtained from cloud systems will be preprocessed automatically to fit LSTM model. After retraining and testing, new retrained model is compared with previous model based on error checking functions. If a new model has better performance than previous model, then it will be accepted as new forecasting model by replacing previous model. Algorithm 4 in Section A.4 Appendix A on line 7 will replace the previous model by new retrained model if the error value is less than previous error value. Finally, retrained LSTM network structure with its corresponding weights are saved in a file automatically.

4.5 LSTM Network Modeling

In this research, time series of CPU usage is used to build the LSTM network. Modeling LSTM for time series prediction includes three main steps: data preprocessing, LSTM model network construction, training and testing LSTM network [35]. The data preprocessing phase involves normalization or scaling the data to speed up LSTM network learning process and fast convergence [37]. Min-Max scaling algorithm converts actual time series input data to fall in range [0, 1] while Min-Max inverse scaling reverts scaled time series data into its actual value [37]. Min-Max scaling algorithm:

$$n_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{4.10}$$

Min-Max inverse scaling algorithm:

$$x_i = (n_i)(x_{max} - x_{min}) + x_{min} \tag{4.11}$$

Where x_i is actual observation, n_i is scaled data point, x_{min} and x_{max} are the minimum and maximum data points in the dataset

Moreover, data reshaping for LSTM network transforms time series data into consecutive sliding windows similar to supervised learning approach that involves independent (input) and dependent (output) variables [32, 33]. Recent observations sampled in specific period of time (e.g. an hour or half an hour) is taken as sliding widow or lag size that determines number of input neurons for LSTM network. Prediction output specifies the number of output neurons. Modeling LSTM network defines parameters such as number of input, hidden and output layer neurons as well as number of hidden layers [37]. In this study, Stacked LSTM model is used which contains two or more LSTM network layers (see Figure 2.6). The model also includes hyper-parameters such as lag size, number of epochs, batch size, and activation function.

After SLSTM network architecture is defined, training and testing is performed using preprocessed dataset to adjust SLSTM hyper-parameters and identify SLSTM model that provides better predictions. Prediction accuracy measures performance of SLSTM model to predict future values with a minimum deviation from actual observations [50]. Root

Mean Squared Error (RMSE), which is given by Equation 4.12, is a common technique to evaluate prediction accuracy. It measures the square root of the average of squared error between measured values and their corresponding predicted values [51].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_i - y_i)^2}{n}} \tag{4.12}$$

Where x_i represents actual observation, y_i represents corresponding predicted value and n denotes number of data points in the sample

5 Experiments

In this experiment, we have used a simulation environment to validate performance of the proposed cloud elasticity solution. Deploying test-bed environment is expensive besides to the difficulty of getting experimental environment from real cloud systems. We conducted simulation based experiments on Intel core-i7 processor (2.67GHz) and 8GB RAM configuration. The experimental evaluation of the proposed Elastic Cloud Control System includes predictive model training and testing, cloud elasticity simulation, and experimental result analysis and discussion.

5.1 Dataset

We obtained time series dataset of historical CPU load collected from an application system in ethio telecom are not in cloud environment. The dataset depicted in Figure 5.1 contains records of CPU utilization in percentage (%) on vertical axis which are sampled at every two minute interval on horizontal axis from the application server.

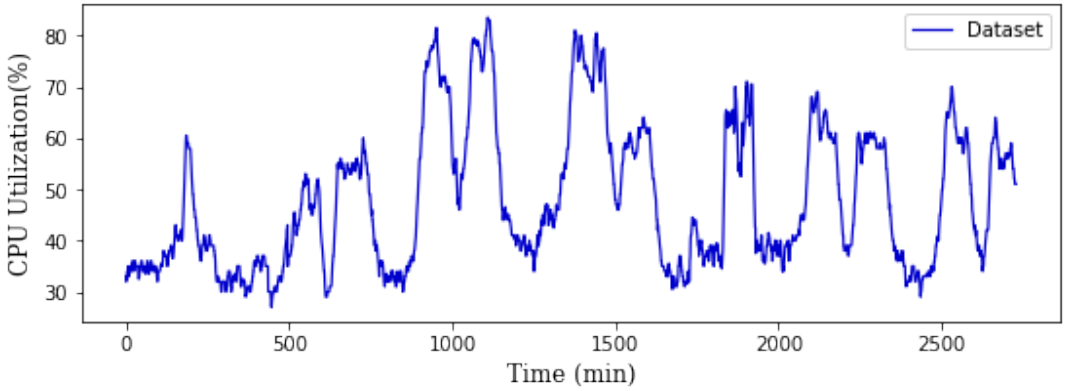

Figure 5.1: Dataset obtained from real system

The total dataset (2730) is divided into two parts where the first part (1730 records) is used for training and testing in LSTM network modeling. The second part (1000 records) is used in cloud elasticity simulation. Table 5.1 lists number of records, average, minimum and maximum CPU usage of each dataset. The first part of the dataset is normalized and reshaped into input and output format following the methods in Section 4.5. Then, we used 80% of it to train LSTM network while 20% for testing.

Table 5.1: Dataset Description.

	Records	Average	Minimum	Maximum
Total dataset	2730	40.05	27.0	83.5
Training and testing dataset	1730	47.7	27.0	83.5
Simulation dataset	1000	48.6	29.0	71.0

5.2 LSTM Model Training

Predictive model has been developed using Python programming language based on Keras libraries with Theano back-end. The model is built with SLSTM network (two LSTM network layers within hidden layer in this experiment). SLSTM network needs to be trained and tested to get trained model parameters for online prediction in proactive mechanism before ECCS becomes functional. Then, various train and test experiments are conducted in order to obtain the final model parameters by changing sliding window size and number of epochs. We updated and trained SLSTM network until test predictions closely approximate the actual dateset with lower RMSE value than previous trained model. RMSE value of test predictions is calculated based on Equation 4.12 to measure performance of trained LSTM model. Smaller value of RMSE indicates better performance of trained model. The final SLSTM network is trained using Adam (Adaptive Moment

