An Auto-Scaling Approach using Load Prediction for IoT-based Cloud Application

A Thesis

submitted in partial fulfillment of the requirements for the award of degree of
DOCTOR of PHILOSOPHY

by

Shveta Verma
(901503036)

under the guidance of

Dr. Anju Bala
Associate Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala -147004

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology,
Patiala - 147004, INDIA

May 2024

Candidate Declaration

I hereby certify that the work, which is being presented in the thesis, entitled **An Auto-Scaling Approach using Load Prediction for IoT-based Cloud Application**, in partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted to the institution is an authentic record of my own work carried out during the period **January 2016** to **May 2024** under the supervision of **Dr. Anju Bala**. I have also cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

The matter presented in this thesis has not been submitted elsewhere for the award of any other degree or diploma from any institution.

(**Shveta Verma**)
Regn. No. 901503036

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

22/5/2024

(**Dr.Anju Bala**)
Associate Professor
Department of Computer Science & Engineering
Thapar Institute of Engineering and Technology, Patiala, 147004
Punjab, INDIA

.....dedicated to all young generation researchers

Acknowledgements

Almighty has bestowed me with the wisdom and perseverance to accomplish this milestone. I am forever grateful to GOD for HIS everlasting blessings.

At the outset, I would like to offer my deep gratitude to my supervisor Dr. Anju Bala for her invaluable advice and encouragement at every step of my PhD program. Without her unfailing support and belief in me, this thesis would not have been possible. Her constant encouragement, immense knowledge and scientific acumen lightened up the way in my darkest times. I am indebted for her insightful discussions and enlightening suggestions that assisted me to shape the direction of this research. The time and energy spent on careful and patient proof readings of this manuscript helped me to present this thesis with the desired quality. And for this, I am truly grateful.

I am highly thankful to Dr. Shalini Batra, Head, Computer Science and Engineering Department for his constant motivation and encouragement. I would like to express my sincere thanks to Dr. Inderveer Chana, Dean of Student Affairs, Computer Science and Engineering Department, for generously sharing her time and knowledge and carrying out this research in an efficient manner.

I also wish to thank Dr. M. D. Singh for being my Doctoral committee member and advising me during the initial stages of my research studies. I would also like to thank Dr. N. Tejo Prakash, Dean of Research and Sponsored Projects, for academic support. I sincerely thank Dr. Padmakumar Nair, Director, Thapar Institute of Engineering and Technology, for providing all the facilities and resources that have been instrumental in creating a robust research environment in the University.

I thank Dr. Anil K. Verma, Professor of Computer Science and Engineering Department, for advising and helping me achieve my objective. I would like to thank administrative and technical staff members of the Computer Science and Engineering Department who have been kind enough to advise and help in their respective roles. I would also like to thank my mates Dr. Gurleen Kaur and Dr. Deep Mann for all the great times we have shared and for making this research experience enjoyable and memorable.

This thesis would never have been completed without the motivation and support of my family. My hardworking family deserves a special mention here. My mother, Ms. Shashi Verma infused moral, ethical and religious values in me. She made me smile and realize their faith in me during the rough road of this journey. She has given me a wonderful humanitarian lineage and a good foundation to face life's challenges. I

gratefully acknowledge the patience and love of my sister, Nitika Bansal, and brother, Anmol Verma, who supported the family in my absence and let me pursue my dream. A special mention to my father, (Late) Mr. Surinder K. Verma who was always a great support and inspired me to dream big.

My warmest thanks to my dear husband Mr. Kuldeep Bansal whose unconditional support during all these years is highly appreciated. He instilled confidence in me and inspired me in all dimensions of life for the successful completion of this journey. I will always be indebted to you for your everlasting love and commitment that carries me through. I look forward to a life full of happiness and togetherness.

I would also like to mention that this work is dedicated to my children Tejas and Kaashvi. You have made me stronger, better and more fulfilled than I could have ever imagined. You are my inspiration to achieve greatness. Without you, I would not be where I am today. Thank you for being there for me at the end of the day. Your love has gotten me through when I wanted to give up.

Last but not least, I would like to extend my deep gratitude to my father-in-law, Mr. Ashok K. Bansal, and mother-in-law, Ms. Sunita Bansal, who extended their cooperation and encouragement to me. I am indebted to them and this thesis would not have been complete without their everlasting support.

Shveta Verma

Abstract

Cloud Computing enables a dynamic platform for deploying numerous internet applications by offering services such as software, and infrastructure. The Internet of Things (IoT) offers a variety of potential opportunities and applications, including smart grids, smart cities, intelligent transportation systems, e-health, and more, to fulfill various Cloud services. Thus, to address the current and future needs of end users, IoT and cloud computing are merged for the overall growth and organization of such applications.

In addition to the advantages, complex cloud scenarios provide several challenges for IoT applications, such as scalability, reliability, heterogeneity, security, and privacy. One of the emerging issues for different IoT-based Cloud applications is auto-scaling. These days, many applications benefit from the auto-scaling capability, which allows them to scale up and down resources automatically. For efficient and autonomic scaling, predicting the future load on the host is recommended. This is because the load fluctuations may occur due to the dynamic resource usage among Cloud tenants. The over-utilized or under-utilized status of the host can be monitored based on the predicted resource utilization, and the migration process can be performed to maintain the load on that host.

In this research work, an auto-scaling technique has been proposed using load prediction for IoT-based Cloud application. First, an extensive literature survey of existing Cloud-based IoT applications has been done to achieve the objectives. Furthermore, state-of-the-art auto-scaling techniques have been surveyed, and necessary Quality of Service (QoS) parameters have been keyed out. From the literature, it can be inferred that load prediction-based auto-scaling is a challenging issue that must be handled intelligently.

For dynamic load prediction, an Ensemble Time-Series Approach for Load Prediction (ETSA-LP) has been proposed which integrates five time-series analysis techniques (ARIMA,

ANN, SVM, LSTM & ES) for predicting CPU and memory utilization. To evaluate the efficiency of the proposed approach, a series of experiments on Google and Planet-Lab traces have been conducted in a real Cloud environment. The proposed ensemble approach gives the best performance over the existing models by showing remarkable accuracy improvement and reducing the error rate and execution time.

Further, an efficient auto-scaling approach for predicting host load through Virtual Machine (VM) migration has been proposed. Different algorithms have been devised to detect over-utilized and under-utilized hosts based on the predicted resource utilization. Also, a VM migration algorithm has been deployed that helps to choose the appropriate VM to be migrated. The designed approach has been validated by experimentation on a real-time Google cluster dataset. The proposed technique significantly improves average CPU utilization and reduces over-utilization and under-utilization. It also minimizes response time, SLA violations, and the slighter number of migrations and scaling overhead.

For testing and validating the proposed approach, actual cloud platforms, GCP (Google Cloud Platform) and AWS (Amazon Web Services) have been used. In this research work, heterogeneous VMs (Virtual Machines) are created for parallel execution and validating the performance of the proposed approach. The average load variations and status of VMs at different time intervals have been monitored.

A case study based on a fog platform for IoT applications has also been considered for further enhancement. A Fog-Enabled Auto-Scaling Technique (FEAST) has been proposed to predict resource usage concerning CPU, memory, disk, and network utilization using the deep learning model. Several experiments have been conducted to evaluate performance and validate several indicators related to resource usage, scaling overhead, delay, and SLA violations, etc.

List of Publications

International Journal

1. **Shveta Verma**, and Anju Bala, "Auto-Scaling Techniques for IoT-based Cloud Applications: A Review", *Cluster Computing*, Springers, 24(3), 2425-2459, 2021. [**SCI Indexed, Impact Factor - 4.4**]

2. **Shveta Verma**, and Anju Bala, "ETSA-LP: An Ensemble Time-Series Approach for Load Prediction in Cloud", *Computing and Informatics*, 2023. [**SCI Indexed, Impact Factor - 0.972**]

3. **Shveta Verma**, and Anju Bala, "Efficient Auto-Scaling for Host Load Prediction through VM Migration in Cloud," *Concurrency and Computation: Practice & Experience*, Wiley Publications, e7925, https://doi.org/10.1002/cpe.7925. [**SCI Indexed, Impact Factor - 1.831**]

International Conference

1. **Shveta Verma** and Anju Bala, "A Review: Intelligent Load Prediction Techniques for CloudIoT", *Proceedings of the 3rd International Conference on Advanced Informatics for Computing Research (ICAICR-2019)*, 2019.

Communicated

1. **Shveta Verma** and Anju Bala, "FEAST: Fog Enabled Auto-Scaling Technique for IoT-based Cloud Application", *Cluster Computing*, 2024 [**SCI Indexed, Impact Factor - 4.4**].

Table of Contents

List of Figures

List of Tables

List of Abbreviations

IoT	Internet of Things
QoS	Quality of Service
SLA	Service Level Agreement
CPU	Central Processing Unit
PMs	Physical Machines
VMs	Virtual Machines
CSPs	Cloud Service Providers
CDCs	Cloud Data Centres
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
RL	Reinforcement Learning
CT	Control Theory
QT	Queuing Theory
MDP	Markov Decision Process
RMSE	Root Mean Square Error
LA	Learning Automata
HPC	High Performance Computing
EC2	Elastic Cloud Computing
ARIMA	Automated Regressive Integrated Moving Average
LR	Linear Regression
NN	Neural Network
SVM	Support Vector Machine
ANN	Artificial Neural Network
LSTM	Long Short Term Memory

ES	Exponential Smoothing
SLAV	Service Level Agreement Violation
ETSA-LP	Ensemble Time-Series Approach for Load Prediction
FEAST	Fog-Enabled Auto-Scaling Technique
GCP	Google Cloud Platform
AWS	Amazon Web Services
AI	Artificial Intelligence

Chapter 1

Introduction

Cloud Computing has emerged as the latest paradigm that offers a dynamic and extensible infrastructure for computing and storage. It offers a huge variety of resources on-demand to cater to the computational obligations of IoT applications such as healthcare, smart cities, and so on. However, certain issues need to be addressed to efficiently utilize this technology, such as scalability, fault tolerance, reliability, security, privacy, etc. Auto-scaling is the process of automatically scaling up and down the resources so that the load on the system can be managed.

In the Cloud, it is recommended to acquire the resources earlier than the time when the load increases. This consequence would be probable only if the the future load can be predicted. If the load exceeds the defined value on a host, it will lead to performance degradation. One way to address the host load imbalance is to leverage the capabilities of the virtualization technology. The improvement in performance on a host can be achieved by switching idle nodes/VMs to low-power VMs. CPU utilization-based host load detection algorithms are able to provide an accurate prediction for autonomic Cloud applications.

This chapter exhibits an overall glimpse of the thesis and presents Cloud computing, several IoT domains, their evolution, and paradigms, in conjunction with the auto-scaling of different resources. The section also discusses how cloud computing is a viable platform for executing IoT-based applications. Further, it provides an overall view of the fundamentals of auto-scaling and the motivation to propose an autonomic approach based on resource usage prediction for Cloud-based IoT applications. The chapter ends with the thesis organization and its pertinent contributions.

1.1 Cloud Computing: An Overview

Cloud Computing is trending these days as it devises various impressive features such as pricing-per-use, broad network access, reliability, scalability, energy efficiency, and on-demand availability of computing resources [1]. Additionally, it also allows a reduction in operating costs as a service benefactors do not have to offer capacities conferring to peak load. Figure 1.1 depicts the main aspects of the Cloud: essential characteristics, deployment models, and service models.

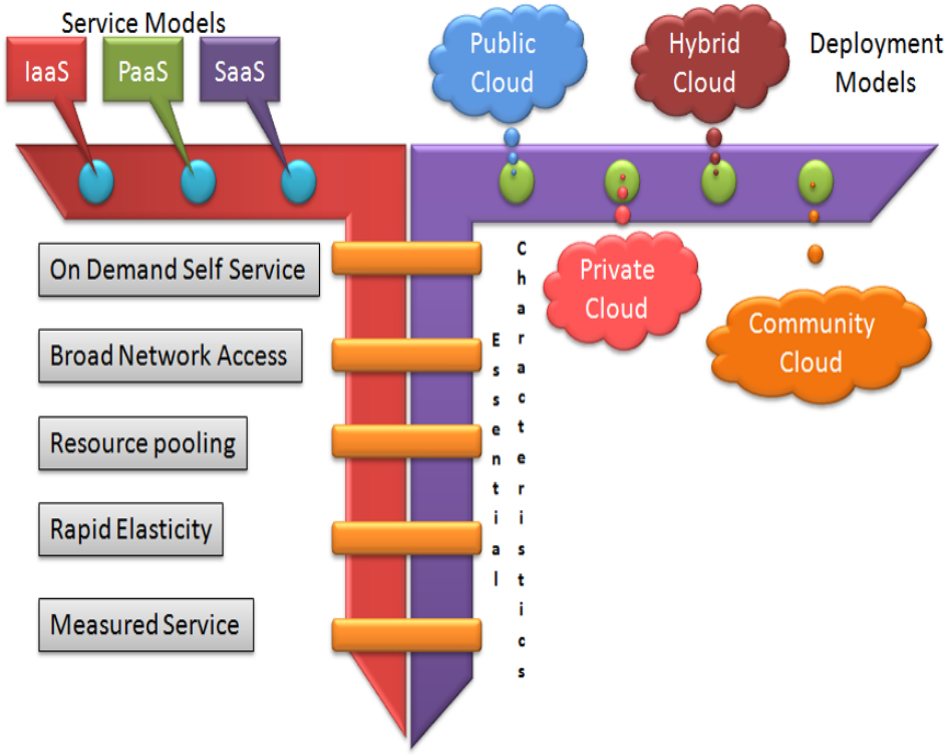


Figure 1.1: Cloud Computing Paradigm

Along with various characteristics, Cloud Computing creates an innovative mode of planning, evolving, testing, installing and preserving applications on the Internet. Nowadays, Cloud Computing services are being utilized by various applications which are listed below:

- Business applications (telecom, e-commerce etc).
- Natural Language Processing (machine translation of one language to another).
- 3D applications (Mathematical simulations, DNA simulations etc).
- Image Processing
- IoT applications (Healthcare, smart cities, smart energy etc).

Clouds have been premeditated for running business applications and web applications but it also can satisfy the high-performance computing demand of IoT applications and that too in a reasonable time. It allows scientists to scale up and scale down the infrastructure as per the application requirements [2]. For the fulfillment of these responsibilities, the Internet of Things (IoT) provides extensive opportunities and applications such as smart grids, smart cities, intelligent transportation systems, e-health and so on. Thus, IoT and Cloud Computing are merged for the overall growth and organization of such applications to meet the current and future needs of end-users.

1.2 IoT-based Cloud Applications

The Internet of Things (IoT) standard is centered on intellectual and automatic configuration of nodes or things incorporated in a dynamic infrastructure and universal network [3]. On the other hand, Cloud Computing is a much more mature technology with indefinite competencies in consent of storage, power and processing. Cloud in integration with IoT gave rise to innovative smart services and applications that have intense effect in everyday life. Numerous applications took advantage when there is a need to interchange information within themselves and also while sending this information for Cloud [2]. Few of these applications are like in the field of healthcare, smart cities and homes, automotive and logistics, smart energy, etc.

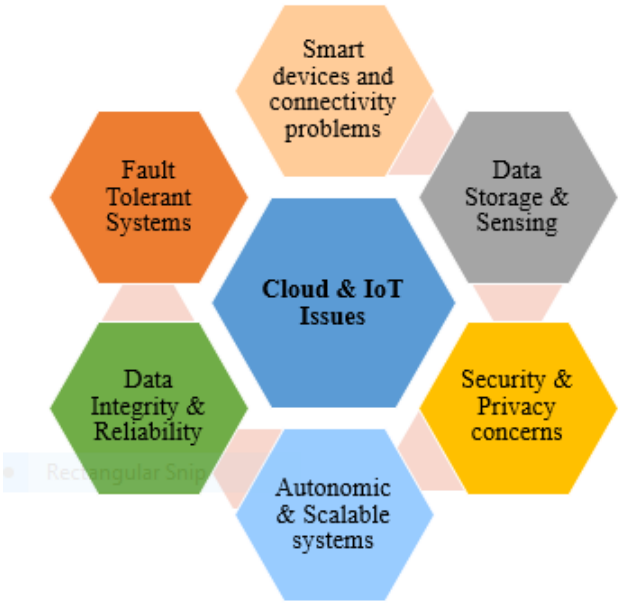


Figure 1.2: Imminent issues with IoT-based Cloud applications

There are many more such applications exist which levies the effectiveness of Cloud Computing in their development. At the same time, it is also reviewed that complex Cloud scenarios forces numerous issues for each IoT application such as scalability, reliability, heterogeneity, security, and privacy, as shown in Figure 1.2. Scalability in autonomic systems is one of them which needs to be resolved in the various IoT applications including healthcare, smart cities, automotive, and smart mobility. Nowadays, various IoT-based Cloud applications are getting an advantage from the auto-scaling feature in which resource consumption can be automatically scaled up and down by Cloud service provider [4].

1.3 Auto-Scaling in Cloud

Cloud Computing with system scalability features permits customers to access vast and elastic resources on-demand. The term scalability can be defined as the concept that signifies the capability of a system to accommodate an increasing number of elements or resources to process cumulative needs effectively [5].

There are considerably three ways of scaling in Cloud: horizontal scaling, vertical scaling and auto-scaling [6]. The horizontal scaling, also referred as scaling in/out, allows the resource units to be added or released as needed and replica of server becomes resource unit i.e. functioning on a VM. On the other hand, vertical scaling or scaling up/down deals with changing system resources allocated to an already functional VM, e.g., increasing or decreasing memory or CPU power allocated [7]. The resource scaling with minimum human intervention is known as auto-scaling.

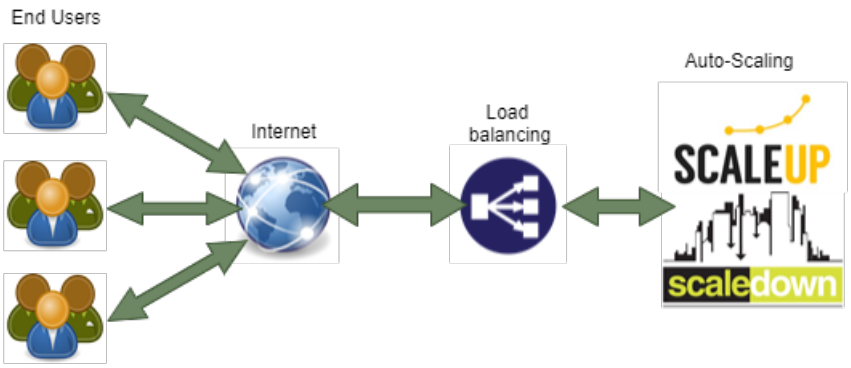


Figure 1.3: General process of Auto-Scaling

Figure 1.3 shows the standard procedure for autonomic scaling. The user's requirement occasionally changes, making the number of operating servers vary and the load oscillate. The need for load prediction in auto-scaling is that a single-tier or multi-tier application is

always scaled based on the incoming load [8]. The general scenario of resource allocation in auto-scaling has been depicted in Figure 1.4. Due to the rise in requests, the auto-scaler chooses whether to initiate more VMs or append resources to the existing VMs. Contrarily, in Figure 1.5, when the number of requests has decreased, the auto-scaler releases some resources from each component by either horizontally halting a few VMs or vertically eliminating specific resources from existing VMs.

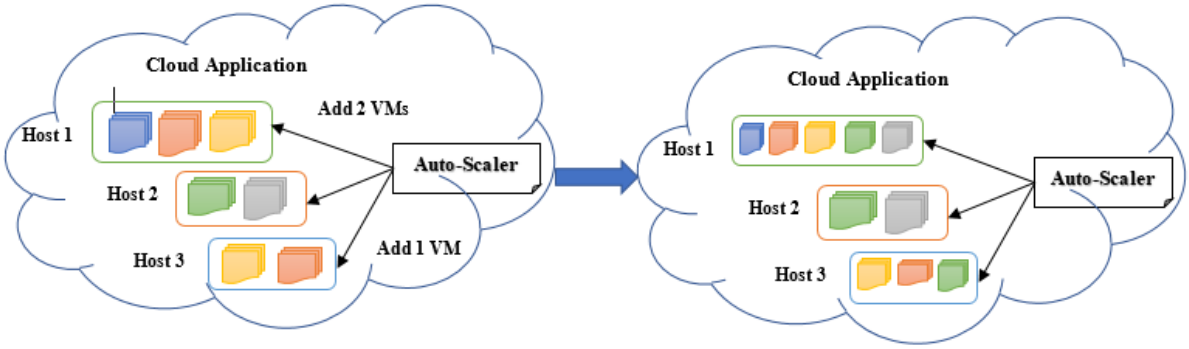


Figure 1.4: Resource Allocation with Auto-scaling: Scale In

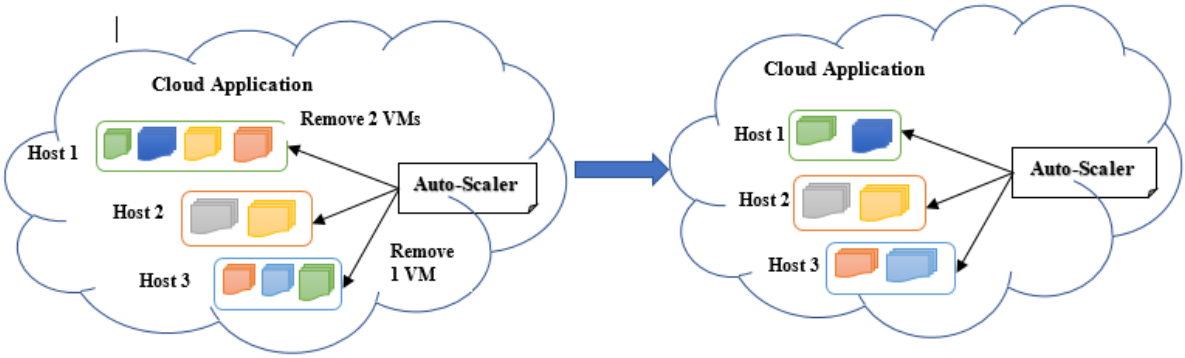


Figure 1.5: Resource Allocation with Auto-scaling: Scale Out

Auto-scaling automates the growth or reduction of system capacity available for use by tenants and is a commonly desired feature in various IoT-based Cloud applications discussed above. The major intention of an auto-scaling system is to fine-tune the attained resources without human intervention which reduces the cost [9]. The auto-scaling system mainly needs a monitor and a scaling unit component. Every auto-scaling technique requires a sound monitoring and prediction system capable of gathering diverse and efficient metrics about the present state of the system, such as CPU, memory, disk, and bandwidth. Therefore, it is recommended to predict the incoming load in advance. The scaling component will make use of these performance metrics to choose which scala-

bility action to conduct, for instance, eliminating a VM or appending some additional memory.

1.4 Load Prediction and VM Migration in Cloud

The various prediction techniques provide an estimation of the load, performance, and cost of an auto-scaling system. Due to the rise of private and public cloud data centers, leasing virtual machines to host is common [8]. Also, it is necessary to acquire the resources earlier than the time when the load increases. This consequence would be probable only if the future load can be predicted. According to the predictive value, log information can be formulated for reclaiming future idle resources, transforming into the energy-saving mode in advance, or adding resources for the upcoming peak load in advance to certify a stable QoS.

In past years, researchers have increased interest in the information of load prediction centered on exploring and monitoring data [10]. Numerous authors have surveyed and implemented load prediction techniques so far. Though diverse prediction methods have been used to date, almost all of these come under either time-series forecasting or machine learning-based procedures like ARIMA, LR, SVM, Neural Networks, etc. These time-series methods are also beneficial for predicting load on VMs in the Cloud and improving crucial quality attributes [11].

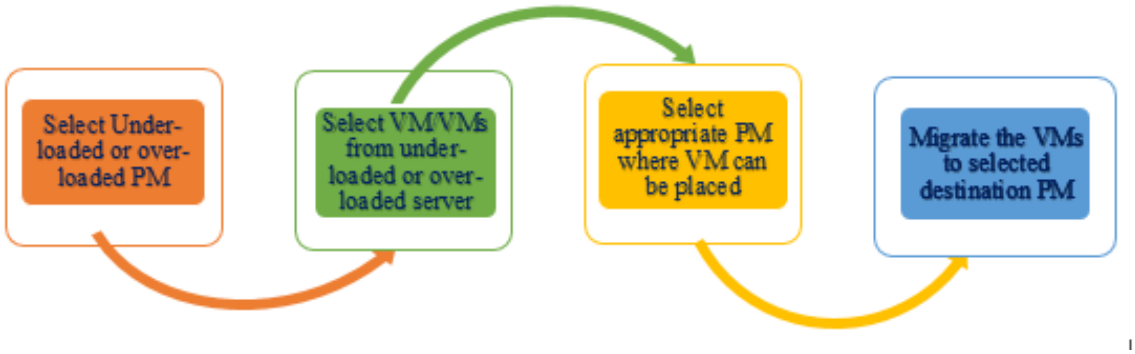


Figure 1.6: Four-Step VM migration process

Virtualization is one of the most prominent features of cloud computing that covenants with the number of resources and users available [12]. By executing this virtual environment, auto-scaling can be easily deployed on any IoT-based Cloud application. It also permits VM (Virtual Machine) migration from one physical machine to another. VM migration is carried out in a directive to reduce the number of running PMs by migrating and consolidating a few VMs into a reduced number [13]. As shown in Figure 1.6,

VM migration is mainly a process of selecting an over-loaded or under-loaded PM and choosing appropriate VM(s) to be migrated onto the target PM.

The most challenging task in the VM migration process is to select a suitable host because the wrong host selection may lead to a repetition of the migration process, resource wastage, increased overhead of machines, and more time, power, and energy consumption [14]. To avoid these circumstances, various VM migration techniques have been proposed so far. However, few researchers have implemented VM migration techniques for auto-scaling, which leads to improvement in the QoS parameters crucial for migrating VMs such as downtime, total migration time, etc. Therefore, there is a need to design and validate an efficient auto-scaling system for load prediction using VM migration for easy deployment of IoT-based Cloud applications. Moreover, cloud providers can optimize resource provision, increase resource utilization, and attain the maximum advantage with an efficient auto-scaling strategy.

1.5 Research Motivation

The Cloud tenants often just pay for the resources they utilize, avoiding the shortcomings and overhead of any idle entities. As a result, it is necessary to keep the system scalable automatically. Therefore, an auto-scaling strategy is required, in which resources can be automatically scaled up or down according to the required resource utilization. Researchers made an effort to deploy the best auto-scaling policies possible for the cloud. Still, an efficient auto-scaling technique is needed to enhance scalability and attain essential QoS metrics in IoT-based Cloud applications. The following are the motivations for this research work:

- Due to dynamic resource requirements in IoT-based Cloud applications, it is recommended to predict the system's load prior and maintain the scalability automatically.
- To predict workload, a dynamic load prediction model is required to gather and monitor different utilization parameters such as CPU, memory, disk, *etc.*
- For enhancing dynamic scalability and prediction accuracy, time-series forecasting techniques can be deployed such as Support Vector Machines (SVM), Neural Networks (NN), Exponential Smoothing (ES), *etc.*
- An IoT-based Cloud application may have over-utilized or under-utilized VMs from time to time. In this unknown forthcoming situation, it is recommended that the VMs be migrated as per the requirement.

- An auto-scaling approach could be deployed to validate various QoS parameters such as CPU utilization, accuracy, response time, scalability overhead, *etc.*, which could improve performance for an IoT-based Cloud application.

1.6 Thesis Contributions

The pertinent contributions of the thesis are mentioned below:

- A detailed survey has been conducted to explore various IoT-based Cloud applications, such as healthcare, smart homes, smart cities, automotive, smart energy, *etc.*

- A comprehensive investigation has been conducted to study the existing load prediction approaches for cloud resource usage prediction and also different migration policies have been surveyed along with their merits and demerits.

- Auto-scaling techniques, including threshold-based, reinforcement learning, control theory, time-series forecasting techniques, *etc.* have been examined for autonomic scaling in Cloud environment.

- An Ensemble-based framework ETSA-LP has been designed and implemented, integrating various time-series analysis techniques for predicting CPU and memory utilization. This model will also provide a dynamic and effective solution to preprocess, collect, analyze, and store large amounts of data for IoT-based Cloud applications.

- An auto-scaling approach for host load prediction has been developed using the VM migration approach for the Cloud environment. It will automatically scale up and down as per the resource requirements of different applications. It would also be capable of improving the overall scalability of the system.

- Finally, a Fog-Enabled Auto-Scaling Technique (FEAST) has been proposed and deployed as a case study to predict resource usage concerning CPU, memory, disk, and network utilization using the LSTM (Long Short-Term Memory) model. This will help avoid over and under-provisioning problems and simultaneously satisfy SLA and QoS requirements.

1.7 Thesis Organization

The introduction to this thesis is presented in ***Chapter 1***. The remainder of the thesis is structured as follows:

Chapter 2 Literature Survey:

Chapter 2 presents a detailed survey of the existing auto-scaling techniques which can be deployed on the IoT-based cloud applications. Also, the metrics to fulfill the QoS requirements are briefly discussed. Besides, the existing load prediction techniques are surveyed and compared on the basis of different parameters such as time, cost, power, SLA violation, scalabilty, CPU load, memory usage and accuracy. Additionally, an extensive survey of the existing VM migration approaches is presented along with the problems solved and the platforms used. Based on the gaps prevailing in the existing literature, the problem formulation and objectives of this thesis are sketched. This chapter is derived from:

- **Shveta Verma** and Anju Bala, *"Auto-scaling techniques for IoT-based cloud applications: a review"*, Cluster Computing, Springers, 24, 2425–2459, 2021. [**SCI Indexed, Impact Factor - 4.40**]

Chapter 3 ETSA-LP-An Ensemble Time-Series Approach for Load Prediction in Cloud:

Chapter 3 This chapter emphasizes deploying a dynamic and autonomic load prediction framework. In this chapter, an Ensemble Time-Series Approach for Load Prediction (ETSA-LP), which integrates various time-series analysis techniques for predicting CPU and memory utilization, has been proposed. To evaluate the efficiency of the proposed approach, a series of experiments on Google and PlanetLab traces have been conducted in a real Cloud environment. The results have been compared according to different performance metrics and models having determined accuracy and the minimal error rate has been selected as the best among others. The proposed ensemble approach gives the best performance over the existing models by showing remarkable accuracy improvement and reducing the error rate and execution time. This chapter is derived from:

- **Shveta Verma**, and Anju Bala *"ETSA-LP: An Ensemble Time-Series Approach for Load Prediction in Cloud"*, Computing and Informatics, 2023. [**SCI Indexed, Impact Factor - 0.972**]

Chapter 4 Efficient Auto-Scaling for Host Load Prediction through VM Migration in Cloud:

Chapter 4 In this chapter, an efficient auto-scaling approach for predicting host load through VM migration has been proposed. The ensemble method using different time-series forecasting models has been proposed to forecast the approaching workload on the host. Based on this predicted load, different algorithms have been devised to detect over-utilized and under-utilized hosts and VMs have been migrated. The designed approach has been validated by experimentation on a real-time Google cluster dataset. The proposed technique significantly improves average CPU utilization and reduces over-utilization and under-utilization. It also minimizes response time, SLA violations, and the slighter number of migrations and scaling overhead. This chapter acquired the content from:

- **Shveta Verma**, and Anju Bala, *"Efficient Auto-Scaling for Host Load Prediction through VM Migration in Cloud"*, Concurrency and Computation: Practice & Experience, Wiley Publications, e7925, https://doi.org/10.1002/cpe.7925 [**SCI Indexed, Impact Factor - 1.831**]

Chapter 5 Verification and Validation of the proposed auto-scaling approach:

Chapter 5 This chapter elaborates on the testing and validation of the proposed approach using the Google Cloud Platform (GCP) and AWS (Amazon Web Services). The chapter also provides the results attained after implementing the proposed approaches. Firstly, the outcomes of ETSA-LP (Ensemble Time-Series Approach for Load Prediction) for predicting resource usage in terms of CPU and memory utilization are shown. The performance of the approach is validated based on error rate, accuracy and total execution time. Finally, the results of the proposed auto-scaling approach are validated on the GCP and AWS. The proposed approach outperforms the existing ones regarding response time, number of migrations, scaling overhead and SLA violation rate.

To validate the effectiveness of the proposed auto-scaling approach, the case study of a fog-based IoT application has been employed. A Fog-Enabled Auto-Scaling Technique (FEAST) has been proposed that predicts resource usage concerning CPU, memory, disk, and network utilization using the LSTM model. Comparing the suggested method to the current fog-based auto-scaling techniques, the experimental findings demonstrate that it improves cost, resource utilization, SLA violation, response time, and delay violation. The content of this chapter is derived from:

- **Shveta Verma**, and Anju Bala, *"Efficient Auto-Scaling for Host Load Prediction through VM Migration in Cloud"*, Concurrency and Computation: Practice & Experience, Wiley Publications, e7925, https://doi.org/10.1002/cpe.7925 [**SCI Indexed, Impact Factor - 1.831**]

- **Shveta Verma** and Anju Bala, "FEAST: Fog Enabled Auto-Scaling Technique for IoT-based Cloud Application", *Automated Software Engineering*, 2024**[SCI Indexed, Impact Factor - 3.4]**. [Communicated]

Chapter 6 Conclusion and Future Directions:

This chapter summarizes the conclusions drawn from the thesis along with the possible future directions.

Chapter 2

Literature Survey

Cloud Computing with system scalability feature permits customers to access vast and elastic resources on-demand. The term scalability can be defined as the concept that signifies the capability of a system to accommodate an increasing number of elements or resources to process cumulative needs effectively. Auto-Scaling systematizes the growth as well as reduction of system capability that is accessible for use by tenants and is a frequently preferred feature in IoT-based Cloud applications.

As the demand for cloud services is increasing, providing resources efficiently based on the application's resource usage requirement has become essential. Also, to balance the load, the resources must be scaled autonomically. So, an extensive analysis of auto-scaling techniques has been done in this chapter to understand existing techniques' key benefits and drawbacks. Also, the literature has been studied to carry out load prediction and VM migration proficiently along with Quality of Service (QoS) requirements.

This chapter investigates the various auto-scaling techniques that can be deployed on cloud-based IoT applications. It discusses the existing techniques that can predict applications' resource requirements in general. The state-of-the-art approaches for VM migration and QoS metrics are also discussed. In the end, the chapter recapitulates the gaps found in existing literature and lists the thesis objectives.

2.1 IoT-based Cloud Applications

The Internet of Things (IoT) standard is centered on the intellectual and automatic configuration of nodes or things incorporated in a dynamic infrastructure and universal network. It embodies one of the utmost turbulent technologies that enable abundant and persistent computing states. Alternatively, Cloud Computing is an emerging technology with indefinite competencies in the consent of storage, power and processing.

Cloud can provide optimal features to enforce IoT service management as well as applications that can feat information produced by them [2]. Cloud in integration with IoT gave rise to innovative smart services and applications that have an intense effect in everyday life. For instance, healthcare applications stimulate and coordinate the administration of medical data, an intelligent transportation system that can improve traffic management and decrease road traffic accidents, smart energy which can improve the efficiency and reliability of power supplies, smart cities, homes, etc [15]. Figure 2.1 describes the application scenario driven by Cloud and IoT integration.

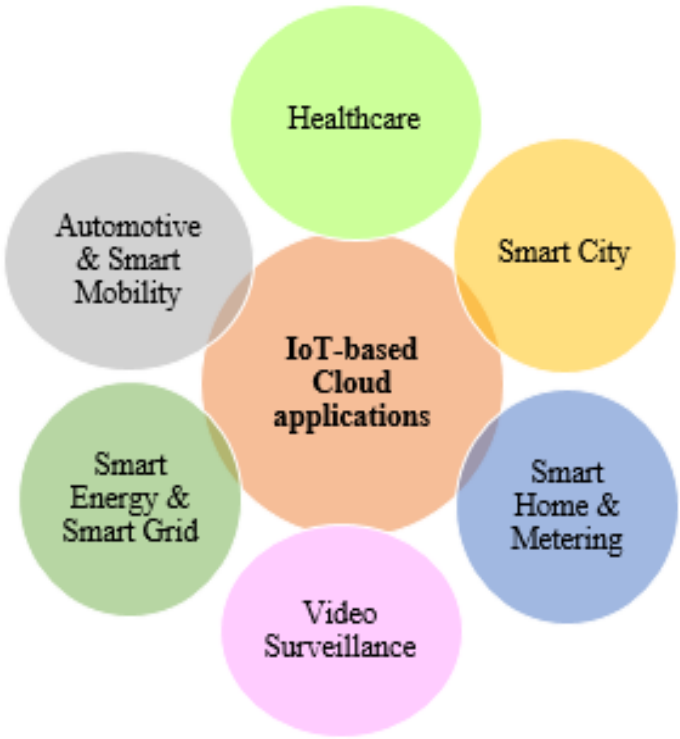


Figure 2.1: IoT-based Cloud Applications

(*i*) **Healthcare**

Various Cloud services, smart devices and mobile Internet are continuously contributing to the efficient invention of healthcare and enable cost operative, well-organized, timely and superior pervasive medicinal services. The implementation of Cloud in healthcare contributes to the generalization of technical details, exterminating the requirement for expertise and to regulate the technology. setup [2]. To develop an ambitious healthcare framework in Cloud, the electronic as well as a medical field need demonstrative and autonomous gadgets, sensors that should be capable enough to cooperate with machine learning and AI to provide personal administration and satisfaction [16].

Through the integration of Cloud and IoT, it becomes very facile to collect patient's dynamic medical information through a set of sensors, deliver and share the electronic health records on Cloud for storage and processing, and proper information management collected from sensors, wearable and implantable devices [17]. One of the examples of using advance IoT functionality using Artificial Intelligence (AI) is the monitoring of child's obesity by placing sensor on the chest of a child to collect information about food diet, physical activities or other environmental and mental factors. A concept of Body Sensor Networks (BSN) has also been proposed in healthcare domain these days that is capable to store, process and analyze huge amounts of relative data in a scalable manner. As our primary contemplation is on accurate health surveillance, cloud-based healthcare application should be time-critical and cost-effective. For instance, RFID chips can be inserted in patients for regular and timely health monitoring with inferior medical costs [7]. Nowadays, various healthcare applications are being deployed on Cloud, but still, there are some common issues exist including interoperability, scalability, security, coursing Quality of Service (QoS) and cumulative storage [2], [18].

(*ii*) **Smart city**

IoT and Cloud together offer a common platform for succeeding Smart-City services by attaining information from diverse sensing infrastructures, retrieving each and every geo-location and showing information in a constant mode [2]. Most of the proposed resolutions suggested practicing Cloud Computing to support the detection, linking and assimilation of sensors and actuators capable of supporting global connectivity along with real-time applications for smart city services. It enables intelligent illumination system in cities, automatic weather reporting systems, sharing of environmental and sensor data between different cities, better informed decision making process and empowers citizens for participation and integration through RFID sensors and geo-tagging. A smart city

IoT framework must include APIs for sensing and stimulating a Cloud platform that offers autonomic, scalable and prolonged storage, processing, management and control of real-world sensing devices [19].