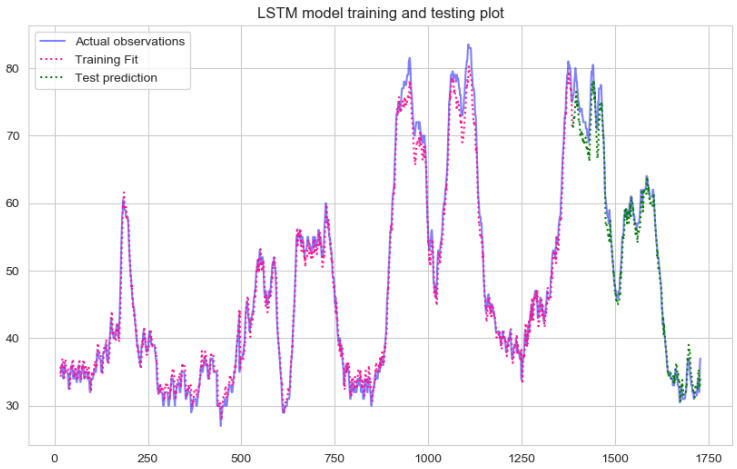

Figure 5.2: LSTM model offline training and testing

Estimation) optimization algorithm and sliding window of size 15 (size of collected data

points within half an hour). The number of epochs is set to 200 with 1 iteration per epoch. Figure 5.2 depicts the visualization of LSTM model training and testing where vertical line represents CPU utilization in percent (%) while horizontal line denotes data size in number sampled at every two minute interval. In the graph, solid blue line represents the original dataset and dotted lines in pink and green colors represent training fit and testing prediction.

5.3 Elasticity Simulation

In this section, first we prepare cloud systems with system monitoring component in the simulation environment. Additionally, experiments for different cloud elasticity scenarios are also configured using proposed solution to control elasticity operations of VMs in the cloud.

5.3.1 Implementation

In this study, simulator tool is used to setup an experimental environment. Experiments can be performed in quick, cheap and repeatable way using simulators. CloudSim is a toolkit for modeling and simulating a large scale cloud computing scenarios in research academy and industry [52]. CloudSim Plus is an extension of CloudSim framework that enables to simulate cloud elasticity besides modeling cloud computing service models [53]. Elastic Cloud Control System is implemented with CloudSim Plus simulator in this research work (see Figure 5.3).

CloudSim Plus contains library functions written in Java programming language such as *Datacenter, Host, VM, Cloudlet* and *Datacenter-broker* which are Java classes to instantiate an object during simulation process [52, 53]. The datacenter in CloudSim Plus represents real datacenter that consists of physical machines (PMs) which are also called hosts, storage disks and network devices. Virtual machines (VMs) are created on top of hosts which are characterized by number of CPU cores, amount of RAM and storage. Cloudlet represents application service requests to be executed by VM instances. The datacenter-broker is responsible for allocating and de-allocating VMs in cloud datacenter. CloudSim Plus supports dynamic creation and deletion of VMs using single Datacenter-broker that allows to experiment auto-scaling in cloud environment [53]. Additionally, it submits cloudlets to running VMs on behalf of applications or users.

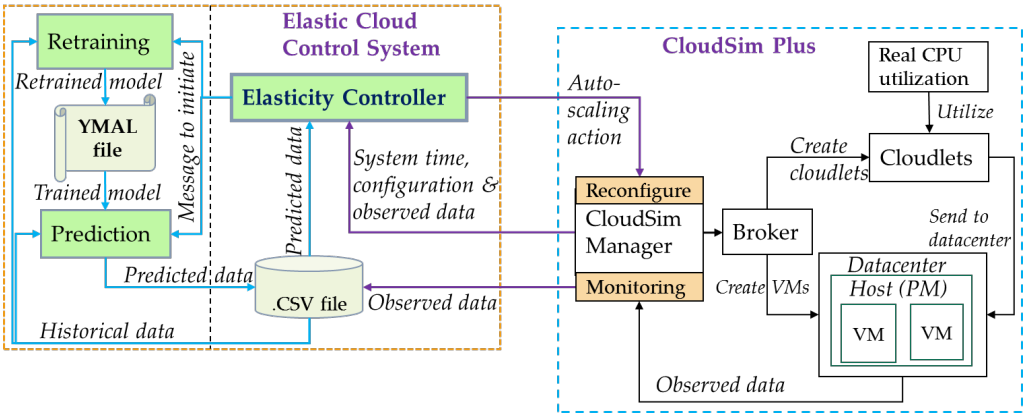

Figure 5.3: ECCS framework for simulation setup

CloudSim Manager is Java class developed to automate and simulate the proposed ECCS in CloudSim Plus environment. The Manager orchestrates VM and random cloudlets creation, monitoring data collection, and auto-scaling operation. Figure 5.3 shows the integration of Elastic Cloud Control System with CloudSim Manager to monitor cloud resources. CloudSim Manager adds or removes VM instances based on elasticity action triggered by Elastic Cloud Control System at run time. VM creation delay is set to one minute to make it similar to VM startup delay in real cloud environment. After VM startup, load balancer dispatches incoming cloudlets to all available VM instances. Monitoring component in Figure 5.3 is added to CloudSim Manager to collect actual observations of CPU load from each running VM instances at run time. CPU usage from individual VM server is sampled at regular monitoring time interval (every two minutes). The monitoring function aggregates total CPU utilization and computes average to generate time series of historical data that will be stored in CSV file (data store). The aggregated utilization is the sum of CPU load from individual VM in a cluster.

During simulation process, CloudSim Manager utilizes simulation dataset obtained from real system to set maximum CPU utilization of each cloudlet. As a result, CPU utilization of VMs do not increases when new VMs are released and decrease when new VMs are added. Thus, *Utilization Model* class in CloudSim Plus is extended to update cloudlet CPU consumption automatically to simulate the real world scenario. The extended utilization model divides real CPU usage by number VMs in each elasticity operation to realize distribution of loads on available VMs.