Smart cities are making daily life easier by providing services such as smart municipal facilities, smart power and energy, smart traffic system, smart agriculture and so on. As per IBM, a smart city application should possess the three main characteristics: first is the instruments/sensors should be reliable and time-critical, second, accurate and well-interconnected devices and third, every service should be capable to intelligently monitor the status. Apart from these features, still some concerns are to be taken care of in these applications such as security, privacy, elasticity and real-time interfaces.

(*iii*) **Smart home and smart metering**

IoT devises great applications in home situations that includes diverse embedded devices that assist the automation of mutual inter-house events. Cloud can be considered as a preeminent platform in these applications for building elastic applications with elite code and therefore transforming home automation into a minor chore. It is also important that household devices must be web-enabled, standardized and have uniform self-involvement [2].

Therefore, management and control of such policies should be rendered by installing extra-potent computing devices for implementing complex functionalities and moderating the volume and frequency of interactions with the Cloud. Few examples of the smart home are automatic on/off electrical appliances, sensor-based lock/unlock doors, monitoring power usage of devices, intelligent remote control, voice-based controlling, smart air-quality adjustment and even observing of home is possible through webcams from remote locations [20].

Now a days, smart lighting systems attracted much growing attention from the research community and proved to save the energy consumed up to 45%. For such scenarios, Cloud enables flexible home automation with only a few lines of code, and provides necessary resources for tasks beyond the scope of local networks. Similarly, examples of smart metering include automatic monitoring devices for electricity, water supply, sensor-based gardening tools and so on [3]. For the IoT industry to create this much ease to customers, the smart home and smart metering applications should possess intelligent recognition of appliances, frequent remote control and automatic management of energy consumption.

(*iv*) **Automotive and smart mobility**

Cloud, as an evolving technology, is also capable to provide fruitful solutions to renovate transportation and automobile services such as intelligent parking systems, vehicular clouds, increased road safety, less traffic congestion. These emerging IoT application provides a promising solution to daily life transportation problems and also offer many business benefits [21]. Examples of newly deployed practices are smart car navigation systems, voice assistance systems, autonomic vehicle sensing, Internet of Vehicles (IoV), and V2V (Vehicle to vehicle) exchange of information.

The IoT framework should aim at providing real-time, secure, scalable and on-demand services to tenants through hybrid or multi-clouds in order to increase storage, processing and computing capabilities. The main characteristics of these applications include intelligent control over vehicles, devices with enough storage and processing capabilities, lesser costs, flexibility, etc, [22]. The assimilation of Cloud technologies with Intelligent Transportation Solutions (ITS) represents an encouraging prospect to handle recent challenges such as the enormous vehicles and their frequently changing number finally leads to difficulty in achieving system scalability [23]. Also, vehicles running at changing speeds cause an impact on performance and reliability along with QoS [24].

(*v*) **Smart energy and smart grid**

IoT and Cloud can be successfully fused together to offer smart organization of energy dispersal and consumption. For smart energy, computational jobs should be accurately distributed among devices dealing with the Cloud to handle more complex and comprehensive decisions. Similarly for smart buildings and grids, the energy alternatives and well-suited usage can be provided by incorporating system information into Cloud, though providing self-configuring and ideal electricity quality [25]. It also enables two-way communication among customers to place electricity requirements through advance sensing, processing and networking capabilities [26]. For example, in our homes, electricity and power consumption can be automatically monitored. For deploying smart energy applications, the main features should be intelligent monitoring, optimization and transmission of energy as per generation and demand, efficient image and video processing, self-healing sensor devices, etc. At the same time, the issues related to security and privacy for Smart Grid also rise [27].

(*vi*) **Smart Video Surveillance**

Cloud and IoT for the perspective of smart video surveillance contribute to easy and efficient storage, management, processing internet in a load-balanced, scalable, and fault-

tolerant manner [2]. The important characteristics of video surveillance applications should be having minimum bandwidth, high security and privacy, high storage media and video analytics, heterogeneity and intelligent monitoring of video content. Some of the challenges mainly considered for these applications are the unfeasibility of using any camera device due to the restricted flow of the supporting technology and tools.

2.1.1 IoT-based Cloud applications: Challenges

The various IoT-based Cloud applications discussed so far shows how the integration of Cloud with IoT offers several features and nurtures the enhancement of these applications. But these applications also impels several challenges for complex Cloud such as reliability, heterogeneity, scalability, security and privacy [2]. Table 2.1 depicts the various challenges imposed by each application discussed above. When the migration of critical IoT applications is performed on Cloud, new issues arise due to the ignorance of some important parameters such as complexities of hardware, SLAs, information about physical storage, the privacy of sensitive information, untimely streaming of huge amounts of data, etc. The various challenges imposed by IoT-based Cloud applications have been discussed here:

(i) **Privacy and security**

It becomes one of the major challenges to maintain security and privacy of IoT applications these days. The reasons behind these sometimes the data is hacked by hackers or third-party users and the critical sensor-based information is manipulated by introducing some malware threats. The solution for these issues is to restrict access to critical and sensitive data and also strict policies should be applied for maintaining privacy at the user level [18].

(ii) **Reliability**

The IoT applications use various device and sensor data for critical decision-making such as healthcare, smart homes and cities and intelligent traffic monitoring. There is a need for these devices to communicate in an interconnected, completely flexible and persistent paradigm. But due to the availability of limited resources in some of the IoT devices, it becomes critical to maintain reliability [23]. To design and implement a reliable IoT framework, the detection of inconsistencies in the application or device must be made and rectified prior which will boost the robustness of the system.

(*iii*) **Scalability**

In some cases, IoT services and applications are not skilled enough to work proficiently in small and large environments concurrently which leads to poor scalability between the user and the things. To improve scalability, the framework must be designed to be adaptable in small as well as the large environments, able to handle multiple data storage services and autonomic [4].

(*iv*) **Performance**

The performance of any IoT application depends on the robustness, fault-tolerance, time-critical, interoperable, scalable and self-organized features of the system. The higher these parameters, performance will automatically be healthier and also results in cost optimization [7].

(*v*) **Heterogeneity**

The complex IoT-based Cloud applications should include different heterogenous devices to improve the quality of the overall system. It also has an effect on computational costs and energy consumption and also needs high computing devices in order to perform well in large-scale environment [27].

(*vi*) **Data streaming**

Most of the IoT-based Cloud applications are not able to cope with the geo-distributed data these days. In other words, to automate real-data flows, the data streaming capabilities must be incorporated with these applications [15]. Therefore, it is recommended to integrate the Cloud and IoT applications and deploy them such that it results in a continuous flow of data generated by various sources (e.g. e-healthcare in a smart city driven by autonomous vehicles). Also, in data stream processing, the challenge is to use infinite data as finite data in finite time and space [18].

As depicted in the Table 2.1, challenges that need to be pertained and resolved from the perspective of each IoT-based Cloud application has been highlighted. Scalability is one of the issues that must be resolved in applications including healthcare, smart cities, automotive and smart mobility. Nowadays, various IoT-based Cloud applications are getting an advantage from the auto- scaling feature in which resource consumption can be automatically scaled up and scaled down by Cloud service provider. Also, there are numerous auto-scaling techniques present for effective deployment in the Cloud such as threshold-based policies, queuing theory, control theory, *etc.*, which are elaborated in the next section.

Table 2.1: Challenges related to IoT-based cloud applications

Applications	Privacy	Reliability	Security	Performance	Heterogeneity	Data Streaming
Healthcare [17],[18]	Yes	Yes	Yes	Yes	Yes	Yes
Smart City [19]	Yes	Yes	Yes	Yes	Yes	Yes
Smart Home & Smart Metering [20]	No	Yes	No	Yes	Yes	Yes
Automotive & Smart Mobility [2]	No	Yes	Yes	Yes	Yes	No
Smart Energy & Smart Grid [20]	Yes	Yes	Yes	Yes	Yes	Yes
Video Surveillance [2]	No	Yes	Yes	Yes	Yes	Yes

2.2 State-of-the-art: Auto-Scaling

Auto-scaling is a crucial component in Cloud that refers to the ability to dynamically allocate and release computing resources in response to changing resource demands [8]. Auto-scaling automatically determines the appropriately allocated resources to meet QoS objectives and Service Level Agreement (SLA).

2.2.1 Auto-Scaling in Cloud

To satisfy the dynamic requirements of Cloud users, auto- Scaling can be applicable to all the service levels of Cloud Computing-IaaS, PaaS and SaaS, which can be further helpful in the easy deployment of IoT-based Cloud applications. Figure 2.2 represents the taxonomy of Cloud auto-scaling at different levels and Table 2.2 depicts a list of different Cloud solutions with auto-scaling features at these three levels.

(*i*) Auto-scaling at IaaS level

Auto-scaling at IaaS level focuses on completing two main requirements, first is the application of prominent vendor techniques and secondly, developing auto-scaling policies for explicit needs of users [28]. It addresses specific demands and techniques such as virtualization, workload monitoring, hybrid/multi-clouds and self-scaling framework. For

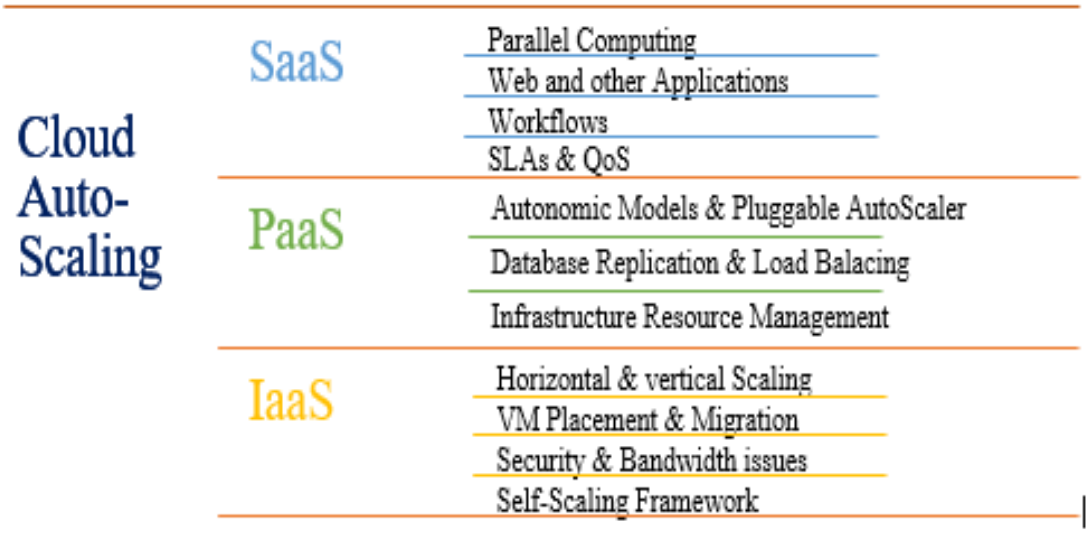


Figure 2.2: Cloud auto-scaling at different levels

instance, Amazon is a prevailing Cloud supplier with its auto-scaling mechanism for the IaaS level.

(*ii*) Auto-scaling at PaaS level

The first main component of PaaS level auto-scaling is infrastructure resource management with the help of database replication or load balancing. At the PaaS level, auto-scaling is also dependable on autonomic models (self-repair, self-optimize and self-monitor) and also helps in coordinating resource consumption of a system [6]. For example, Microsoft Azure works on PaaS level with a built-in auto-scaling mechanism.

(*iii*) Auto-scaling at SaaS level

On the topmost level Saas, auto-scaling along with Cloud offers services based on the pay-per-use model and handles concerns of handling resources according to inconsistent load along with meeting SLAs and QoS requirements. Various Hadoop services, web and other applications work on SaaS level to provide auto-scaling capabilities [10].

Table 2.2: Existing cloud solutions with auto-scaling feature at different service levels

Service providers	**Service level**	**Policy**	**Application**
Amazon EC2 [6]	IaaS	Auto-Scaling, threshold based	Web Services
Microsoft Azure [29]	PaaS	Manual	Diagnostic mechanisms
Paraleap (based on Azure) [29]	PaaS	Auto-Scaling	Management APIs, embedded auto-scaling

Table 2.2: Existing cloud solutions with auto-scaling feature at different service levels (cont.)

Service providers	**Service level**	**Policy**	**Application**
CloudWatch (Amazon) [30]	IaaS	Auto-Scaling	Metrics-based, Load Balancing
GoGrid [9]	IaaS	Manual	APIs
RightScale (GoGrid & EC2) [31]	IaaS	Auto-Scaling	Alert System, Metric-based
RackSpace [27]	IaaS	Manual	APIs
Enstratius (Rackspace) [6]	SaaS	Auto-Scaling	Configuration management tools, on-premises deployment
Scalr (RightScale & EC2) [29]	IaaS	Auto-Scaling	Web Applications
OpenStack [32]	IaaS	Auto-Scaling	Load Balancing
Google Compute Engine [33]	PaaS	Auto-Scaling	Monitoring, XML supportive
Claudia [6]	-	Auto-Scaling	Flexible, Open Source
Amazon Elastic Beanstal [6]	PaaS	Auto-Scaling	Resource Provisioning
Hadoop [34]	SaaS	Auto-Scaling	Parallel web applications
CloudScale [6]	IaaS	Auto-Scaling	Resource prediction, error correction, VM migration, scaling
Flexiscale [35]	IaaS	Auto-Scaling	Load balancing
Snooze [9]	IaaS	Auto-Scaling	Power management, VM consolidation, fault tolerance
Nimbus [29]	IaaS	Manual	Works as a tool
Eucalyptus [29]	IaaS	Manual	Resource management
Open Nebula (CloudStack) [4]	IaaS	Manual	Hosting services, virtualization, security
SmartScale [29]	IaaS	Auto-Scaling	Prediction, workload classification, virtualization management

In general, horizontal and vertical scaling techniques are considered at different resource levels in Cloud. The horizontal scaling adjusts and maintains the number of VM instances on a large scale. Still, it sometimes provokes a waste of virtual resources and takes much time to initiate a VM [9]. Conversely, vertical scaling can scale virtual resources in a few milliseconds, but it is restricted by the number of resources available on the host machine on which the VM resides. Both scaling techniques are suitable for diverse conditions, and an inventive blend of these two may comfort users in discovering an ideal scaling policy

[36]. Numerous auto-scaling techniques present for effective deployment in the Cloud, such as threshold-based policies, queuing theory, control theory, etc., are discussed below, along with their merits and demerits.

2.2.2 Auto-Scaling: Existing Techniques

To fulfill the objective of scalability, Cloud professionals have followed schedule-based and rule-based techniques of Auto-Scaling to attempt the co-ordination between computing necessity and resources, as shown in Figure 2.3.

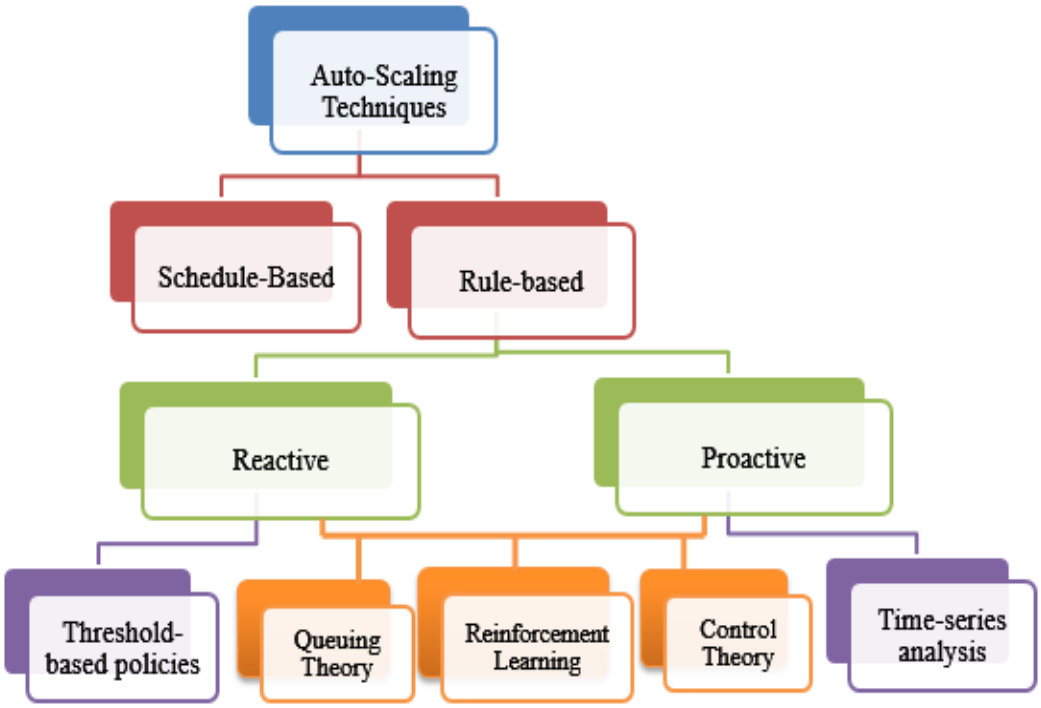


Figure 2.3: Classification of auto-scaling techniques

A Schedule-based methods: Schedule-based methods evaluate the recurring outline of regular workload. In this, scaling measures are physically organized based on time, so that system cannot get used to the unpredicted deviations in the load [6].

B Rule-based methods: Many familiar Cloud providers like Amazon generally recommend rule-based auto-scaling techniques. A rule-based method usually generates rules to identify when and how to scale [29]. To accomplish each rule, the client needs to specify a new state supported by a final variable, e.g., CPU load¿80%. If a given state is achieved, it activates an action for scaling up or scaling- down that is pre-defined, e.g. appending a new VM. The rule-based method can be further

categorized as reactive and proactive algorithms:

- Reactive: The reactive method implies that it responds to system changes, but does not act to them. Threshold-based policies visibly belong to reactive techniques.
- Proactive: Predictive or proactive auto-scaling methods attempt to foresee future prerequisites and accordingly attain or relinquish resources beforehand so that they are ready whenever required [12]. If we reflect on the convention of a specific resource such as CPU or memory as a time series, we can reflect on numerous foretelling substitute. Time-series analysis is entirely a proactive approach.

2.2.3 Threshold-based policies

Threshold-based rules or policies are widely autonomous and used in the commercial auto-scalable Cloud Computing systems [29]. There are two rules for employing a threshold-based auto-scaling: one for scaling up and another for scaling down e.g. if processor time is greater than 70%, then scale up and if processor time is less than 30%, then scale down. These rules generally depend upon a time variable and after each scaling process, a cool-down period (also known as resize calm time) is applied. In this cooling down period, no action is performed means no scaling is applied [37]. It is a reactive technique in which a request is generated for an instance only when a threshold point is reached or aroused. The major shortcoming of the threshold-based policy is the effort required for setting appropriate threshold values [38].

According to the researches carried so far, threshold based policies can easily handle and adapt the number of resources assigned to Cloud applications for executing auto-scaling features [39]. On the other hand, a user requests to choose an appropriate performance variable or combination of variables to fix threshold-based rules. The upper and lower thresholds are key variables to be set for accurate working of these rules. Hasan et al. [40] reflected four thresholds: ThrU (upper threshold), ThrbU (slightly below upper threshold), ThrL (lower threshold) and ThroL (slightly above lower threshold). The time duration, measured in seconds, used for testing strength of performance metric value above/below are denoted as ThrU /ThrL and ThrbU /ThroL. This type of parametric organization can better track the performance metric value tendency.

Table 2.3: Summary of existing threshold-based auto-scaling systems

Ref.	**Scaling Type**	**Technique**	**Proposed Methodology**	**Parameters/ Metrics**	**Platform**
[7]	Both Horizontal & Vertical	Reactive	LS, LSU & LSD algos	Response time,CPU load	IC- Cloud
[37]	Both Horizontal & Vertical	Reactive	Integrated and Autonomic Cloud Resource Scaler (IACRS)	Response time,CPU load	No experimentation
[38]	Horizontal	Reactive/ proactive	dynamic Scoring Algorithm	CPU load, request rate	Custom Simulator
[40]	Both Horizontal & Vertical	auto	Threshold-based	upper threshold and lower threshold, response time	Custom simulator
[41]	Vertical	Reactive	Robust Hybrid aoto-scaling (RHAS)	Response time,CPU load	Custom simulator
[42]	Horizontal	Reactive	Auto Scaling Approach Based on Learning Automata (ASTAW)	Scaling overhead, Threshold values, Bandwidth	CloudSim
[43]	Horizontal	Reactive	Novel Auto Scaling Approach Based on Learning Automata (NASLA)	No of VMs,Threshold values, Scaling overhead	CloudSim
[33]	Not defined	Reactive	System model,VM Allocation algos	CPU utilization, No of VMs, Threashold values	CoMon project, PlanetLab, CloudSim

On the other hand, RightScale's auto-scaling algorithm gives an accompaniment to reactive rules. It is an elementary autonomous process of voting in which an action is taken only if most of the VMs approve to scale up or down otherwise no action is taken. Based on a set of rules, every VM has to vote for scale up or down. After completion of each scaling act, there is a cool down time (resize calm time) where no action can be performed. RightScale suggests 15 min of calm time period because fresh machines normally take 5–10 min to start. Chieu et al. [44] suggested adopting reactive rules centered on number of active sessions and also extended this by adopting the RightScale approach.

Kupferman et al. [45]compared RightScale with other algorithms and made the assumption that RightScale's voting system is extremely reliant on threshold values defined by a user and their workload features. To save costs, they developed a method called smart kill which said that even if the load is low, there is no need to terminate a VM earlier an hour is completed. Beloglazov [32] proposed a method for the consolidation of VMs dynamically based on adaptive threshold utilization and also determined the probability distribution of CPU utilization for Cloud. Lim et al. [46] defined a proportional threshold-based method for an integral controller. The summary of research work related to threshold-based auto-scaling policies is shown in Table 2.3.

As per the literature reviewed about threshold-based policies, it has been observed that the straightforward and native nature of these purely reactive policies creates them very attractive to the users and commercial Cloud providers. But, still there are some of the gaps being identified in this concern:

- The setting of threshold values is a time-consuming process and entails a profound knowledge of workload trends because an inappropriate adjustment can cause oscillations in VM number and thus lead to immoral performance [45].

- CPU load, response time and threshold latency parameters play an important role while applying threshold-based policies. Many researchers have used these policies for auto-scaling in Cloud, but the effectiveness of the rules under burst workloads is still questionable [46].

2.2.4 Reinforcement Learning (RL)

The reinforcement learning approach of auto-scaling is an automatic decision-making technique in which VM representation is anticipated via learning conduct [6]. It interprets the performance framework and strategy of a target application online without any preceding information. In this method, all decisions are approved based on of communication between an auto-scaling agent and a scalable application environment [47]. An agent is a decision-maker that will learn from experiencing the best possible action to be performed for all environment states and to maximize returned reward always. The Markov Decision Process (MDP), Q-learning, parallel learning and neural network algorithms are considered as the current methods to resolve complications in RL approach, as shown in Figure 2.4.

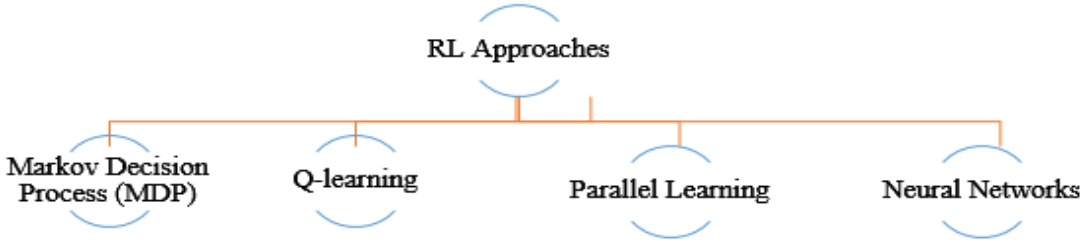


Figure 2.4: Types of RL techniques

(*i*) MDP

The fundamental elements of a MDP can be specified for a single problem in many ways. For instance, a state s is defined as [w; u; p], where w denotes the total number of user requests according to time-period, u represents the number of allocated VMs and p specifies performance in the consent of average request response time, particularly in case of horizontal scaling [48].

(*ii*) Q-learning

Dutreilh et al. [49] proposed a preliminary approximation of Q-learning function to improve the bad performance of RL by updating the value for each state in all iterations.

(*iii*) Parallel Learning

In the parallel learning technique, each mediator does not require to call each state and action every time, as an alternative it can acquire the value of non-visited states from its adjacent agents [50].

(*iv*) Neural Networks

Neural networks can foresee the value of non-visited states and considered a pair of state actions as input and Q-value as output.

Besides these approaches, a different proposal concentrated on vertical scaling which considered a state assumed by the usage of memory and CPU for a VM individually [29]. The probable action to be taken highly depends on the nature of scaling i.e. whether horizontal or vertical. For example: append, eliminate or sustain the number of VMs or increment/decrement the volume of CPU and memory allocated in case of horizontal scaling. As per reward function is concerned, it generally takes the cost of resources and cost evolved due to SLO violations into consideration.

Rao et al. [51] stated that the problem to reduce time of training can be solved either with a policy that imposes numerous states at each step or by applying parallel learning agents. Cooper [52] stated that E-greedy is one of the widely used strategies for neural networks, where arbitrary actions are chosen with a low probability 1-E.

Table 2.4: Existing reinforcement learning based auto-scaling systems

Ref.	**Scaling Type**	**Technique**	**Proposed Methodology**	**Parameters/ Metrics**	**Platform**
[47]	Horizontal	Proactive	VirtRL workflow	No of VMs, Response time, Cost	Olio application
[50]	Horizontal	Proactive	Parallel Q-learning architecture & algorithm	No of VMs, Response time, Cost	Matlab,Amazon EC2, Xen
[51]	Vertical	Proactive	VCONF (RL based virtual machine auto-configuration agent)	CPU, memory usage ,Response time,Throughput	Xen,TPC-C, TPC-W,SpecWeb
[49]	Vertical	Proactive	Unified reinforcement learning (URL) approach	Response time	Xen,TPC-C, TPC-W, SpecWeb
[53]	Horizontal	Proactive	Q-learning,FQL4KE technique	CPU load,response time, network link bandwidth	OpenStack
[54]	Not defined	Both reactive & proactive	SLA-aware and Resource-efficient Self-learning Approach (SRSA)	Bandwidth, Total VM running time, Scaling in/out time	Huawei TCRM
[55]	Not defined	proactive	State-Action-Reward-State-Action (SARSA) algorithm	Total no of VMs, ,cost	Eucalyptus AWS
[48]	Not mentioned	proactive	Auto-Scale model architecture, Neuro-fuzzy based model, MDP	CPU time,Response time, Throughput,	Xen driver interface,RUBiS

As depicted in Table 2.4, the reinforcement learning approach acts as an auspicious technique for auto-scaling with distinct types of performance metrics such as response time, no. of VMs, cost, throughput, etc. RL approaches attempt to perform auto-scaling but without any prior awareness or model of an application. These techniques might seem challenging for users because of the following reasons:

- Inferior performance and Time-consuming: Sometimes initially and during an infeasible online live training period, the performance can be intolerably reduced [52].
- Curse of dimensionality problem: It refers to having large state space means the number of states increases frequently with the number of state variables, resulting in scalability problems [51]. The modest solution is to use a lookup table that stores a separate value for every feasible pair of state and action. But, with the increase in the size of the table, the performance also degrades.

2.2.5 Queuing Theory

Queuing theory is applied to increase capacity by examining and creating decisions on the basis of a queue e.g. queued requests in a load balancer. Traditional queuing theory has been widely implemented to form applications and servers on Internet for the cause to analyze performance parameters like queue length or average request waiting time [6]. Queuing theory uses a mathematical reference model from the precise study of waiting lines or queues. When a client request arrives in the system with mean arrival rate k, it is appended in the queue until their processing. There may be a possibility that one or more servers in the model can exist and will join requests at this arrival rate [56].

Queuing theory can perform estimation of performance metrics only, therefore it has been combined with other techniques such as threshold-based policies, control theory or RL by most of the researchers to solve auto-scaling issues. There are two main drawbacks of applying queuing theory approaches: one is that they require few assumptions which are non-realistic and not valid for real-time premises and secondly they are also not well-suited for the critical type of systems [35]. The concept of queuing networks can be implemented in one of the following types, as shown in Figure 2.5.

(*i*) **Open queuing network model**

This autonomous and parallel model can be constructed as n open networks having one server and a demand level of k/n each. The researchers have used a parallel M/M/1 queuing expression to represent mean response time with exponential smoothing [57].

(*ii*) **Closed-loop queuing network model**

This model represents an application with a finite M number of customers and can be modeled as n autonomous parallel closed networks with one server and M/n customers each. For this, Mean Value Analysis (MVA) can be used to define the mean response time of an application which uses two parameters: Z i.e. average think time and 1/ l i.e.

average service time at a server [56].

(*iii*) **Queue per server**

This approach was implemented with one server and a network of G/G/1 queue. They developed the formula for k using this model and the Little's Law in order to decide peak workload and the number of servers required on each tier. This peak workload can be predicted and improved using histograms and some reactive techniques [58]. The main shortcoming of this approach is that providing resources for peak load ultimately leads to eminent under-use of resources.

(*iv*) **Queue per tier**

This approach is used to represent a single queue for each tier and can be used to configure multi-tier applications.

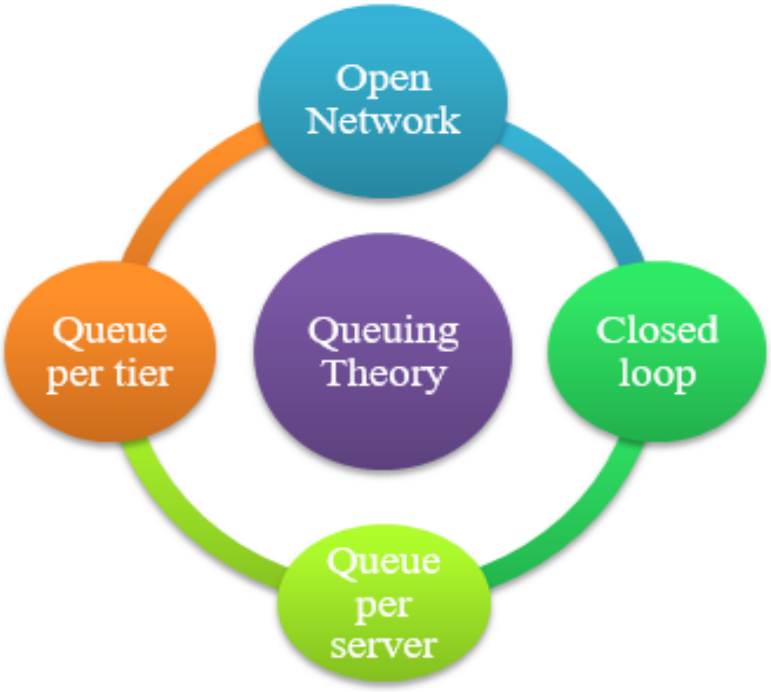


Figure 2.5: Types of Queuing Theory techniques

In previous research works, simple queuing models, as well as, queuing networks have been used to represent applications. The common Internet applications have been usually developed using a simple queuing model and Cloud framework, which can be modeled as a G/G/N queue, where N represents the number of servers. This model can be used to

approximate various parameters such as essential resources required by input workload or mean request response time. This data can be further integrated into diverse techniques e.g. to resolve an optimization problem in some applications [59]. The queuing models with a single queue have this limitation; therefore, the use of queuing networks is recommended to communicate with a single application as well as a complete multi-tier application.

Vilaplana et al. [56] concentrated on the e-commerce application tier in which each server can be modeled as M/GI/1/PS queue. They considered the arrival process of an e-commerce application trace and found that this arrival process can be effectively depicted by a Poisson process (M or Markovian). Zhang et al. [60] used a closed system with a network of queues by considering a limited number of users. This type of model can be well-enlightened using Mean-Value Analysis (MVA). The information, like the number of requests, transactions or service time needed for a queuing model can be acquired by online monitoring. They used a regression-based calculation method to analyze CPU demand on user transactions. The list of reviewed approaches related to queuing theory is enumerated in Table 2.5.

Table 2.5: Existing queuing theory based auto-scaling systems

Ref.	**Scaling Type**	**Technique**	**Proposed Methodology**	**Parameters/ Metrics**	**Platform**
[56]	Horizontal	Both reactive & proactive	Provisioning & Work prediction algorithm	Response time, Peak workloads	RUBiS,RUBBOS, Xen
[35]	Horizontal	Reactive	Poisson process,Application serving service model	Response time,Arrival rate	Monte-Carlo simulator
[57]	Not defined	Proactive	Regression analysis,3-tier e-commerce architecture	Mean value Analysis (MVA), CPU load,No & type of transactions	C++Sim,TPC-W
[59]	Horizontal	Reactive	Cost-Aware Scaling (CAS) algorithm,iSSe architecture	, Cost,Response time,Arrival rate	IC-Cloud testbed, TPC-W
[61]	Both horizontal & vertical	Both reactive & proactive	Transactional Auto Scaler (TAS) architecture, Analytical modeling	Mean value analysis, No of transactions,CPU utilization	Amazon EC2,FutureGrid, TPCC-W, Red Hat's Infinispan Data Grid

Table 2.5: Existing queuing theory based auto-scaling systems (cont.)

Ref.	**Scaling Type**	**Technique**	**Proposed Methodology**	**Parameters/ Metrics**	**Platform**
[62]	Not defined	Proactive	Distributed stream processing systems (DSPS) workflow, Smart city sensor networks	throughput,end-to-end latency	Apache Storm
[63]	Not defined	Not defined	Cloud architecture with Open Jackson network	Response time, bandwidth	Sage 5.3,OpenStack, Apache JMeter
[58]	Vertical	Not defined	Feedback controller,Resource control algorithm	No of CPU, Throughput	VMWare ESX,Zimbra architecture

From the existing research works, it has been concluded that queuing theory is used to find a correlation between incoming and outgoing jobs of a computing system. This elementary approach models a single VM or set of VMs as a queue of requests and helps estimate required performance metrics like response time [59]. A main drawback of these models is that they are very inflexible and must be re-analyzed each time if there are changes in an application or the workload [6]. Therefore, there is still a need to develop an efficient queuing model with flexible and better analytics features.

2.2.6 Control Theory

Control theory helps in automating the organization of different systems on Cloud Computing platforms such as web server, storage systems, data centers and server clusters. Control systems are mostly reactive kind of methods, but some proactive techniques are also present such as Model Predictive Control or a combination of a predictive model with control system [64]. As shown in Figure 2.6, control systems can be classified into three categories: open loop, feedback and feed-forward controllers.

(*i*) **Open-Loop**

Open-loop controllers work on input of target system using its current state and system model. A good example of this is a central heating boiler controlled only by a timer, so that heat is applied for a constant time, regardless of the temperature of the building.

(*ii*) **Feedback controllers**

These controllers use output of the system and are also capable to rectify any difference that occurred from desired value.

(*iii*) **Feed-forward controllers**

This type of controller forecast system performance using a model and respond to the actual occurrence of an error formerly. The key objective of a controller is to preserve output of target system to a preferred level by regulating control input such as the number of VMs [65].

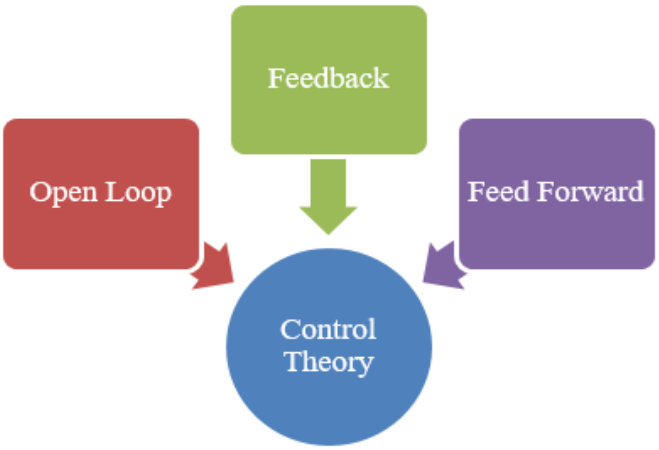


Figure 2.6: Categories of Control Theory techniques

Various researchers have discussed and implemented different types of controllers for control theory such as fixed gain type which comprises a PID controller. Another one is Integral controller that can regulate the number of VMs and is based on average CPU usage [29]. Similarly, the PI controller given by Park and Humphrey is capable to control the resources based on their execution progress as required by batch jobs [64]. Many authors also discussed adaptive control techniques. Padala et al. [65] implemented a Multiple Input Multiple Output (MIMO) adaptive controller to simulate non-linear and timely correlation in the allocation of resources and their synchronized performance.

Bodik et al. [66] proposed a gain-scheduling adaptive controller to implement this performance model and suggested two scaling rate control parameters a and b. Kalyvianaki et al. [67] developed two different SISO (Single Input Single Output) and MIMO controllers based on Kalman filters to control the CPU allocation of VMs. Some researchers also used fuzzy models in control theory-based systems to record workload and required resources as input and output variables respectively. The initial step is to map input as

well as output variables into fuzzy sets of a system [68]. This mapping is denoted by a relationship function that decides a value within the interval (0, 1). Another control system that depends on a rule-based fuzzy model is called a fuzzy controller. Xu et al. [69] implemented a fuzzy controller to regulate the necessary CPU load. Wang et al. [70] followed the same approach to foretell the upcoming resource needs by focusing on the database tier. They also proposed a fuzzy model predictive controller which was based on control theory along with fuzzy rules. Lama and Zhou [71] implemented a neural fuzzy controller that was proficient in the automatic construction of fuzzy rules and their related functions. The summary of the literature related to control theory is listed in the Table 2.6.

Table 2.6: Existing control theory based auto-scaling systems

Ref.	**Scaling Type**	**Technique**	**Proposed Methodology**	**Parameters/ Metrics**	**Platform**
[46]	Vertical	Proactive	ARMAX, SVM regression, Great lake forecasting	Application adaptive parameters,	Xen
[69]	Vertical	Reactive	Self-tuning controller, PI controller	CPU usage, Job deadline, Job progress	5 HPC applications, Hyper-V,Matlab
[64]	Horizontal	Reactive	Proportional thresholding, Prototype control system	CPU utilization, Request rate	Amazon EC2, Hyperic-HQ
[65]	Vertical	Proactive	AutoControl system,MIMO controller	CPU usage, Disk I/O, Response time	Xen platform,OpenSuSE 10.3, Emulab, RUBiS,TPC-W
[66]	Horizontal	Proactive	Simple linear regression,Statistical machine learning, Smoothing spline performance model	No of requests, No of servers,Response time	CloudStone, Amazon EC2, Web 2.0
[71]	Vertical	Reactive	Two-level controller architecture	No of requests,CPU load,reply rate	Java Pet Store,Httperf, Perl,VMWare ESX server
[70]	Vertical	Proactive	Autonomic resource management system	CPU load, Throughput, Bandwidth, Response time	TPC-H,Xen platform,RUBiS

Table 2.6: Existing Control theory based auto-scaling systems (cont.)