The second part of real system dataset in Subsection 5.1 with 1000 data records is utilized to experiment the simulation. The magnitude of few records in the simulation dataset are above upper threshold value of utilization; hence, scale-out operations to experiment resource over utilization cannot be tested. To get higher utilization, the simulation dataset is scaled by a factor of 1.4 keeping the maximum value less than 100 in unit of percentage (%). Elasticity Controller calculates upper and lower CPU utilization thresholds based on 95^{th} percentile of response time corresponding to SLO threshold requirements. Two SLO response time with values 0.02 and 0.04 seconds are considered to be upper and lower threshold response time. Then, lower and upper utilization thresholds can be calculated using Equation 4.5 and 4.6 assuming application service time of VM 0.004 second. Thus, ECCS is responsible to keep the average CPU utilization of VM instances in a cluster between the lower and upper thresholds in order to maintain the SLO requirement.

5.3.2 Experiment Setup

The components of ECCS are developed using Java and Python in addition to extending *"Horizontal_Auto-Scaling"* in CloudSim Plus example to *"CloudSim Manager"* as shown in Figure 5.3. Elasticity Controller and CloudSim Manager are Java classes where Elasticity Controller is working on the basis of client-server architecture with prediction and retraining components (developed in Python). Elasticity Controller contains functions to make a scaling decision, estimate resources, compute utilization threshold and connect to the predictive model for prediction and retraining automatically. It has an interface with CloudSim manager that includes functions to collect CPU usage and add or remove VMs besides orchestrating CloudSim Plus simulation environment. In addition to VM auto-scaling, Elasticity Controller initiates online retraining of SLSTM network at run time. The online retraining utilizes the last one day historical data (30 $data\ points\ per\ hour$ x 24 $hours$ = 720 $sequential\ data\ points$ can be generated in one day) from $.CSV$ file (data store).

In this experiment, one VM (2vCPU, 4GB RAM and 100GB storage) is configured initially and ECCS adds and removes VM instances with the same configuration according to load fluctuations. Simulation experiments are carried out under three main scenarios to demonstrate performance of ECCS:

1. **Reactive elasticity:** In this experiment, Elasticity Controller gets actual observations CPU utilization and number of running VM instances directly form the monitoring component in CloudSim Manager in real time. Then, it uses an aggregated value of CPU usage for scaling decision and capacity planning that can maintain the average utilization in the range of upper (70%) and lower (40%) thresholds. The scaling decision and action in reactive policy are executed in every four minute control interval.

2. **Proactive elasticity:** Predictive components are running as servers to accept prediction and retraining requests. Elasticity Controller connects to LSTM predictive model in every four minutes prediction interval to anticipate future value of average VM CPU utilization ahead of its actual observation. LSTM model uses the last half hour data points stored in $.CSV$ file (15 data points). The predicted values are also stored in $.CSV$ file and Elasticity Controller obtains them in each scaling control interval (four minutes). Scaling decision and action are applied before the next scaling control time (after two minutes of prediction time). Elasticity Controller multiplies predicted average CPU utilization by running VMs to calculate total CPU load of VMs in a cluster. Resources are estimated based on the total utilization to maintain average utilization between upper and lower thresholds.

3. **Combined proactive and reactive elasticity:** In this scenario, we conduct two experiments to demonstrate the combined effect of proactive and reactive elasticity approach. LSTM model predicts two-step ahead data points in the future based on the steps given in Section 4.2 with similar procedures described for proactive elasticity. The predictive model utilizes the last 15 data points from data store described in Section 5.1 to forecast first two-step ahead prediction. Afterwards, LSTM model follows sliding widow or walk-forwarding techniques by adding and removing data point for each prediction. proactive elasticity is scheduled in every four minutes scaling control interval.

   Two error thresholds are considered depending on deviation between observed and predicted values for the two-step ahead predictions. Proactive elasticity is applied when the deviation between first-step predicted value of CPU usage from its ac-

tual value is lower than 5% in the first scenario and 10% in the second scenario. Percentage(%) is used as a unit for CPU utilization. Elasticity Controller calculates required VM instances by multiplying the second-step prediction. After two minutes of proactive elasticity, Elasticity Controller uses using reactive model to adjust proactive model inaccuracy if one of the prediction value deviates from the threshold. Reactive elasticity and two step-ahead predictions are executed at the same time. Elasticity Controller coordinates the combined elasticity solution based on Algorithm 3 in Appendix A.

In each experiment, initially one VM is allocated to run the simulation. When average CPU utilization is greater than upper threshold, new VMs will be added to maintain application performance whereas if average CPU utilization is less than lower threshold VMs will be released to maximize resource efficiency and reduce cost of unused resources. Elasticity Controller estimates VM instances in each scaling control interval based on Algorithm 1 given in Section A.1 of Appendix A. A reconfiguration function in CloudSim Manager facilitates the elasticity action to add or remove estimated VM instances. ECCS performance is measured based on number of scale-out and scaling operations. Elasticity operations can be reactive scale-out, reactive scaling-in, proactive scale-out, proactive scale-in or no scaling action. Elasticity Controller executes only one scaling action in each scaling control time.

5.4 Results and Discussion

This section demonstrates results of simulation experiments conducted based on the setup in Section 5.3.2 and trained LSTM model from Section 5.2. The experiment is executed for each scenarios taking 33 hours long. Figure 5.4, 5.5, 5.6 and 5.7 illustrate visualization of the experimental results. In each figure gray solid line denotes CPU usage obtained from real system, green solid line denotes observed average CPU utilization of VMs during the simulation period and blue dotted line denotes predicted CPU utilization. While pink broken line represents number of allocated VM instances. The vertical axis on the left side represents percentage of CPU utilization while on the right side represents number of allocated VMs. While horizontal axis represents duration of the simulation with two minutes data sampling (system monitoring) interval. Moreover, letters P_i and R_i in

Figures 5.6 and 5.7 indicate scaling decisions performed by proactive and reactive model respectively in combined elasticity scenario where i denotes number of scaling operations.