Ref.	Scaling Type	Technique	Proposed Methodology	Parameters/ Metrics	Platform
[67]	Vertical	Proactive	Neural fuzzy controller	Resource usage,End-to-end delay	Simulation with 3-tier server cluster
[53]	Vertical	Proactive	Kalman filter Various controllers	CPU usage, Response time	RUBiS, Xen platform
[49]	Vertical	Both Reactive and proactive	Feedback elasticity controller, Hybrid controller	No of requests,Total resource usage	Python simulator
[72]	Not defined	Proactive	PID controller	Bandwidth, Response time, CPU utilization	OpenStack, Amazon EC2

The literature reviewed about control theory concludes that control theory banks upon generating a prototype of an application. The main focus of this technique is to specify a reactive or proactive controller that can fine-tune required resources to an application's requirements automatically. The quality and functioning of a controller depend extremely on an application model as well as on controller itself [65]. Similar to RL approach, it does not require any prior knowledge, but still it can be considered as an effective approach for auto-scaling.

2.2.7 Time series analysis

A time-series process is an organization of data points that are to be measured at constant time intervals [29]. This technique is used in several diverse fields to indicate any deviation in measurement over time such as finance, engineering, economics and bio-informatics, etc. This technique can also be applied to determine patterns in the input workload that are self-repetitive or to predict forthcoming values.

The time-series forecasting methods can be precise to envisage both resource usage and workload prediction. The prospect value is foreseen based on the last few interpretations. Few techniques based on these are Moving Average, Auto-regression, exponential smoothing and various machine learning-based approaches as shown in Figure 2.7.

(i) Averaging methods

Averaging methods are used to remove dissonance or to generate future forecasts. The prediction value $yt+1$ is measured as a weighted average of previous w consecutive values. It can be further categorized as MA, WMA and exponential smoothing. Moving Average

(MA) represents arithmetic mean of previous q or w values. In other words, it allocates equivalent weights 1/w to all observations [11]. Weighted Moving Average (WMA) allocates variated weights to each and every observation. Correspondingly, Exponential smoothing deals with decreasing weights over time exponentially.

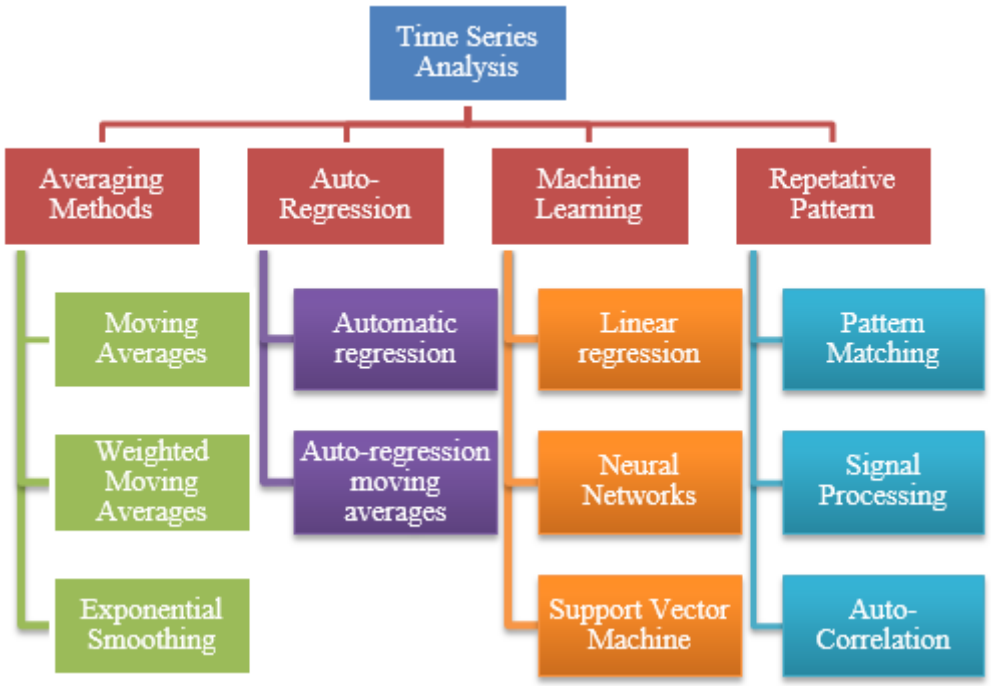


Figure 2.7: Various Time-Series Analysis Methods

(*ii*) **Automatic regression methods**

These are of two types- Auto-regression (AR) and Autoregressive moving averages (ARMA). In AR, the weighting factors are resolute by calculating coefficients of auto-correlation and linear equation solutions. On the contrary, ARMA method combines both moving average and autoregression and the previous inputs and outputs are considered as a base for predicted output [33].

(*iii*) **Repetitive pattern-based techniques**

An extensive variety of methods are used for the creation of repetitive patterns in time-series such as pattern matching, signal processing techniques and auto-correlation. Pattern matching is almost analogous to the string matching process and it permits the inspecting of similar patterns in historical time- series technique [73]. Fast Fourier Transform (FFT) is a signal processing technique that decomposes signal based time-series into diverse frequency modules. In auto-correlation, an input is recurrently shifted and correlation is measured from original and shifted time-series.

(*iv*) **Machine learning techniques**

Machine learning techniques empower the exploration of algorithms that work on computer systems and are proficient to attain complex interactions/patterns from realistic data [10]. More indeed, machine learning includes design, implementation and validation of algorithms that can abstract visions from data regarding the relationships among objects and events, often captured in the form of statistical models. Besides the fruitful application of machine learning to financial applications and e-commerce, it has also been gaining popularity among IoT-based Cloud applications [74].

There has been enormous research in the preceding era on supporting machine learning based techniques to create self-analysis for the applications . Supervised learning, semi-supervised learning and unsupervised learning are the three main classes of machine learning methods [75]. The machine learning techniques are most useful to perform classification and prediction using a variety of statistical models, including decision trees, regression models, neural networks,Bayesian networks, Support Vector Machines (SVM), as depicted in Figure 2.8.

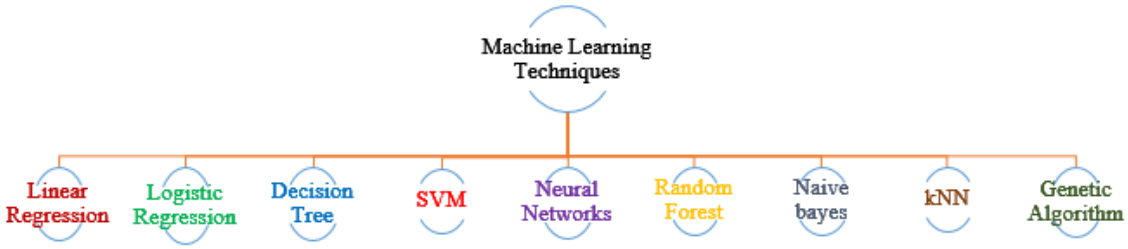


Figure 2.8: Machine Learning Models

As per the literature reviewed, time-series analysis methods have widely opted for both workload and resource usage prediction. While, the simple moving average method can also be applied for the prediction of load/resource, but it results in reduced performance [60]. Due to the cause, this method is only used to eliminate noise from time- series. Kupferman et al. [45] employed auto-regression of order 1 to predict request rate and commenced that its performance depends typically on several user-defined parameters like size of history window and adaptation window. The combination of Moving Average (MA) and Auto-Regression (AR) technique gives a subsequent algorithm known as the Auto-Regressive Moving Average method (ARMA). Roy et al. [10] used a second-order ARMA for predicting workload which was based on past three observations and this predicted value is further used to evaluate response time. An optimization controller considered this response time for its input and computed a better resource allocation based

on various parameters like the cost of SLO violations, leasing resources cost and reconfiguration cost. The summarized layout of the literature related to time-series forecasting techniques is depicted in Table 2.7.

Table 2.7: Existing Time-series analysis techniques

Ref.	**Scaling Type**	**Technique**	**Proposed Methodology**	**Parameters/ Metrics**	**Platform**
[73]	Vertical	Proactive	PRedictive Elastic reSource Scaling (PRESS)	CPU load, Xen,RUBiS, Response time	Google Cluster
[60]	Vertical	Both proactive & reactive	CloudScale architecture,Prediction error correction	CPU load,Memory usage, Response time	Xen,RUBiS,IBM system
[10]	Not defined	Proactive	Genetic algorithm, Brown's quadratic exponential smoothing	No of requests per VM,VM load	TPC-W,Httperf tool
[76]	Horizontal	Proactive	Look-ahead controller algorithm	No of users in the system	Custom testbed
[11]	Horizontal	Proactive	Neural Network, ARMA	Response time,VM cost,CPU load	Amazon EC2
[77]	Horizontal	Both reactive & proactive	Prototype system for resource provisioning	CPU load,No of VMs, Response time	Eucalyptus, RUBiS, Httperf
[66]	Both horizontal & vertical	Not defined	Adaptive algorithm for parameter selection	CPU load,Queue latency	PlanetLab traces by CoMon,Matlab
[33]	Both horizontal & vertical	Proactive	RPPS architecture, ARIMA prediction model	CPU load, Prediction accuracy	Xen platform,KVM

Some authors have also considered the history window values as input for a neural network or a multiple linear regression method. The precision of these approaches reckons on size of input window and even achieved superior results with more than one past value for prediction. Islam et al. [34] found that the prediction interval parameter should also be considered as an important aspect. Since the setup time of VM Cloud instances is normally around 5 to 15 min, the researchers used a 12-min interval.

Few authors have also combined reactive methods with time-series forecasting techniques. Also, many authors have also concentrated on detecting recurring patterns in the given input workload. Gong et al. [73] gave a comprehensive comparison of the techniques with auto-correlation, auto-regression and histogram. They used Fast Fourier Transform

(FFT) for identifying repetitive patterns in resource usage e.g. CPU, memory, I/O and network. Few researchers also reflected the histogram technique to forecast about resource utilization of applications by taking distribution mean or highest frequency mean into concern.

From the literature, it has been studied that time series forecasting includes a huge variety of methods to recognize patterns and forecast future values. The accuracy in forecast value or average CPU utilization will be resolute by selecting an appropriate technique along with precise parameters, specifically history window and prediction interval. Nowadays, various time-series techniques (averaging methods, auto-regression, machine learning, etc.) are being extensively employed for load prediction. Also, these techniques can be applied to achieve better auto-scaling results during the deployment of IoT-based Cloud applications.

2.3 State-of-the-art: Prediction and migration policies

The various prediction techniques provide an estimation of load, performance and cost of an auto-scaling system. Due to the rise of private and public cloud data centers, it is rather common nowadays to lease virtual machines to host applications. Generally, Cloud users do not prefer to recompense for the resources they acquire but do not use when the load is light. Therefore, it is necessary to acquire the resources earlier than the time when the load actually increases. This consequence would be probable only if the future load can be predicted [78].

According to the predictive value, log information can be formulated for reclaiming future idle resources, transforming to energy-saving mode in advance or adding resources for the upcoming peak load in advance to certify a stable QoS [75]. Moreover, they may have to bear the possibility of performance humiliation when the load is heavy. Also, cloud providers, such as Amazon EC2, offer resources on a VM basis. VMs are added, free, or migrated according to the deviation of load. Every process includes substantial overhead without any authentic benefit [79]. Therefore, it is highly recommended for cloud providers to deliver dynamically finer-grained autonomic scalable facilities that acquire resources according to applications request and grant charges based on the size of their VMs. Moreover, cloud providers can optimize resource provision, increase resource utilization and attain the maximum advantage [80]. In this section, a survey of existing load prediction techniques and existing VM migration policies has been done which can be helpful in the deployment of an effective auto-scaling system.

2.3.1 Load prediction techniques

In past years, the researchers are increasing interest in the information of load prediction centered on exploring and monitoring of data. Few researchers have classified categories of load prediction into short-term, medium-term and long-term load [33]. The short-term load is usually for 1 h to 1 week and help regulate and schedule power systems in routine operations. Medium-term and long-term are for one week to more than one year time period. These methods are also profitable to regulate the capacity of generation, transmission, dispersal and expansion of power system infrastructure, etc. [34].

Numerous authors have surveyed and implemented load prediction techniques so far. Some of them have proposed and implemented load prediction techniques using a combination of linear regression and SVM [33], [81], [82]. Few researchers tried to improve important QoS metrics such as cost, accuracy, CPU utilization, execution time, by using auto-regression methods (AR, ARMA & ARIMA) [83]. On the other hand, neural network methods are also applied for predicting load in past research works [34], [77]. Huang et al. [84] developed a novel approach for load prediction by employing double exponential smoothing. Li et al. [42] proposed an algorithm for autonomic resource scaling and load forecasting using a combination of linear regression with Knuth-Morris-Pratt method. Neto et al. [85] developed an algorithm named as Multi-tenant Load Distribution Algorithm for Fog Environments (MtLDF) to improve the load balancing in Fogs environments. Also, Rahmanian et al. [86] applied learning automata techniques for ensemble resource usage in the Cloud environment. Table 2.8 depicts the summary load prediction techniques discussed so far.

Table 2.8: Existing Load Prediction Techniques

Ref.	**Technique**	**Parameters**	**Application type**
[33]	Linear Regression, SVM	Mean arrival rate, No. of predictions, cost, execution time	CloudWatch
[84]	Double exponential smoothing	Prediction Accuracy, resource idle rate	Cloud application
[83]	NF, AR, MA, SES, ARMA, ARIMA, GA	MAE, MEI, RMSE, MAPE	Web applications
[34]	Neural Network and Linear regression	CPU load, prediction accuracy	Amazon EC2
[87]	ARIMA	No. of requests, CPU load, prediction accuracy, total no of nodes	Web application
[77]	ANN & Black hole	RMSE	Web Applications

Table 2.8: Existing Load Prediction Techniques (cont.)

Ref.	Technique	Parameters	Application type
[12]	ANN, SVM, Random Forest, KNN	RMSE, MAPE	Cloud Application
[81]	LR, SVM	MRPE(Mean Relative Percentage Error)	Real-world Cloud traces
[85]	Multi talent load distribution algorithm	Load, distribution value	Fog computing traces
[88]	Deep Learning methods	Approx. error, classification accuracy drop, speedup	PlanetLab
[86]	Learning automata methods	CPU Load, time, resource usage	CoMon Project

As depicted in Table 2.8, several authors have commenced momentous effort in the field of load prediction in the Cloud auto-scaling environment. These methods are beneficial for predicting load on VMs in Cloud and in improving crucial quality attributes [9]. To conclude, load prediction is an important aspect in auto-scaling which can further support in decision-making of VM migration on the host machine and in refining QoS parameters associated with auto-scaling in Cloud environments.

2.3.2 Virtual Machine (VM) migration techniques

Virtualization is one of the most prominent features of Cloud Computing to covenant with the number of resources and users available. It generates a virtual environment on a single PM (physical machine) by abstracting its hardware details. This helps a tenant to handle multiple processes concurrently and also allows to consumption several instances of an operating system, commonly named as guest operating system [14]. Auto-scaling can be easily deployed on any IoT-based Cloud application by executing this virtual environment. Virtualization also permits VM (Virtual Machine) migration from one physical machine to another.VM migration is carried out in directive to reduce the number of running PMs by migrating few VMs and consolidate them into a reduced number of PMs [88].

The most challenging task in the VM migration process is to select the suitable host because the wrong selection of host may lead to repetition of the migration process, resource wastage, the increased overhead of machines and more time, power and energy consumption [89]. To avoid these circumstances, various VM migration techniques have been proposed so far. As per the literature reviewed, the migration process is generally performed in live, non-live and predictive modes, as shown in Figure 2.9.

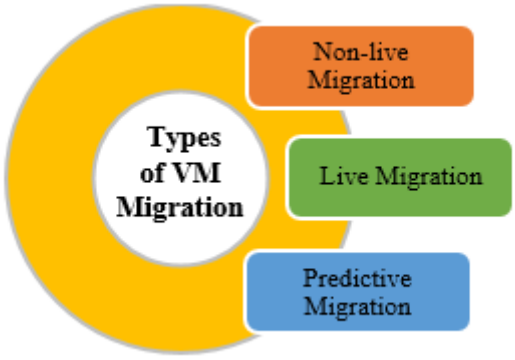


Figure 2.9: Categories of VM migration policies

(*i*) Live VM migration

It allows interruption-free features to users during the relocation of VM pages. Live VM migration perk ups the application performance, increases bandwidth utilization and reduces server downtime [90]. It is further classified into pre-copy, post copy and hybrid techniques.

- Pre-copy: The pre-copy migration method reduces the downtime involved in the migration process and transfers overall VM memory from the source machine before recommencing it at the target machine. It basically works in six steps: (1) Pre-Migration for selection of destination machine, (2) Reservation of resource required at destination machine, (3) Iteratively pre-copying the VM pages in partial rounds, (4) Stop and suspend VM on source node and readdresses the whole traffic to target node, (5) Commitment to set free VM on source machine and (6) Initiation of VM on target machine. The eminent hypervisors such as Xen, KVM and VMware use the pre-copy migration method. However, this method sometimes lead to excess use of network bandwidth, long migration time and repetition of VM pages [91].

- Post-copy: This method interrupts the migrating VM at the source node and then transfers its status information which includes data of CPU registers, I/O states, etc., to the target node. During this process, the state of a VM is shared between source as well as target machine [92]. Firstly, it suspends VM at source node and then captures and transmits its status information to the target machine. After that, VM is resumed on destination machine and the remaining pages are transmitted. The main advantage of this method is that it can easily recover the migrating VM even if the target machine fails during the migration process.

- Hybrid: Hybrid migration is the combination of both pre-copy and post-copy tech-

niques. In this, the first step is to perform a limited pre-copy and after that post-copy approach should be accomplished. This process upsurges the performance as the most used pages are transmitted before the VM is resumed on the destination machine [88]. Hybrid migration also decreases network traffic and reduces the occurrence of page faults at the destination machine.

(*ii*) **Non-live VM migration**

In this mode, execution of a user's application interrupts prior to transfer of VM pages and when VM pages are successfully transferred to the destination PM, the execution is recommenced. This policy is not appropriate for autonomic web and IoT applications as it leads to performance degradation [92].

(*iii*) **Predictive VM migration**

Predictive schemes are introduced in order to reduce the risks that arise during the suitable host selection by predicting the future load in different resources [93]. In other words, various prediction algorithms (e.g. ARIMA, markov model, queuing based, KNN-based, neural network-based, etc.) are used in combination with VM migration policies.

Table 2.9: Existing VM migration techniques in Cloud

Ref.	**Migration Policy**	**Hypervisor**	**QoS parameters**	**Auto-Scaling**
[13]	Hybrid	Xen	Migration time, network traffic	No
[14]	Hybrid	KVM	Migration time	No
[94]	Hybrid	KVM	Migration time, network traffic	No
[90]	Precopy	Xen	Migration time, down-time, n/w traffic, band-width	yes
[95]	Precopy	Xen,KVM, VMWare	Migration time, downtime	No
[92]	Non-live	KVM	Migration time, band-width	No
[91]	Predictive	Xen	Migration time	No
[96]	Predictive	Xen	Migration time, network traffic	No
[97]	Predictive	-	Migration time	yes
[93]	Predictive	CloudSim	downtime, n/w traffic,	No
[98]	Predictive	KVM	Migration time	No
[27]	Predictive	Xen	Migration time, downtime	No

Table 2.9: Existing VM migration techniques in Cloud (cont.)

Ref.	Migration Policy	Hypervisor	QoS parameters	Auto-Scaling
[99]	Predictive	CloudSim	Migration time	No
[100]	Predictive	CloudSim	Bandwidth	No
[101]	Predictive	CloudSim	Migration time, bandwidth	No

As depicted in Table 2.9 above, numerous authors have applied diverse types of VM migration techniques in Cloud Computing and tried to improve the QoS parameters crucial for migrating VMs such as downtime, total migration time, etc. But very few researchers have implemented VM migration techniques for auto-scaling till now. Therefore, there is a need to design and validate an efficient autoscaling system using VM migration for easy deployment of Cloud-based applications.

2.4 Essential QoS (Quality of Service) Metrics for Auto-Scaling

The literature survey shows that auto-scaling is the evolving need for emerging IoT-based Cloud applications. Cloud providers are dedicated through a document that contractually describes the quality of service expected, known as Service Level Agreement (SLA). The convergence of Cloud and IoT requires unified and intelligent interaction among physical as well as virtual entities that also leads to the difficulty to meet a certain level of QoS (Quality of Service). For enhancement of IoT services, it is recommended that each and every IoT-based Cloud application must support the QoS functional aspects [9].

Auto-scaling enables the users to adjust resources as per their needs while complying with SLO (Service Level Objectives) as well as cost reduction [4], [102]. Though necessities of user services diverge significantly with time, there is a need to define a certain QoS. In [103], the authors proposed that parameters essential for auto-scaling in container-based Cloud applications can be divided into two groups- relative and absolute. Relative performance metrics are based on the data collected from any virtual file and absolute measures define the status of cumulative counters in the operating system. The QoS attributes/ metrics such as response time, CPU load, throughput, threshold latency, scalability overhead, etc. are usually part of an SLA and are frequently changing [6]. These parameters must be monitored per user requirements to impose the agreement. The review conducted above shows that various authors have determined different auto-scaling parameters to achieve better scaling results. The important QoS parameters that

various auto-scaling techniques can improve are summarized in Table 2.10.

Table 2.10: QoS parameters required for different auto-scaling techniques

Technique	**NOM**	**RT**	**CL**	**AWT**	**TP**	**SO**	**TL**	**PA**	**C**
Threshhold-based	✓	✓	✓	✓	✗	✓	✓	✗	✓
Reinforcement Learning	✓	✓	✓	✗	✗	✓	✓	✓	✓
Queuing Theory	✓	✓	✓	✗	✓	✗	✗	✗	✓
Control theory	✓	✓	✓	✗	✓	✗	✗	✗	✗
Time-Series	✓	✓	✓	✓	✓	✗	✗	✓	✓

NOM:No. of Migrations; *RT*:Response Time; *CL*:CPU load, *AWT*:Avg. Waiting Time, *TP*:Throughput; *SO*:Scaling Overhead; *TL*:Threshold Latency; *PA*:Prediction Accuracy; *C*: Cost

As depicted in the above table, the number of researchers have used different auto-scaling techniques in Cloud for effective implementation of their systems with distinctive metrics. As per the review discussed above, the parameters including the no. of VMs (Virtual Machine), response time, CPU load, scaling overhead and cost have a great impact on the performance of threshold-based auto-scaling systems. Also, throughput and prediction accuracy parameters are still to be explored while employing threshold-based policies. Similarly, to develop an auto-scaling system based on time-series analysis requires a number of parameters to be taken into consideration such as response time, CPU load, throughput, cost, prediction accuracy, etc. Although, scaling overhead and threshold latency are also important parameters for time-series techniques, still none of the authors have considered these parameters yet. Therefore, for developing auto-scaling systems using time series forecasting, parameters such as response time, CPU load, scaling overhead, threshold latency, prediction accuracy, need to be considered and improved in the future.

The foremost objective of designing an effective auto scaling system is to find a trade-off between these performances metrics to reduce cost of leased Cloud resources (e.g. maximum response time should be not more than 2 seconds) [29]. The number of VMs provisioned or de-provisioned throughout the runtime has straight effect on CPU load as it depends on the no. of VMs waiting or running on physical host. Also, it will affect the response time, average waiting time, scaling overhead, and throughput of the system. To maintain balance between these metrics, many researchers have proposed and implemented load balancer or resource monitor that will keep the status of all the resources of a system [103]. Similarly, increase in CPU load will increase the waiting time and scaling overhead that results in low throughput and performance degradation.

Another common problem occurs in auto-scaling due to the inappropriate setting of threshold value. The fluctuations in the number of VMs is one of the reason that makes it difficult to set threshold value [104]. Mostly the thresholds are calculated at instance level, for example, if average CPU usage is 75% and instance count is 3, it means scale-out action has to be performed when the average CPU across all instances will be greater than 75%. Therefore it is always recommended to set threshold value prior to scale-up or scale-down. The combination of time, memory, bandwidth or other resources that are required to perform auto-scaling also gives rise to overhead sometimes. Similarly, the prediction accuracy/sensitivity of an auto-scaling system depends on different performance inputs used for scaling actions [61].

Due to the changing and uncertain nature of IoT application's architecture and infrastructure, the measurements of QOS demands extra consideration and examination. It generally becomes difficult to define new metric to address the simulations of each gadget or machine used in IoT. To handle this situation, it is recommended to define a degree of acceptance for performance because some metrics cannot remain constant at certain values in certain situations [105]. For instance, a temperature checking IoT device demands availability as a performance metric but not necessarily all the time. Therefore a range of response time can work well for such applications. Similarly, if we consider driverless cars, the car detection system requires low latency for better performance [24]. In case of smart cities, face detection systems are considered as latency-sensitive applications. As a solution, dynamic thresholds for response time may fulfil the latency requirements and help to upgrade the system performance as well [7]. Therefore, for an effective auto-scaling system in IoT-based Cloud applications, these performance metrics must be validated and achieved in order to meet customer's requirements. Although, it is not promising to validate all these parameters at once or in a single auto-scaling framework, as minimizing one or two metrics can maximize the other metric values or vice-versa.

2.5 Gap Analysis

Based on the various issues discussed in the literature survey, some of the research gaps that have been identified are:

- Cloud Computing a prominent technology as it provides various benefits to end user such as leasing resources as per their need and only pay for use. Many Cloud providers like Amazon EC2, Google App Engine, Microsoft Azure, etc. provide computational and storage capacities for various IoT-based applications. Cloud

Computing in integration with IoT raises the improvement of a number of interesting applications like smart cities, healthcare, automotive and smart mobility, video surveillance and so on. Most of these applications are deployed by installing various sensors which produce petabytes of data that needs to effectively stored, processed and analyzed. Therefore, a lot of work is required to explore these data storage issues in order to fulfil the aim of implementing efficient IoT-based Cloud applications [29], [9].

- Various IoT-based Cloud applications are achieving great commercial success in recent years, still many issues are need to be addresses in context of these applications, such as heterogeneity of complicated devices and technologies, performance, reliability, security & privacy preservation, scalability, etc. Scalability is one of the imminent issues for IoT-based applications. With the optimum scalability, service providers don't need to be worried for over-provisioning or under-provisioning of resources, thus eliminate wastage of expensive resources and missing potential customers as well the revenue. Providing flexible services and management of resources while promising scalability with respect to IoT-based Cloud applications and their respective users is still reflected as open issue [8].

- While planning to implement any service/application with auto-scaling features, performance and cost would be considered as the major factors. The cost will decrease if we rent few virtual resources from a Cloud provider, but performance will be impacted with the increase in peak load. On the other hand, when resources are used properly, it may lead to enhanced performance, but also increases the cost. With such a problem there is a need to predict platform workload in which virtual resources can be appended and expelled accordingly [39], [7].

- Various researchers attempted to develop optimal policies for automatic scalability which has many components such as virtual machine placement, task scheduling, load forecasting and so on. Time-series forecasting techniques such as averaging methods, auto-regression, machine learning, etc. plays a major role in these efforts. In fact, the use of suitable time-series forecasting technique for load prediction in auto-scaling systems can give optimal results in improving prediction accuracy [77].

- In some conditions, the reactive auto-scaling systems might not be appropriate to handle unexpected changes in input workload. The main reason is time needed to attain and assemble new resources. In such cases, focus should be relayed on proactive auto-scaling systems that are competent to predict future necessities and to identify possible prediction errors. The auto-scaling systems can yield benefit from prediction abilities of time series forecasting techniques along with automated

controllers [4].

- Although, numerous auto-scaling approaches have been proposed, but still there is a need of an effective auto-scaling strategy for IoT-based Cloud applications. The reason behind is the advancement of technology day by day and to handle the increasing demands of end users [26].
- According to the need of resources, setting threshold value is necessary to control the increase or decrease in the number of running instances committed. During allotment process more instances may require to be append to manage spikes of the incoming CPU load. Some of the metrics such as CPU load, threshold latency, scalability overhead, etc. have definitely a huge influence on performance and cost of the system. Therefore, optimal setting of these parameters is required to implement any auto-scaling approach [106].

2.6 Problem Statement

Cloud computing empowers IoT applications to apply on-demand and scalable resources dynamically. However, elasticity, reliability, and high performance are always difficult to achieve for cloud applications. The process of dynamically assigning resources to coordinate execution requirements is known as auto-scaling. An application may need more resources when the workload increases to keep performance within acceptable SLA. For instance, it could be difficult to recognize a high spike resulting from increasing users on Amazon or Flipkart. If the auto-scaling metrics and policies of the system are monitored efficiently and timely, then the system could respond much more quickly. So, by considering the past and predicting the future load, the auto-scaling method offers the best performance against the drawbacks of the reactive approach. Moreover, auto-scaling allows the scaling of Cloud services, such as virtual machines and server capacities. The load on the physical machine is scaled up or down automatically based on factors such as CPU, memory, and disk utilization. Auto-scaling tools are available from cloud computing providers like AWS, Microsoft Azure, and GCP. These service providers allow users to automatically launch or terminate virtual instances based on predefined policies, schedules, and regular load status checks. While certain researchers have tried to develop the best policies for auto-scaling, scalability in IoT-based cloud applications can still be improved using an efficient auto-scaling approach.

Therefore, there is a need to develop a proficient auto-scaling approach that should be capable of automatically scaling up and down according to user requirements. Also, to predict workload, a load prediction model is required that can improve prediction

accuracy using time-series forecasting techniques. Based on the predicted load, the scaling decision can be made, i.e., whether to scale up or down VMs using a VM migration approach. An auto-scaling approach could be deployed to subsequently validate various QoS parameters such as CPU utilization, response time, error rate, scalability overhead, etc. Moreover, the ideal setting of these parameters could lead to improved performance and cost reduction for IoT-based Cloud applications.

The broad objectives of this research work are:

i. To explore and study existing auto-scaling techniques for IoT-based Cloud applications.

ii. To propose a load prediction model using time-series forecasting techniques to improve the prediction accuracy.

iii. To design and implement an Auto-Scaling approach using the proposed load prediction model to enhance the scalability.

iv. To validate the proposed auto-scaling approach in the Cloud environment for offering requisite performance to the end user.

2.7 Conclusion

This chapter highlighted the basics concepts as well as functional aspects of auto-scaling and also covered the limitations and challenges faced by various applications. For autonomic scaling, it is recommended to predict and maintain the future load (e.g. no. of VMs consumed on physical host). Hence, the review of load prediction and VM migration techniques for auto-scaling has been done. Further, various essential parameters have been concluded that needs to be optimally set and validate the effectiveness of an auto-scaling framework.
The next chapter proposes a solution to the research problem by proposing a load prediction approach based on an ensemble technique for predicting resource usage in cloud computing. The proposed technique addresses the gaps related to load prediction identified in the literature survey.

Chapter 3

ETSA-LP: An Ensemble Time-Series Approach for Load Prediction in Cloud

Cloud computing has become a prime infrastructure for tenants to deploy their IoT applications as it offers a parallel and distributed environment for large-scale computations. During deployment, load prediction is essential to achieve optimal and intelligent scaling of IoT-based Cloud applications.

The existing load prediction models fail to provide reasonable accuracy owing to the high variances in cloud metrics. Therefore, to handle the changing cloud resource demands, it is necessary to accurately predict the future requirements for automatically provisioning the resources. Thus, this chapter emphasizes deploying a dynamic and autonomic load prediction framework.

This paper proposes an Ensemble Time-Series Approach for Load Prediction (ETSA-LP), which integrates various time-series analysis techniques for predicting CPU and memory utilization. To evaluate the efficiency of the proposed approach, a series of experiments on Google and PlanetLab traces have been conducted in a real Cloud environment. The results have been compared according to different performance metrics and models having determined accuracy and the minimal error rate has been selected as the best among others. The proposed ensemble approach gives the best performance over the existing models by showing remarkable accuracy improvement and reducing the error rate and execution time.

3.1 Objectives of the Proposed Technique

For vast cloud data centers, physical and virtual interrelated networks are required to communicate at a faster Giga-bit speed. However, this initial requirement gives rise to the problem of predicting the resource availability in the system for upcoming processes in advance. As the present computer systems and their workloads are dynamic, it becomes challenging to make such autonomous predictions [8]. Cloud computing platforms must be able to propose and acquire resources quickly to ensure high scalability, flexibility, and effective cost. This will also help sustain the prerequisites of various Cloud-based applications by dynamic load prediction for appropriate resource utilization.

Based on a Cloud system's estimated future load and performance, users can make desired decisions to prevent the system from traffic flow caused by peak load [81]. There should be an accurate simulation of the correlation between historical and future values for precise and intelligent load prediction in Cloud Computing systems. Proper knowledge of backend workloads is also necessary [107]. Much literature has been published on load prediction as discussed in Section 2.3.1 of Chapter 2, but numerous challenges still exist. The reasons why autonomic load prediction in Cloud is recommended are:

a) The first reason is that most recent applications in the Cloud (e.g., e-health, smart homes, smart cities, etc.) have varying loads because of their resource consumption, power usage and configuration details that change from time to time. It may result in complex and unpredictable behavior of resources and has a major effect on the load of a Cloud System [77].

b) Due to the ever-increasing demand for resource usage and intrusion between the applications accommodated on physical machines, it becomes crucial to design and implement autonomic resource usage prediction in terms of load [12].

c) When allocating several requests to a single VM, it may result in the over-provisioning of resources. Due to the reason that number of requested resources becomes more than the number of available resources, the performance features of an application assigned to the VM will degrade. Hence, a prediction model must be able to notify about the under-provisioning or over-provisioning situation of resources prior to the resource manager [108].

d) As there are high inconsistencies in cloud performance metrics, the existing load prediction models usually fail to offer appropriate accuracy. Therefore, an intelligent ensembling technique combining prediction results from different models can proficiently schedule cloud resources in this scenario [109].

To address the above-mentioned challenges, an ensemble approach is proposed to predict the load in the cloud environment. The primary objectives of the proposed prediction technique are summarized below:

i. The problem of load prediction using CPU and memory utilization is analyzed so that Cloud systems can intelligently handle under-provisioning and over-provisioning situations of resources.

ii. Modeling an Ensemble Time-Series Approach for Load Prediction (ETSA-LP) framework for load prediction using five time-series analysis techniques (ARIMA, ANN, SVM, LSTM & ES).

iii. Using the proposed framework ETSA-LP, CPU and memory usage estimation has been precisely done.

iv. Execution of experiments has been performed to assess the performance of the recommended ETSA-LP algorithm with the traces of Google and PlanetLab datasets regarding the accuracy and error benchmarks.

3.2 Load prediction: Preliminaries

With the advancement of private and public cloud data centers, leasing virtual machines to host applications is quite common. Usually, Cloud tenants do not choose to pay for the resources they attain but are not consumed when the load is light. This concern would be feasible only if the upcoming load can be predicted earlier before validating Quality of Service (QoS) parameters [77]. Furthermore, there can be a probability of performance degradation in case of heavy load [110].

Generally, Amazon EC2 service providers recommend resources on a VM basis and allow VMs to be added, released, or migrated according to the load divergence. So, it is enormously endorsed for cloud service providers to offer finer-grained autonomic facilities that dynamically attain resources according to an application request and permit cost based on the VM size [79]. Also, there should be an accurate simulation of the correlation between historical and future values for precise and intelligent load prediction in Cloud Computing systems. Moreover, proper knowledge of backend workloads is necessary. Therefore, predicting over-utilized and under-utilized hosts requires more sophisticated algorithms such as time-series analysis techniques to predict the load proactively.

3.2.1 Load Prediction: Time-series Analysis techniques

A time-series method organizes data points to be measured at constant time intervals. This technique can be applied to regulate self-repetitive input workload patterns or foresee forthcoming values [25]. For instance, every performance parameter such as average CPU load will be calculated at fixed intervals. therefore, time-series analysis is the most prominent resource usage and workload prediction approach in Cloud Computing.

Efficient forecasting works with clean, time-stamped data and can identify genuine trends and patterns in historical data [111]. But, this contrasts while working on static data. Also, the choice of algorithms in time-series techniques differs entirely from those in static. Most static machine learning algorithms like linear regression do not have this capability as they generalize the training space for any new prediction. In this context, machine learning-based models (such as SVM, LSTM, ANN, ES) are attaining inclusive attention in forecasting load nowadays because of their introspection aspect for multiple applications [47]. So, the primary focus of this paper is to evaluate different statistical, neural, and ensemble techniques in their ability to predict resource load. This paper uses the following five models as base predictors:

a) Auto-Regressive Integrated Moving Averages (ARIMA): This model defines the prospect value of a variable that is supposed to be a linear function of the last few observations and some random errors. This statistical model is popular and could be a good baseline for comparing other models because of its simplicity [17]. However, one challenge in the ARIMA model is its pre-assumption of linearity in the underlying time series, which may need to be improved in various practical scenarios that contain non-linear time-series data.

b) Artificial Neural Networks (ANN): ANN technique can solve many problems in classification and long-term forecasting compared to linear statistical models. This model belongs to the data-driven approach, where training depends on the available data with little prior rationalization regarding relationships between variables. ANNs are self-adaptive as they do not make assumptions about the underlying time series' statistical distributions and can naturally perform non-linear modeling [77].

c) Support Vector Machines (SVM): This model can categorize linear data accurately and is also used for complex decision boundaries because of its lower complexity [82]. This classic method is generally used in classification, regression, etc. to solve complex real-world problems with data having multiple input features.

d) Long Short-Term Memory (LSTM): A type of neural network with the capability to memorize the past values in the network is known as the LSTM model. LSTM

is a supervised deep learning method widely used in time series nowadays [112].

e) Exponential Smoothing (ES): Exponential smoothing is an alternative linear model dependent on fundamental historical observation values to make predictions. It weighs so that recent observations are much heavier than old observations.

3.2.2 Load Prediction: Ensembling Approach

Though much research has been done on load prediction and time-series models, there is a need for an inclusive methodology to combine different types of prediction models. More profoundly, one prediction model would effectively predict some trends, but the same would be imprecise in other trends due to its dynamic performance [108]. In conventional ensembling techniques, the most often occurred, or the average prediction from multiple models is chosen as the concluding prediction outcome from the ensemble [113].

This research proposes a dynamic Ensemble Time-Series Approach for Load Prediction (ETSA-LP) that combines the best prediction results from compound models (ARIMA, ANN, SVM, LSTM and ES) to predict CPU and memory utilization. A dynamic exponential weighting algorithm has been proposed, in which a distinct model's best prediction outcome selected from diverse models has been dynamically weighted. A brief discussion of this proposed algorithm is given in the upcoming sections.