5.4.1 Experiment 1: Reactive Elasticity

The initial experiment demonstrates performance of ECCS using only reactive model to compare it with proactive model and proposed elasticity approach. Figure 5.4 shows experimental results of reactive elasticity which adds and removes VM instances automatically depending on CPU load variation. Elasticity Controller performed scaling decision for resource planning when real time observations of average CPU usage deviates the upper and lower thresholds described in Section 5.3 (utilization 70% and 40% that corresponds to SLO response time with values 0.04 and 0.02 seconds).

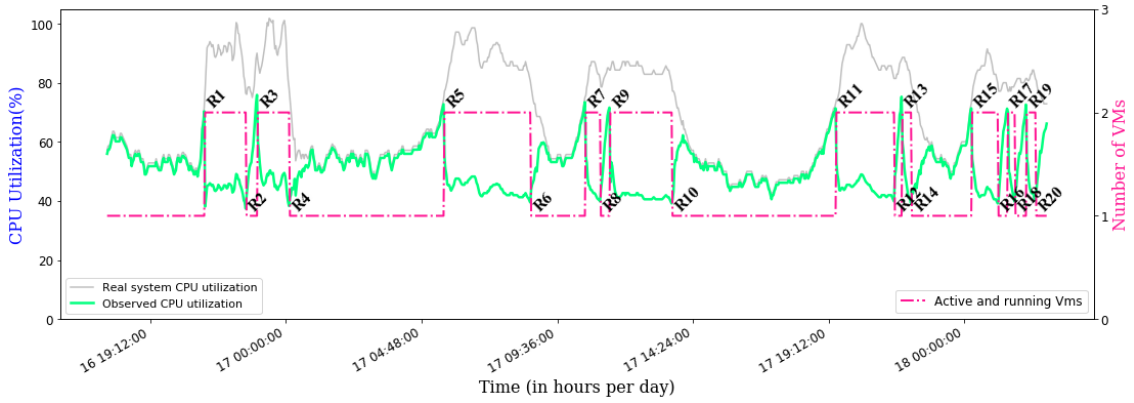

Figure 5.4: Result of Reactive cloud elasticity

In this experiment, Elasticity Controller executed 10 scale-out actions (allocate VMs 10 times) (indicated by R_i where i = 1,3,5, . . ,19) and also 10 scale-in actions (de-allocated VMs 10 times) (indicated by P_i where i = 2,4,6, . . ,20) during the simulation period. The Elasticity results demonstrated that if we use one VM, then it will be over utilized and if we use two VMs, then they will be under utilized or one of the VMs will be idle in conventional computing model. Hence, reactive elasticity adds VM to maintain application performance and remove under utilized or idle VMs to improve utilization efficiency better than convectional computing. Elasticity controller tries to keep the average CPU usage (green solid line) between 40% and 70%. However, VM readiness lags behind to respond instantly in each scale-out operations due to VM startup delay. According to the simulation setup in Section 5.3, VMs are ready to accept application requests after one minute from

the time when elasticity action is triggered. Thus, each VM is delayed by one minute in every auto-scaling action while running VMs are being over loaded. The delay in VM deployment violates application performance requirement in reactive elasticity. Moreover, newly allocated VMs can not share loads of active VMs in reactive elasticity instead it only handles loads of incoming requests after its startup time. Therefore, overloaded VMs will be a bottle-neck until their utilization load drop below upper threshold.

5.4.2 Experiment 2: Proactive Elasticity

Similar experiment is repeated to demonstrate proactive elasticity that can allocate VMs beforehand using predicted CPU usage from LSTM predictive model. Elasticity Controller is able to manage observed average CPU utilization below upper threshold and above the lower threshold values by adding and removing VMs. The actual utilization (green solid line) and corresponding predicted values (blue dotted line) in Figure 5.5 show that predicted CPU usage approximates with observed utilization.

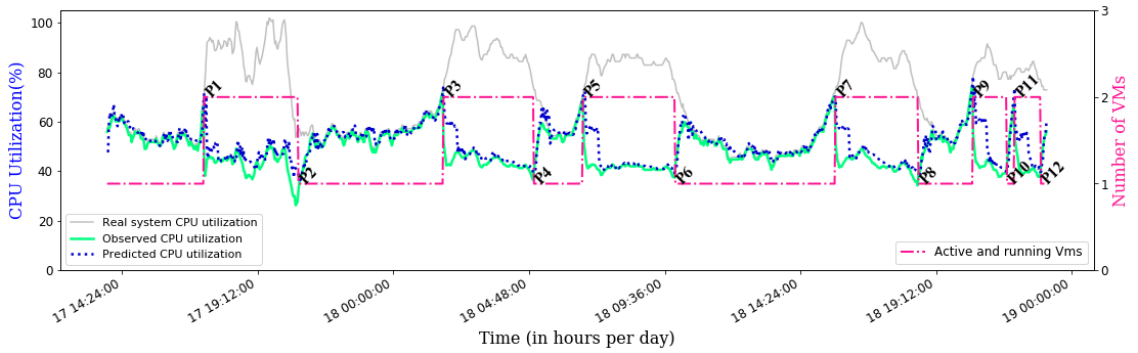

Figure 5.5: Result of proactive elasticity

Proactive elasticity executes 6 scale-out (VM addition) and 6 scale-in (VM release) actions during the simulation period. Thus, VMs are ready to share incoming loads before running VMs are overloaded in proactive scale-out operation. Two scale-out (indicated by R7 and R9) in Figure 5.4) and scale-in (R8 and R10) operations are performed by reactive elasticity between the time interval of scale-out (P5) and its corresponding scale-in (P6) operations in Figure 5.5 proactive elasticity. Shorter time intervals between the scale-in and scale-out operations indicate oscillation problem of reactive elasticity. Proactive elasticity performs better in maintaining application performance compared to reactive elasticity. However, prediction inaccuracy results inefficient utilization in proactive model. In Figure 5.5, CPU

usage falls below the lower threshold before scale-in operation is applied indicated by P2 due to prediction inaccuracy. Uncertainty of proactive model increases when the original pattern changes due to the dynamic allocation and de-allocation of computing resources. The deviation between prediction and actual values became higher when the CPU load drops during VM addition in Figure 5.5 (indicated by P3, P5, P7 and P9). Therefore, predictive model fails to handle prediction accuracy when the system load drops suddenly.

5.4.3 Experiment 3: Combined Elasticity Approach

In this scenario, the experimental results in Figures 5.6 and 5.7 depict performance of proposed approach. The first result is with 5% and second one with 10% error thresholds between predicted values and actual observations of average CPU utilization.

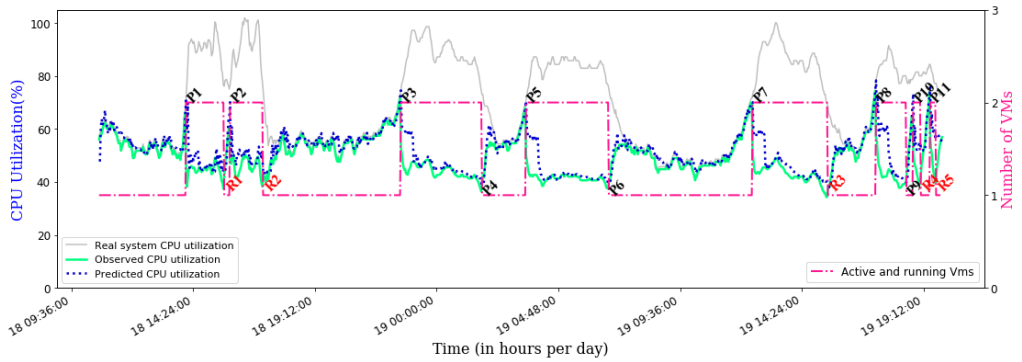

Figure 5.6: Result of combined elasticity(5% Threshold)

Elasticity Controller uses reactive model to adjust 5 prediction inaccuracy from 8-scale-in actions (indicated by R1, R2, R3, R4 and R5) with 5% threshold and 3 prediction inaccuracy from 7 scale-in actions (indicated by R1, R2, R3) with 10% threshold respectively. Experimental results in Figure 5.7 indicates the trade-off between oscillation problem and prediction inaccuracy when the deviation threshold increases from 5% to 10%. When the threshold increases, oscillation problem will be minimum whereas inaccuracy of prediction techniques rises causing either performance degradation or inefficient resource utilization. Elasticity solution with 5% threshold performs 8 scale-out and 8 scale-in operations which is better in resources efficiency than 10% threshold with 7 scale-out and 7 scale-in operations. From the experimental results, 5% deviation threshold between actual observations and predicted values is more preferable than 10% deviation threshold to guarantee the balance between application performance and efficient resource utilization.

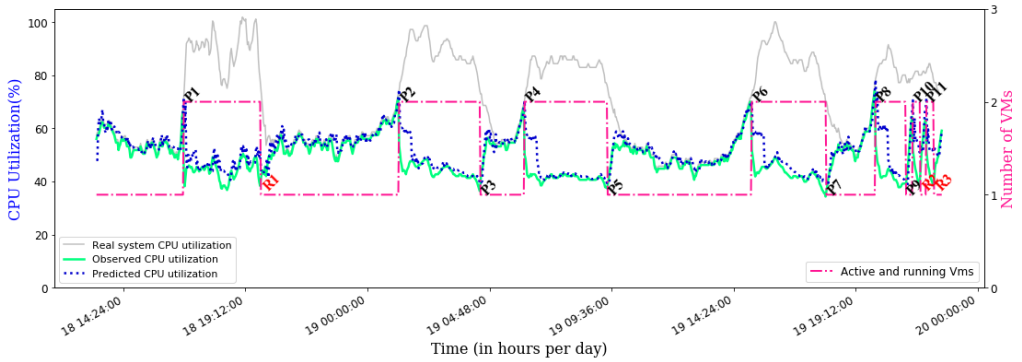

Figure 5.7: Result of combined elasticity(10% Threshold)

Elasticity Controller performs 10 scale-out and scale-in actions in reactive elasticity, 6 scale-out and scale-in actions in proactive elasticity and 8 scale-out and 8 scale-in actions in the combined approach (with 5% prediction error threshold). The proposed approach mitigates deployment delay, prediction inaccuracy and oscillation problem compared to reactive elasticity in Subsection 5.4.2 and reactive elasticity in Subsection 5.4.1. In addition, ECCS maintains application performance requirements while utilizing minimum VM instances. Therefore, the proposed solution balances application performance and efficient resource utilization better than each single model. Table 5.2 shows summarized outputs of the three experimental scenarios. In each experiment, CloudSim Manager al-

Table 5.2: Simulation results.

	Reactive	**Proactive**	**Combined approach with 5% deviation**
Scaling action (up/down)	10/10	6/6	8/8
VM startup delay	10 minutes		
Proactive inaccuracy		5	
Reactive correction			5

located initially one VM to receive and process randomly created cloudlets that represent application requests. Elastic Cloud Control System manages dynamic provisioning of VM instances using different methods. The VM startup delay in reactive elasticity results in VM over or under utilization. The experiments showed that elasticity solution using reactive method degrades performance of an application while proactive model leads to prediction

inaccuracy over time. Elastic Cloud Control System allocated VMs proactively and start them up ahead of time before actual observation occurs only at scaling control time with better prediction accuracy. The proposed elasticity solution based on combined approach mitigates challenges of reactive and proactive models. In Figure 5.5-5.7, prediction values deviates from real time observations when CPU utilization decreases rapidly and proactive model is unable to capture new patterns that leads to over or under provisioning of VM instances. Furthermore, online retraining of predictive model at run time enables proactive model to adapt changes due to dynamic properties of cloud environment [9].