3.2.3 Load Prediction: Evaluation Metrics

The QoS attributes/ metrics, such as response time, CPU load, throughput, accuracy, cost, etc., are usually part of an SLA and are frequently changing. These parameters must be monitored according to user requirements to impose the agreement. Numerous researchers have used different load prediction techniques in Cloud to effectively implement their systems with specific metrics [9]. As there may be many user requests in multiple service queues, existing load prediction systems need more evaluation metrics for dynamic user interactions with the system. Therefore, QoS parameters like CPU usage, memory usage, error rate, response time, and so forth can be added to test the system's performance. In this research work, the base predictors ARIMA, ANN, SVM, ES and LSTM have been compared with the proposed ETSA-LP to assess the best performance using the following metrics:

a) RMSE: Root Mean Square Error
For experimental evaluation, RMSE is calculated to measure the error in percentage. RMSE is defined in Equation 3.1 where X_t is the actual output,x_t is the predicted output and n specifies the total number of observations in the dataset. The lower

value of RMSE validates to be a more precise prediction technique.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(x_t - X_t)^2} \tag{3.1}$$

b) R (Coefficient of Correlation) and r^2 (Coefficient of Determination)
R is a measure of the goodness-of-fit, which its value falls within the range (0, 1) and is generally applied to the linear regression models [8]. r^2 is a statistical measure to denote how close the data is fitted on the regression line. It is also defined as the coefficient of determination or the coefficient of multiple determination for multiple regression. It is the percentage of the response variable variation that a linear model represents. Specifically, the higher the r^2, the better the model fits the data. The value of r^2 is always between 0 and 100%:

- 0% means that the model describes none of the variability of the response data around its mean.
- 100% means that the model describes all the variability of the response data around its mean.

c) Accuracy
Accuracy is the degree to which the outcomes of an actual calculation or measurement are grouped around the accurate value. The mathematical notation to compute accuracy is given below in Equation 3.2:

$$Accuracy = \frac{tp + tn}{tp + fp + tn + fn} * 100 \tag{3.2}$$

Here, tp, tn, fp and fn denote the number of true positives, true negatives, false positives and false negatives, respectively.

d) Total Execution Time
Total execution time is the overall time a process occupies to finish execution. The formula to calculate total execution time is indicated in Equation 3.3 given below:

$$TotalTime = t_e - t_s \tag{3.3}$$

Here,t_e denotes the finish time for executing data and t_s is the initial time when execution starts actually.

3.3 Proposed Ensemble Approach: ETSA-LP

The main components of the proposed ETSA-LP framework are depicted in Figure 3.1. It consists of a load balancer, Load Predictor module, resource manager module, data storage module, and Cloud infrastructure in the system. Cloud tenants use different resources or virtual machines to deploy various applications and services on Cloud and these resource utilizations are monitored and profiled by the resource manager. It will also gather efficient metrics to check resources' past and present state. The required metrics (CPU & memory usage) and logs are transferred to the workload database and the historical information. Also, the historical workload information is used to train the prediction model.

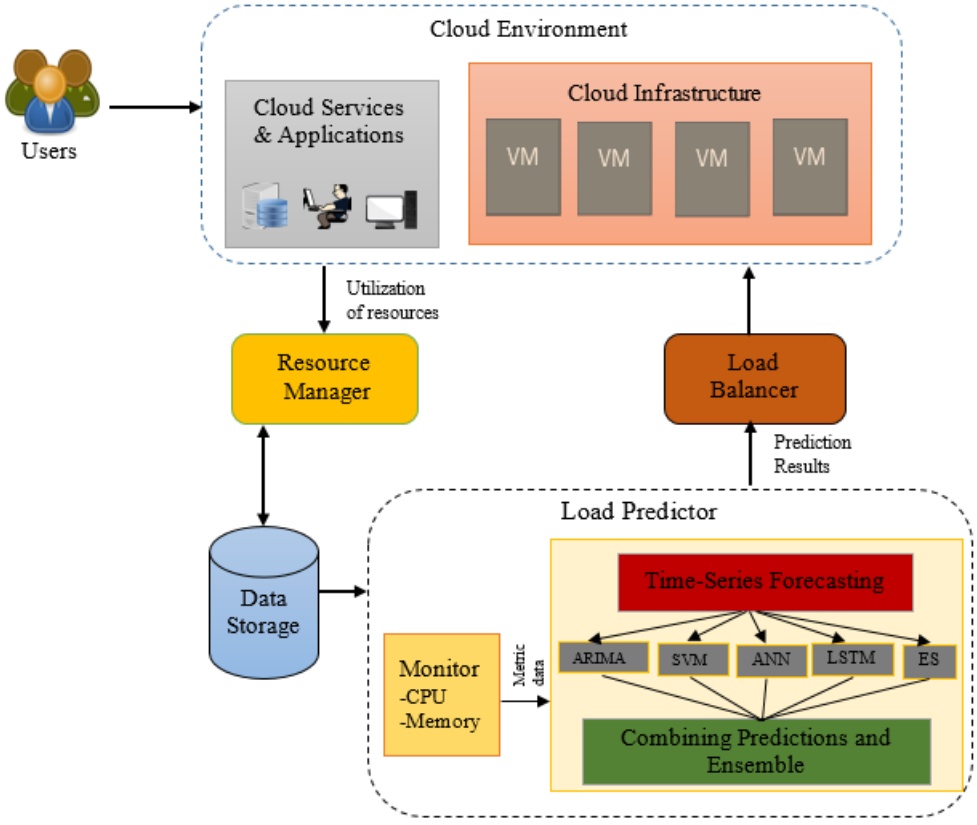


Figure 3.1: Proposed Ensemble Load Prediction Framework: ETSA-LP

Load predictor allocates suitable time-series forecasting techniques for the prediction of workload. Moreover, it is responsible for combining the forecasts of base models and applying an ensemble algorithm to find optimal results. The resource manager also dynamically manages the resource allocation according to user request changes. It estimates the number of resources according to the status of the resource monitor. It also dynam-

ically manages the resource allocation according to user request changes and publishes load-balancing instructions to the load predictor. This load-balancing module controls load among various VMs deployed on the Cloud environment.

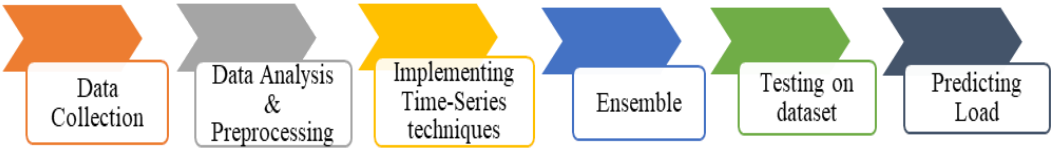


Figure 3.2: Methodology for proposed ETSA-LP approach

The methodology used for the proposed load prediction framework is depicted in Figure 3.2. The first step is to gather logs of different resources Cloud users use as a dataset. After the data pre-processing, the next step is to apply time-series forecasting techniques on the given dataset to compare the accuracy of different models. The time-series forecasting techniques used in this proposed work are ARIMA, NN, SVM, LSTM and ES. The predictions from these base models have been ensembled for efficient and dynamic load prediction using the proposed ETSA-LP. Further, the effectiveness of the proposed approach is validated by testing on Google Cloud, and the results are compared with those of existing models. Table 3.1 presents the list of notations used in this approach.

3.3.1 ETSA-LP: Exponential Weighted Algorithm

The working of the exponentially weighted algorithm used for the ensemble in the proposed ETSA-LP is presented in Algorithm 1. The algorithm starts with a given set of N predictions from diverse models on the training data and assigns equal weight to them (line 1). In the second step, the weight of each model is assumed as $1/n$, which verifies that the sum of all model weights is equal to 1 at each step (line 2). Then, for each training sample (e.g. 75 training samples in the case of Google traces), the squared error between each model's prediction and actual data is calculated i.e. RMSE is calculated for different model predictions (line 3). After each training sample, each model's weights are revised using the squared error for other model predictions, where x is assumed as a variable-rate parameter (line 4). Finally, the weights of each model's predictions are normalized by dividing them by the total sum of the weights across all models (line 5). The normalization of weights is performed to bring all weights between 0 and 1. This process repeats until all training samples are covered. The different values for x are defined to obtain the optimal weight corresponding to each model, for instance, 0.0 to 1.0. These

Table 3.1: Notations used in the chapter

Component	Description
n	Set of predictions
1_n	weight assigned to each model
T	Training samples
M	Model used
W	weight of each model (updated)
S	Sum of models weights
x	variable-rate parameter
P	Prediction outcomes of each model
R	RMSE values
N	No. of nodes
J_N	No. of jobs
V_N	No. of VMs
C_i	Clock cycles per instruction
IC	Instruction count
CR	Clock rate
V_n	No. of VMs running on Nth node
$totalCU_N$	Total CPU utilization
$avgCU_N$	Average CPU utilization
$totalMU_N$	Total memory utilization
$avgMU_N$	Average Memory utilization

weights are then selected to obtain the minimum value for RMSE and maximum value for R^2.

As discussed above, the different values for the variable-rate parameter x are defined to obtain the optimal weight corresponding to each model. Therefore, 11 different values of x parameter are tried in this ensemble algorithm. For instance, with $x = 0.3$, the ensemble results were obtained with corresponding weights: ARIMA (0.313), SVM (0.332), ANN (0.329), LSTM (0.300) and ES (0.335). Similarly, when the value of x is taken as 0.8, the ensemble results were obtained with these weights: ARIMA (0.011), SVM (0.231), ANN (0.534), LSTM (0.768) and ES (0.422). The overall results and comparison of the proposed ETSA-LP approach based on this ensemble algorithm are discussed in the upcoming sections.

3.3.2 Dynamic Resource Prediction Algorithm using CPU and Memory utilization

The exponential weighted algorithm for ensemble gives the ideal combination of machine learning models and improves the prediction accuracy. This ensemble model is

Algorithm 3.1 ETSA-LP: Exponential Weighted Algorithm

Require: $M_i, T, P\frac{T}{M_i}$

1. For each model M,
   $W\frac{1}{M_i} \leftarrow \frac{1}{n}, M_i = 1 \ldots nModels$
2. Sum of all models be equal to 1
   $S = \sum_{i=1}^{n}(W\frac{1}{M_i} == 1)$
3. For i=1 to N, Calculate RMSE of models
   $R(P\frac{T}{M_i}Y_t) = (P\frac{T}{M_i} - Y_t)^2$
4. Update weights by adding RMSE
   $W\frac{R}{M_i} + 1 \leftarrow W\frac{R}{M_i} * xR(P\frac{T}{M_i}Y_t)$
5. Normalize the weight of each model
   $W\frac{R}{M_i} + 1 \leftarrow \frac{W\frac{R}{M_i}+1}{\sum_{i=1}^{n} W\frac{1}{M_i}+1}$

employed on the Google Traces and PlanetLab datasets for prediction. The final output of Algorithm 1 is used as an input parameter in Algorithm 2, which is further used for dynamically predicting CPU and memory utilization. The first step is to initialize the value of the number of jobs and the number of VMs as 1. Then, for each Kth job running on the Nth node on Mth VM, CPU and memory utilization are calculated as shown in step 2. Further, the total CPU utilization is predicted by the formula given below:

$$totalCU_N = \frac{C_i * IC}{CR} \tag{3.4}$$

In Equation 3.4, C_i refers to the clock cycles per instruction, IC is the instruction count and CR denotes the clock rate. By taking this total CPU utilization into account, the average CPU utilization is calculated as follows:

$$avgCU_N = \frac{\sum_{M=1}^{V_N}\sum_{K=1}^{J_N} CU_x}{totalCU_N} * 100 \tag{3.5}$$

In the above equation 3.5, $avgCU_N$ is calculated at any given time for Nth node. considering V_N as the number of VMs running on the Nth node and J_N as the number of jobs assigned to V_N VMs. CU_x is the CPU utilization of K jobs running on M VMs on the Nth node. Similarly, the average memory utilization is calculated by the formula:

$$avgMU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} MU_x}{totalMU_N} * 100 \tag{3.6}$$

At any given time, for the Nth node, the average memory utilization $avgMU_N$ can be given as shown in Equation 3.6, where V_N is the number of VMs running on the Nth node and J_N is the number of jobs assigned to V_N VMs. MU_x is the memory utilization of K jobs running on M VMs on the Nth node and $totalMU_N$ is the total memory utilization for the Nth node.

Algorithm 3.2 Dynamic Resource Prediction Algorithm (CPU and Memory)

Require: N, M, J, K, C_i, IC, CR

1. **Intialize** M=1,J=1
2. For each Kth job running on Nth node on Mth VM,
   Calculate CPU & Memory utilization
   $CU_x, MU_x; x = NMK$
3. For each Nth node on Mth VM,
   Calculate total CPU utilization
   $totalCU_N = \frac{C_i * IC}{CR}$
4. Calculate Average CPU utilization
   $avgCU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} CU_x}{totalCU_N} * 100$
5. Calculate Average memory utilization
   $avgMU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} MU_x}{totalMU_N} * 100$
6. **Return** $avgCU_N, avgMU_N$

3.4 Experiments and Evaluations

This section elaborates on the dataset used and compares the proposed approach with the existing techniques for good performance.

3.4.1 Dataset Description

Due to great variabilities in Cloud Computing workloads, it is nearly 20 times noisier than Grid Computing workloads. Therefore, for effective validation of our proposed approach ETSA-LP, two real cloud workload traces from Google and PlanetLab have been used as

baseline benchmarks. Google traces includes the data of 29 days collected from Google's cluster cell. This workload is tested every 5 minutes and contains 75 samples inclusive of 9218 jobs and 176,580 tasks [114]. In the experiments, the resource usage proportion comprises the traces of CPU and memory demands of VM instances hosted on the cluster as shown in Table 3.2. All these measurements are regulated between 0 and 1 by the relative maximum values for which the machine has been furnished.

Table 3.2: Google Cluster: Workload Information (CPU & memory traces)

Attribute Name	**Datatype**	**Attribute Name**	**Datatype**
start time	integer	max memory usage	float
end time	integer	disk IO time	float
job ID	integer	local diskspace usage	float
task index	integer	maximum CPU rate	float
machine ID	integer	maximum diskIO time	float
CPU rate	float	cycles per instruction	float
canonical memory usage	float	memory access per instruction	float
assigned memory usage	float	sample portion	float
unmapped page cache	float	aggregation type	boolean
total page cache	float	sampled CPU usage	float

On the other hand, PlanetLab data traces cover the mean CPU utilization of more than 1000 VMs as shown in Table 3.3. These VMs are sampled over a 5-minute interval on 10 different days from 03/03/2011 to 20/04/2011. During the simulations, each VM is randomly assigned a workload trace from one of the VMs from the corresponding time. The workload makes initiating VMs with a real configuration possible, while CPU utilization of initiated VMs according to the PlanetLab dataset changes over time, similar to real VMs [115].

3.4.2 Evaluation of base models and ETSA-LP

Firstly, the performance of five machine learning base models, statistical ARIMA, ANN, SVM, LSTM and ES is dignified and then with their resulting best predictions, the ensemble model is evaluated. Figure 3.3 depicts the comparative results of the five base models tested on the Google cluster for CPU utilization. Similarly, Figure 3.4 represents the results of base models and the proposed model tested on the Google cluster for

Table 3.3: PlanetLab: Workload Information

Date	**No. of VMs**	**Mean%**	**SD%**
03/03/2011	1052	12.31	17.09
06/03/2011	898	11.44	12.83
09/03/2011	1061	10.70	15.57
22/03/2011	1516	9.26	12.78
25/03/2011	1078	10.56	14.14
03/04/2011	1463	12.39	16.55
09/04/2011	1358	11.12	15.09
11/04/2011	1233	11.56	15.07
12/04/2011	1054	11.54	15.15
20/04/2011	1033	10.43	15.21

memory utilization. On the other hand, the experimentation results conducted using PlanetLab traces for CPU utilization are highlighted in Figure 3.5. Likewise, memory utilization metrics tested on PlanetLab traces are shown in Figure 3.6.

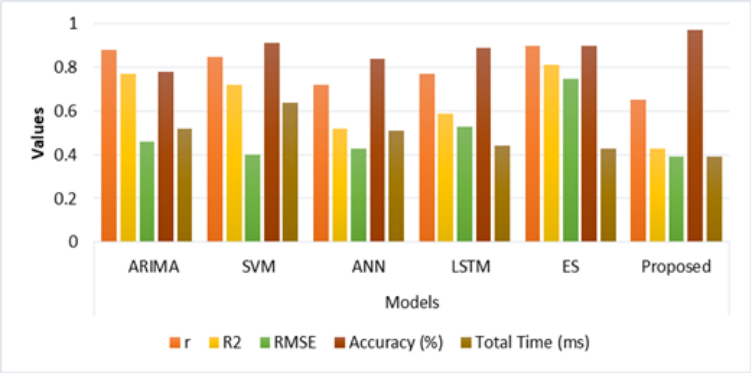


Figure 3.3: Performance Metrics of different models on Google traces (CPU)

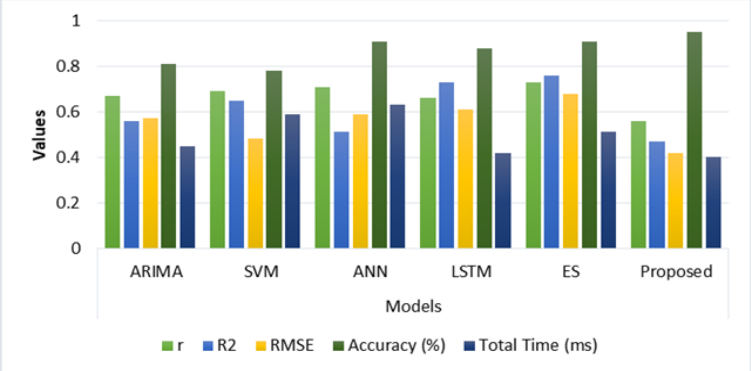


Figure 3.4: Performance Metrics of different models on Google traces (Memory)

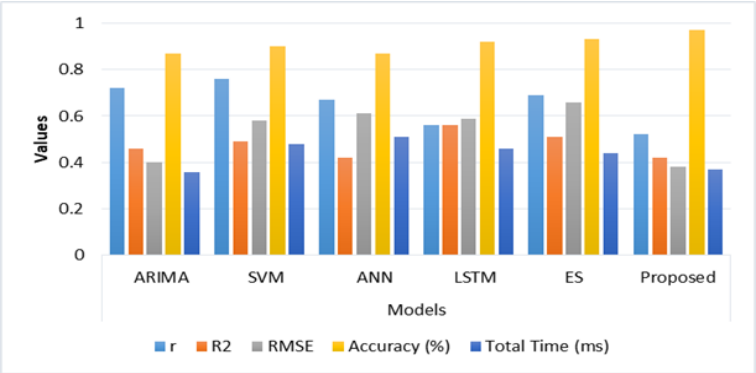


Figure 3.5: Performance Metrics of different models on PlanetLab (CPU)

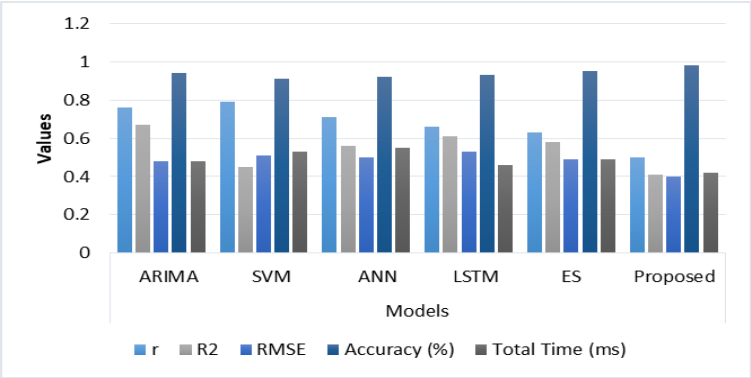
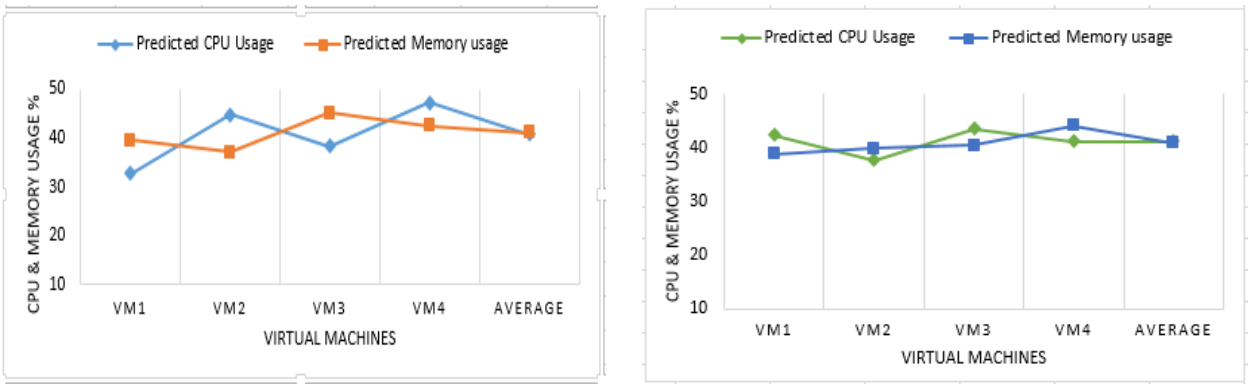


Figure 3.6: Performance Metrics of different models on PlanetLab (Memory)

As per the experiments, it has been observed that an individual model offers a different performance concerning any evaluation metrics due to its dependency on the tested dataset type. If there is a change in the dataset, the values of the performance metrics also vary. For instance, when applied to the Google traces dataset, SVM beats the other four time-series-based models in accuracy (91.34%). But, it performs less with accuracy (78.70%) for the PlanetLab dataset. Therefore, these base time-series models are combined based on the proposed ensemble algorithm discussed in Section 3.3.1 to boost the performance. The proposed ensemble model ETSA-LP gives the best accuracy (97.45%) among all other base models with a total execution time (0.39 ms) for the Google traces dataset.