5.4.4 LSTM Model Online Retraining

Prediction values in Figures 5.5, 5.6 and 5.7 deviates from actual observations when CPU load pattern changes due to addition or removal of VMs. SLSTM network is retrained to adapt new patterns in Figure 5.8 using observations of CPU utilization gathered during simulation process. The vertical line represents CPU load in percent (%) and horizontal line data sampling duration. On the graph, red dotted line for retraining and green dotted line for testing closely matches with actual observation in blue solid line. The test performance provides RMSE value of 1.9 with similar parameters to offline training in Section 5.2.

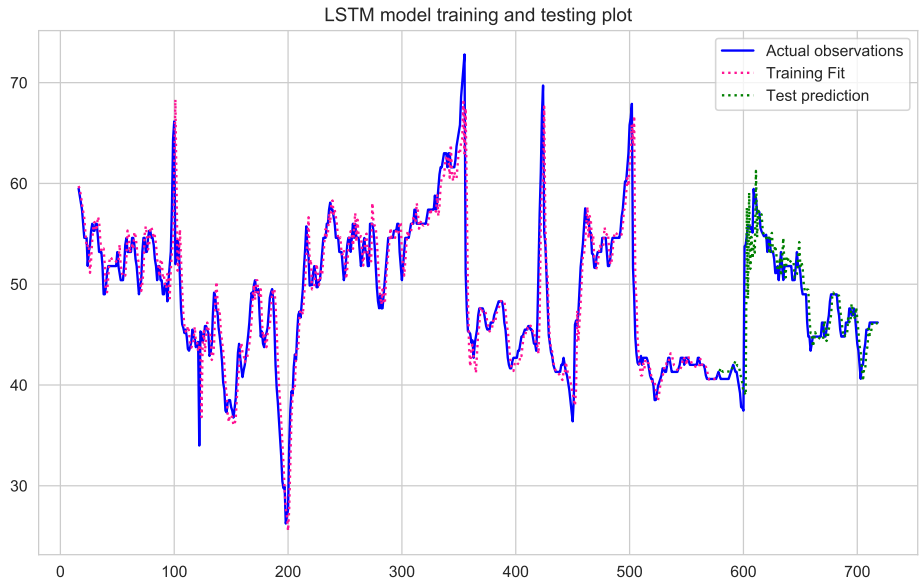

Figure 5.8: LSTM model online retraining and testing

Table 5.3 shows RMSE values and training duration of offline training from Section 5.2 and online retraining results. The visualization in Figure 5.8 and RMSE result shows

Table 5.3: RMSE values and training duration.

	Offline trained model	**Online trained model**
RMSE value	1.91	1.9
Training duration	12:58:40	05:30:39

that online retraining of LSTM predictive model is able to learn new changes at run time. Thus, retraining predictive model enriches proactive elasticity to improve its prediction inaccuracy by adapting new load changes over time. In this study, elasticity simulation experiment using retrained model is limited by number of real system data after retraining and simulation environment and performance of computer used for simulation and retraining. The training duration depends on not only computational performance of a computer system used to implement predictive model but also number of epochs and training data size. This research contributes further insight to knowledge in cloud computing elasticity.

6 Conclusion and Future Works

In this research work, Elastic Cloud Control System have been proposed using combined proactive and reactive mechanisms with two step ahead prediction for elasticity decision making. ECCS processes are organized based on MAPE-K loop architecture to dynamically allocate minimum number of VMs maintaining application performance requirement. We have used Long-Short Term Memory Recurrent Neural Network for predictive techniques in proactive mechanism to perform prediction of VM CPU usage at run time. Proactive and reactive models in the proposed approach are combined using deviation between first-step predicted value in two-step ahead prediction and its corresponding observed value. LSTM predictive model has been developed and implemented to perform offline and online training. Online retraining of predictive model enables LSTM model to adapt new changes of system load in the cloud environment and improve proactive model accuracy. Elasticity Controller, which coordinates proactive and reactive models, is developed to work with proactive model on the basis of client-server architecture. The client-server architecture enables to integrate components of proactive model with other elasticity solutions by adjusting only model parameters or retraining.

CloudSim Plus is used to setup simulation environment utilizing real CPU usage to perform empirical analysis of the proposed solution. Experiments are conducted with three different scenarios to demonstrate comparative analysis of reactive, proactive and combined elasticity approaches. Threshold value for deviation between first-step predicted value and its observed value are used to determine on the use of proactive or reactive model. The experiment for combined elasticity model was conducted using two error threshold values (5% and 10%). The results showed that proposed Elastic Cloud Control System performs better to improve proactive model accuracy and resource deployment delay with 5% threshold value. The result demonstrates significance of ECCS to maintain application performance keeping efficient resource utilization. Furthermore, online retraining of LSTM model shows that predictive model is able to adapt new changes in the system load at run time.

In this study, simulation is implemented to setup an experiment and evaluate Elastic Cloud

Control System. As a future work, ECCS will be investigated in real cloud environment that allows to experiment the effect of online retrained model. Further investigation is also required to extend this work to provision computing resources form heterogeneous cloud infrastructures. In this study, horizontal elasticity adjusts VMs in course grained manner; thus combining horizontal with vertical elasticity solution enhances resource utilization efficiency and application performance.

Challenges and Limitations:

Predictive model training and simulation experiment takes longer time due to the lower computational capacity of physical machines that is used in this research. Setting CloudSim Plus is tedious and time consuming to configure precise interval for sampling system load data, scaling operation and cloudlet creation for cloud elasticity solution in the combined approach. Experimental simulation of elasticity using online retraining model was not conducted due to the limitation on performance of computing devices and simulation environment to support the experiment. Cloudlet maximum utilization is set based on real system CPU consumption sampled with values between 0 and 100%. As a result, ECCS is limited to add only one VM in each scaling control interval to satisfy application requirement during over utilization of system load.

References

[1] A. Fox, A. D. Joseph, and R. H. Katz, "Above the Clouds : A Berkeley View of Cloud Computing," no. February, 2009.

[2] D. David and K. Felicity, "Cloud Computing: The Emergence of the 5th Utility," in *Industry Trends in Cloud Computing*, ch. 1-3, pp. 1–44, 2018.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," vol. 53, pp. 50–58, 2010.

[4] L. M. Vaquero, L. Rodero-merino, J. Caceres, and M. Lindner, "A Break in the Clouds : Towards a Cloud Definition," vol. 39, no. 1, pp. 50–55, 2009.

[5] R. Buyya, C. Shin, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms : Vision , hype , and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, pp. 1–18, 2009.

[6] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," 2011.

[7] M. G. M.A.N. Bikas, A. Alourani, "How Elasticity Property Plays an Important Role in the Cloud: A Survey," *Advances in Computers*, vol. 103, pp. 1–29, 2016.

[8] C. Qu, R. N. Calheiros, and R. Buyya, "A Auto-scaling Web Applications in Clouds: A Taxonomy and Survey," vol. 51, no. 4, 2018.

[9] G. Galante and L. C. E. D. Bona, "A Survey on Cloud Computing Elasticity," no. 1, 2012.

[10] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing : state-of-the-art and research challenges," pp. 7–18, 2010.

[11] J. Varia, S. Mathew, "Overview of Amazon Web Services." [Online] Available: `https://media.amazonwebservices.com/AWS_Overview.pdf` [Accessed:2019-10-02], January 2014.

[12] Y. Al-dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in Cloud Computing : State of the Art and Research Challenges," pp. 1–18, 2017.

[13] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing : What It Is , and What It Is Not,"

[14] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications ' QoS," pp. 1–11, 2014.

[15] J. A. Pascual, J. A. Lozano, and J. Miguel-alonso, "Effects of Reducing VMs Management Times on Elastic Applications," 2018.

[16] M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," 2012.

[17] Y. Liu, N. Rameshan, E. Monte, V. Vlassov, and L. Navarro, "ProRenaTa : Proactive and Reactive Tuning to Scale a Distributed Storage System," 2015.

[18] N. Herbst, S. Spinner, A. Ali-eldin, and S. Kounev, "Chameleon : A Hybrid , Proactive Auto-Scaling Mechanism on a Level-Playing Field," pp. 1–14, 2018.

[19] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, "The Characteristics of Cloud Computing," 2010.

[20] G. Aceto, A. Botta, W. D. Donato, and A. Pescapè, "Cloud monitoring : A survey," 2013.

[21] S. Lima, Á. Rocha, and L. Roque, "An overview of OpenStack architecture : a message queuing services node," *Cluster Computing*, 2017.

[22] G. Huang, S. Wang, M. Zhang, Y. Li, Z. Qian, Y. Chen, and S. Zhang, "Auto Scaling Virtual Machines for Web Applications with Queueing Theory," pp. 433–438, 2016.

[23] T. Lorido-botran, J. Miguel-alonso, and J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," vol. 12, no. 4, pp. 559–592, 2014.

[24] E. Casalicchio and L. Silvestri, "Architectures for Autonomic Service Management in Cloud-based Systems," pp. 2010–2011, 2011.

[25] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting," 2011.

[26] G. P. M. D. D. Demetris Trihinas, Zacharias Georgiou, "Improving Rule-Based Elasticity Control by Adapting the Sensitivity of the Auto-Scaling Decision Timeframe," pp. 123–137, 2018.

[27] M. Borkowski, S. Schulte, and C. Hochreiner, "Predicting Cloud Resource Utilization," 2016.

[28] J. Brownlee, "The Promise of Recurrent Neural Networks for Time Series Forecasting." [Online] Available: `https://machinelearningmastery.com/promise-recurrent-neural-networks-time-series-forecasting/` [Accessed:2019-10-20], 2019.

[29] J. Gamboa, "Deep Learning for Time-Series Analysis," 2017.

[30] J. Deepak and B. Enda, "CPU Workload forecasting of Machines in Data Centers using LSTM Recurrent Neural Networks and ARIMA Models," 2017.

[31] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep Learning with Long Short-Term Memory for Time Series Prediction," *IEEE Communications Magazine*, pp. 1–6, 2019.

[32] G. Bontempi, S. B. Taieb, and L. Borgne, "Machine Learning Strategies for Time Series Forecasting," pp. 62–77, 2013.

[33] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, "Host load prediction with long short-term memory in cloud computing," 2017.

[34] M. Hermans and B. Schrauwen, "Training and Analyzing Deep Recurrent Neural Networks," pp. 1–9, 2013.

[35] I. Kaastra and M. Boyd, "Designing steps of NN for forecasting TS.pdf," 1996.

[36] C. Ian, Goodfellow; Yoshua, Bengio; Aaron, "Deep Feedforward Networks and Sequence Modeling: Recurrent and Recursive Nets," in *Deep Learning*, ch. 6 & 10, pp. 168–227 & 373–420, MIT Press, 2016.

[37] J. Brownlee, "Recurrent Neural Networks," in *Deep Learning With Python*, ch. 23-25, pp. 169–200, v1.7 ed., 2016.

[38] S. Sepp, Hochreiter; Jurgen, "LONG SHORT-TERM MEMORY," vol. 9, no. 8, pp. 1–32, 1997.

[39] Colah, "Understanding LSTM Networks." [Online] Available: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/` [2019-08-20], 2015.

[40] J. Brownlee, "Stacked Long Short-Term Memory Networks." [Online] Available: `https://machinelearningmastery.com/stacked-long-short-term-memory-networks/` [Accessed:2019-09-10], 2019.

[41] "An architectural blueprint for autonomic computing," 2006.

[42] C. Barna, M. Fokaefs, M. Litoiu, and M. Shtern, "Cloud Adaptation with Control Theory in Industrial Clouds," 2016.

[43] E. Bauer and J. Weinman, "Cloud Automation and Economic Efficiency," no. April 2018, pp. 26–32.

[44] M. Maurer, I. Breskovic, V. C. Emeakaroha, and I. Brandic, "Revealing the MAPE Loop for the Autonomic Management of Cloud Infrastructures," pp. 147–152, 2011.

[45] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," vol. 37, 2015.

[46] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Monitoring Elastically Adaptive Multi-Cloud Services," pp. 1–14, 2015.

[47] An Isle, "Keras + LSTM for Time Series Prediction." [Online] Available: `https://talkai.blog/2018/10/01/keras-lstm-for-time-series-prediction/` [Accessed:2019-09-08], 2018.

[48] S. William, "Queuing Analysis." [Online] Available: `www.csci.csusb.edu/ykarant/courses/w2008/csci531/QueuingAnalysis.pdf` [Accessed:Jun 20, 2019], 2000.

[49] M. Harchol-Balter, "Performance Modeling and Design of Computer Systems Queueing Theory in Action," ch. 13, pp. 236–281, cambridge university press, first ed., 2013.

[50] C. Vazquez, R. Krishnan, and E. John, "Time Series Forecasting of Cloud Data Center Workloads for Dynamic Resource Provisioning," pp. 87–110.

[51] P. Singh, "TASM : technocrat ARIMA and SVR model for workload prediction of web applications in cloud," *Cluster Computing*, vol. 6, 2018.

[52] R. N. Calheiros, R. Ranjan, A. Beloglazov, and A. F. D. Rose, "CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," no. August 2010, pp. 23–50, 2011.

[53] M. C. S. Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inacio, and M. M. Freire, "CloudSim Plus : A Cloud Computing Simulation Framework Pursuing Software Engineering Principles for Improved Modularity , Extensibility and Correctness," *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 400–406, 2017.

A Appendix

A.1 Algorithm for Resource Estimation

Algorithm 1 Algorithm for VM capacity estimation

input: $RT\_upperThreshold, RT\_lowerThreshold, totalCPU\_util, RunningVms$
output: $additionalVms$
procedure ResourceEstimation
    additionalVms $\leftarrow 0$
    $caculateUtilThreshold(RT_upperThreshold, RT_lowerThreshold)$
    $Avg\_util = totalCPU\_util/numOfRunningVms$
    if $avg\_util \leq upperUtilThreshold$ **then**
        while $avg\_util \leq upperUtilThreshold$ **do**
            $additionalVms = additionalVms + 1$
            $Avg\_util = totalCPU\_util/(RunningVms + additionalVms)$
        end while
    else if $avg\_util \geq upperUtilThreshold$ **then**
        while $avg\_util \leq upperUtilThreshold$ **do**
            $additionalVms = additionalVms - 1$
            $Avg\_util = totalCPU\_util/(RunningVms + additionalVms)$
        end while
    end if
end procedure

A.2 Algorithm for Two-step Ahead Prediction

Algorithm 2 Algorithm for Two-step ahead Online prediction

input: $trainedModel.YAML, trainedModel_weights.H5, slidingWindowSize$
output: $firstPrediction, secondPrediction$
procedure TwostepAheadPrediction
    while ***true*** **do**
        $acceptPredictionRequest()$
        $model \leftarrow loadTrainedModel(trainedModel.YAML, trainedModel_weights.H5)$
        $predictionDataset \leftarrow getLastObservedDataset(slidingWindowSize)$
        $firstPrediction \leftarrow predictUtil(predictionDataset, model)$
        $predictionDataset \leftarrow add(firstPrediction, predictionDataset)$
        $secondPrediction \leftarrow predictUtil(predictionDataset, model)$
    end while
end procedure

A.3 Algorithm for Elasticity Controller

Algorithm 3 Algorithm for Elasticity Controller

input: $controlTime, accuratePrediction$ // *Initialize accuratePrediction* $\leftarrow$ *false*
output:
procedure ElasticityController
    numVms $\leftarrow getRunningVms()$
    if $controlTime = proactive_timeInterval$ **then**
        $firstStepPredictedUtil \leftarrow getPredictions(controlTime)$
        $nextTime \leftarrow controlTime + controlInterval$
        $secondStepPredictedUtil \leftarrow getPredictions(nextTime)$
        $observedUtil \leftarrow systemMonitoring(controlTime)$
        $error \leftarrow calculateError(observedUtil, firstStepPredictedUtil)$
        if $error \leq errThreshold$ **then**
            $accuratePrediction \leftarrow$ ***true***
            $aggregatedUtil \leftarrow secondStepPredictedUtil$ x $numVms$
        else
            $accuratePrediction \leftarrow$ ***false***
        end if
    else if $controlTime = reactive_timeInterval$ **then**
        $secondStepPredictedUtil \leftarrow getPredictions(controlTime)$
        $observedUtil \leftarrow systemMonitoring(controlTime)$
        end if
        $prevError \leftarrow getPrevTimePredictionError(prevTime)$
        $error \leftarrow calculateError(observedUtil, secondStepPredictedUtil)$
        if ($error \geq errThreshold$) **or** (**not** $accuratePrediction$) **then**
            $aggregatedUtil \leftarrow sumOfObservedUtil(controlTime)$
        end if
        $forecastNewPredictions(controlTime)$ // *Connect to online predictive model*
        end if
        $additionalVms \leftarrow estimateResource(aggregatedUtil, numVms)$
        $reconfigureVms(additionalVms)$
end procedure

A.4 Algorithm for Predictive Model Retraining

Algorithm 4 Algorithm for Predictive Model Online Retraining

input: $retrainingDataSize, trainedModel.YAML, trainedModel\_weights.H5$
output: $retrainedModel$
procedure OnlineRetraining
    while ***true*** **do**
        $acceptRetrainingRequest()$
        $retrainingTimeSeries \leftarrow getRetrainingDataset(retrainingDataSize)$
        $model \leftarrow loadTrainedModel(trainedModel.YAML, trainedModel_weights.H5)$
        $retrainedModel \leftarrow retrainPredictiveModel(retrainingTimeSeries, model)$
        $retrainModelErr \leftarrow calculateRetrainingTestPerformance()$
        $prevTarinedModelErr \leftarrow getPreviousTrainingError()$
        if $retrainModelErr \leq prevTarinedModelErr$ **then**
            $update(trainedModel.YAML, trainedModel_weights.H5$
        end if
    end while
end procedure