Similarly, in the PlanetLab dataset, ETSA-LP gives superior accuracy (95.78%) among the existing models with a total execution time (0.40 ms). The overall accuracy is improved by approximately 3% and execution time is reduced by 12.2%. Also, the RMSE value for ES is maximum (2.82%) whereas ETSA-LP has a minimum error value (0.39%). Hence, it is substantiated that the proposed ETSA-LP ensemble approach performs better than the existing individual time-series-based models.



(a) Google traces (b) PlanetLab

Figure 3.7: Predicted CPU & Memory usage

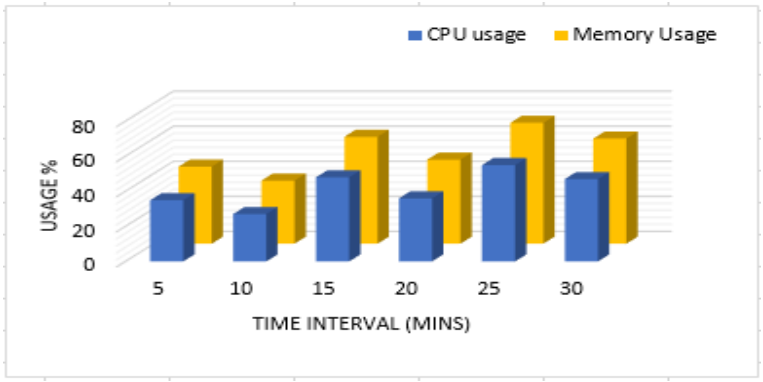


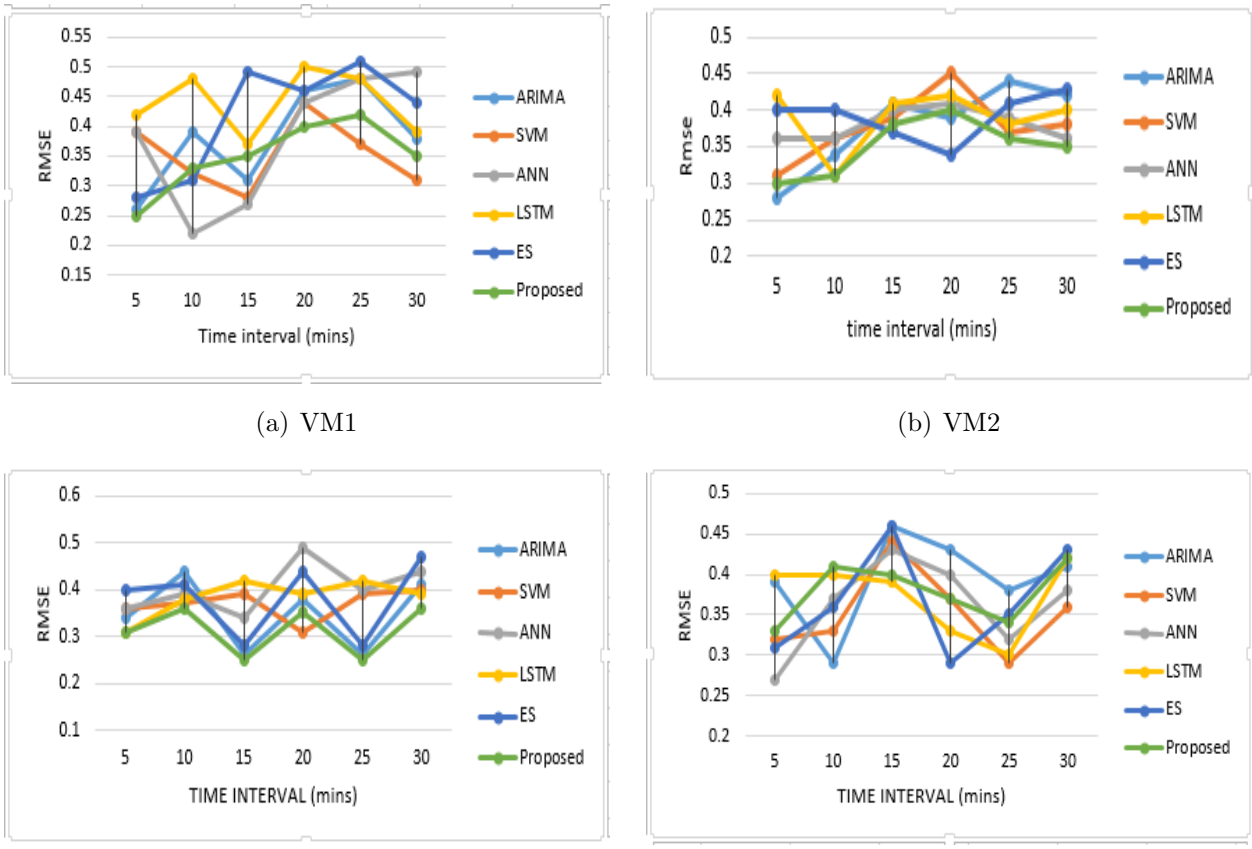
Figure 3.8: Average CPU & Memory utilization of VMs at different time intervals

a) **Average CPU & Memory utilization**

Figure 3.7 illustrates a VM's predicted CPU and memory usage during training models on the Google traces and also corresponds to the utilization deployed on the PlanetLab dataset. In this proposed work, an effort has been made to assess all possible effects of load prediction with low or high CPU and memory usage utilization factors that can directly or indirectly affect the performance of the recommended ensemble approach. For instance, in the case of Google traces, the predicted CPU usage of VM1 (31%) is lower than its memory usage (39%). Similarly, the predicted CPU usage of VM1 (42%) is higher than its memory usage (37%) in the PlanetLab dataset. The results also vary as the time intervals change, as shown in Figure 3.8. With the difference of five-minute intervals, the CPU and memory usage goes high or low according to the number of user requests. The results suggest that with prior knowledge of resources to be utilized for an application, the issue of an under-provisioned host/VM can be resolved and resource-task mapping can be done efficiently.

b) **RMSE**

Figure 3.9 compares the mean error RMSE of different VMs at different time intervals. With five minutes interval, SVM gave the error rate of (3.62%, 4.23%, 4.96%, and 3.74%) while using ARIMA the error rate was (2.9%, 3.71%, 3.32% and 4.50%) for VM1, VM2, VM3 and VM4, respectively. The error rate of the proposed ensemble approach is (2.5%, 3.14%, 2.75% and 3.7%) on VM1, VM2, VM3 and VM4, respectively. The results clearly indicate that the proposed ensemble approach is better than the existing ensemble approach on the basis of error rate.



(a) VM1

(b) VM2

(c) VM3

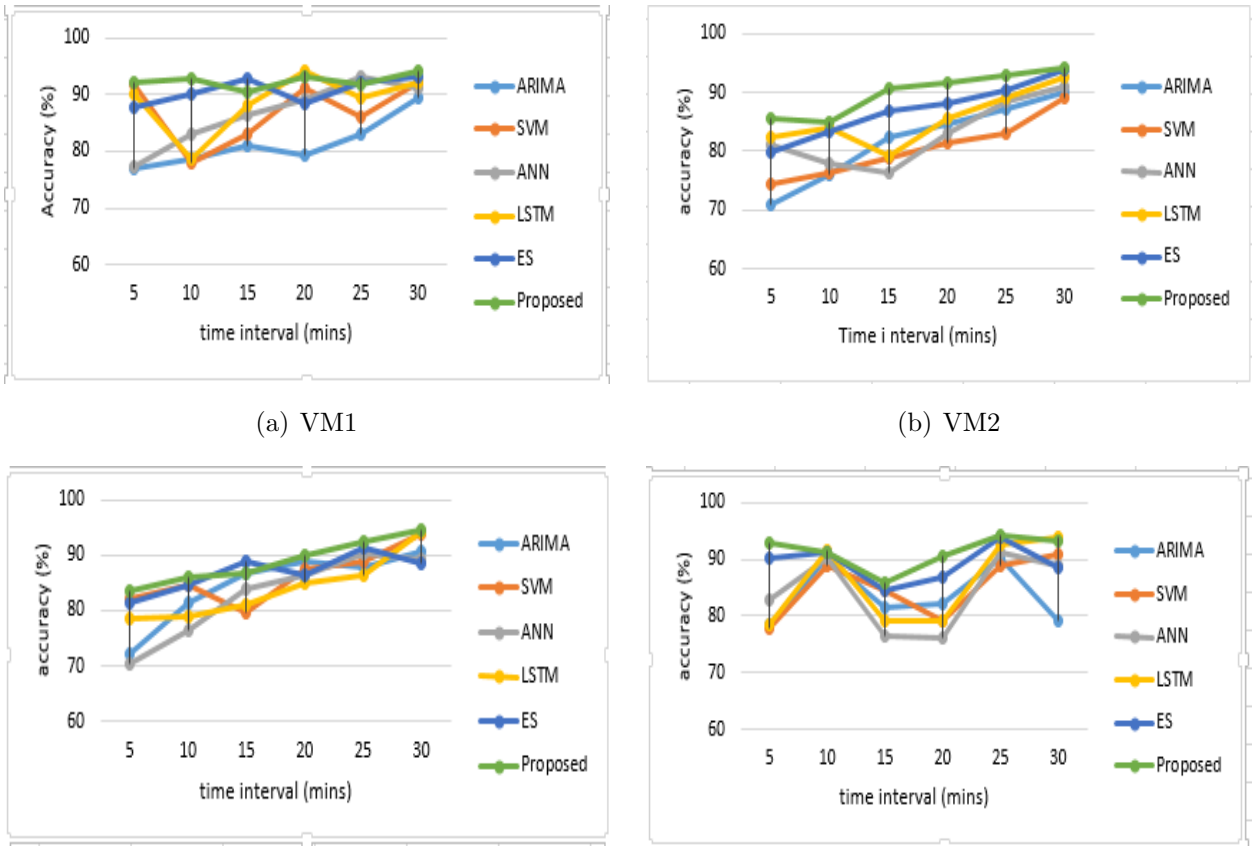
(d) VM4

Figure 3.9: RMSE comparison of different VMs at different time intervals

c) **Accuracy**

The accuracy comparison of other models with the proposed is demonstrated in Figure 3.10. Taking an example, ANN outperforms the other models with 92.62% accuracy for VM2 but it gives a lower accuracy of 71.34% for VM3. In order to enhance the performance, these individual models are combined together on the basis of the proposed ensemble algorithm. The proposed ensemble model gives the best accuracy of 95.45% among all, with 0.39ms total execution time. The overall

accuracy is improved and execution time is also reduced. The results recommend that if the load can be effectively expected with former awareness of resource utilization, the over-utilization or under-utilization of resources can be either ceased or handled commendably.



(a) VM1

(b) VM2

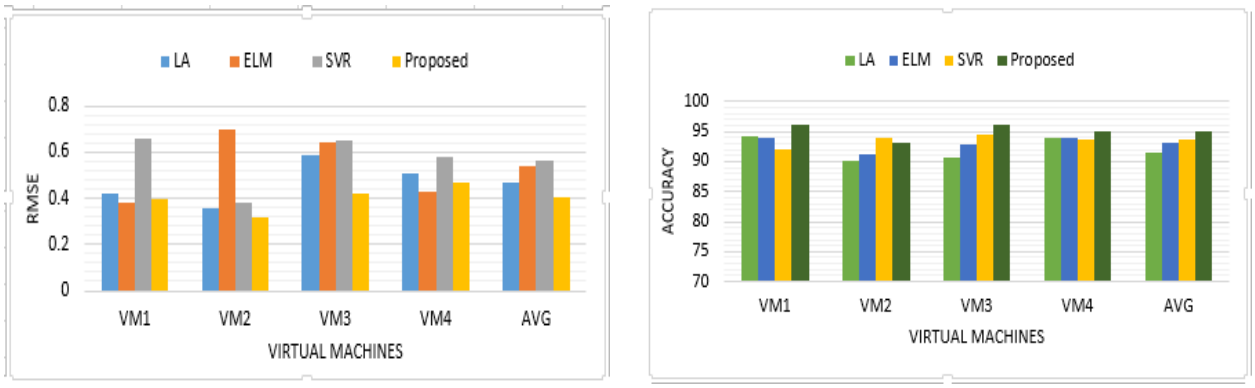
(c) VM3

(d) VM4

Figure 3.10: Accuracy comparison of different VMs at different time intervals

3.4.3 Comparison with existing approaches

The comparison of the proposed approach ETSA-LP with the existing approaches, namely, LA (Learning Automata), ELM (Extreme Learning Machine) and SVR (Support Vector Regression), is shown in Figure 3.11, in terms of RMSE and accuracy. For instance, ELM has the highest error rate, whereas the proposed ETSA-LP gives a minimum error value compared to existing approaches. Moreover, ETSA-LP outperforms existing models with the highest accuracy. The overall accuracy is improved by approximately 3% and the error rate is reduced by 2.2%. As can be seen, the proposed ensemble approach has less error rate and the best accuracy on all four VMs as compared to the existing ensemble approaches.



(a) RMSE

(b) Accuracy

Figure 3.11: Comparison of existing and proposed approach

3.5 Conclusion

This chapter discussed in detail how the resource load is predicted for a Cloud application using the proposed Ensemble Time-Series Approach for Load Prediction (ETSA-LP). The proposed ensemble framework's design methodology and the data set description are illustrated. The proposed ensemble model ETSA-LP gives the best accuracy (97.45%) among all other base models with a total execution time (0.39 ms) for the Google traces dataset. Similarly, in the PlanetLab dataset, ETSA-LP gives superior accuracy (95.78%) among the existing models with a total execution time (0.40 ms). The overall accuracy is improved by approximately 3% and execution time is reduced by 12.2%. Also, ETSA-LP has a minimum error value (0.39%). Hence, it is substantiated that the proposed ETSA-LP ensemble approach performs better than the existing individual time-series-based models.

The next chapter explains the auto-scaling approach for host load detection using VM migration and also illustrates the preciseness of the proposed host load prediction technique for performing the auto-scaling.

Chapter 4

Efficient Auto-Scaling for Host Load Prediction through VM Migration in Cloud

The previous chapter expatiated the design of the proposed ensemble approach for load prediction in the Cloud. This chapter proposes an efficient auto-scaling approach for host load detection using Virtual Machine (VM) migration. With increased resource utilization, virtualization also greatly impacts achieving desired performance. The major challenges in virtualization are detecting over-utilized or under-utilized hosts at the right time and properly scaling VMs on the accurate host.

Effective utilization and autonomous scaling of resources eventually reduce the load, energy consumption, and operating costs. In this chapter, an efficient auto-scaling approach for predicting host load through VM migration has been proposed. The ensemble method using different time-series forecasting models has been proposed to forecast the approaching workload on the host. Based on this predicted load, different algorithms have been devised to detect over-utilized and under-utilized hosts and VMs can be migrated. The designed approach has been validated by experimentation on a real-time Google cluster dataset.

The proposed technique significantly improves average CPU utilization and reduces error rate. It also minimizes response time, SLA violations, and the slighter number of migrations and scaling overhead.

4.1 Need of Auto-Scaling for Host Load Prediction

When a user leases limited virtual resources from the Cloud service provider, the cost would be negligible but the performance would be overwhelmed by the increase in load. Far from it, leasing more virtual resources will boost the performance if the resources are consumed right but also leads to excessive overhead [26]. Due to the variability in workloads, it becomes challenging for the service Clouds to maintain scalability while nourishing the user SLA [39]. Moreover, the workload for many IoT-based Cloud applications becomes complicated due to the effect of its dynamic network traffic on the host.

One way to address the host load imbalance is to leverage the capabilities of the virtualization technology. The improvement in performance on a host can be achieved by switching idle nodes/VMs to low-power VMs [116]. Most of the current approaches apply regression-based algorithms and the static utilization threshold method since it is based on a fixed threshold, but it is unsuitable for a dynamic environment. CPU utilization-based host load detection algorithms are able to provide an accurate prediction for autonomic Cloud applications [117]. Once a host overload or under-load state is detected, the next step is to select VMs to offload the host to avoid performance degradation. After selecting a VM to migrate, the host is rechecked for being over-loaded or under-loaded. If it is still considered in any state, the VM selection and migration policy is applied again to select another VM to migrate from the host. This is repeated until the host is considered to be in a normal state. An effective auto-scaling strategy that can scale VMs up and down in response to host load predictions can streamline the entire process.

Hence, in this chapter, an initiative has been made to propose a more precise and novel auto-scaling approach that predicts the host load using an ensemble technique (ETSA-LP) that integrates the prediction values of its basic models by giving weight to each prediction. Hereinafter referred, first, there is a brief introduction of the proposed auto-scaling framework in context with load predictor, auto-scaler and VM migration. Then, the proposed algorithms for dynamic load prediction, host underload and overload detection, VM selection and auto-scaling policy have been discussed. Furthermore, the experimental results, which include a comparison of the proposed auto-scaling technique with existing models has been presented. For the closing section, the chapter discusses the implication of using the proposed approach for predicting host load and scaling of VMs.

4.2 Proposed Framework for Auto-Scaling

Auto-scaling dynamically scales the resources assigned to different elastic applications with varying requirements. It is a continual intelligent process with a control loop of MAPE (Monitor, Analyze, Plan and Execute), as shown in Figure 4.1. The MAPE loop has compelling aspects to accomplish the auto-scaling process while controlling the host load. In this control loop, the resources/workload is monitored at regular set intervals, and analysis of host load through prediction and checking of threshold rules has been performed. The next step is to plan the migration process of VMs and finally send an execution decision for scaling.

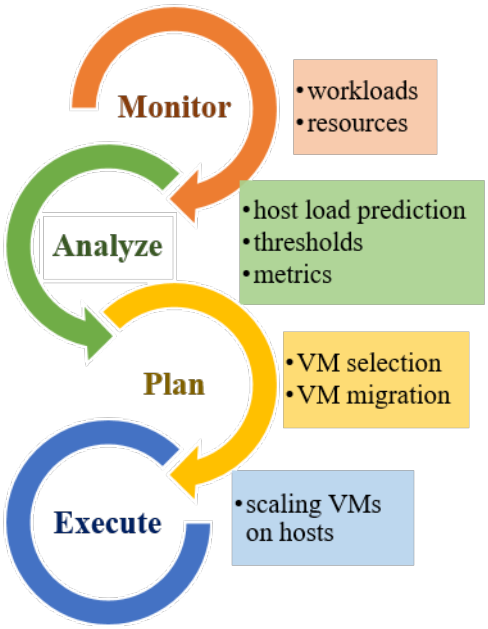


Figure 4.1: MAPE Process of Auto-Scaling

The general idea of auto-scaling is that if multiple VMs are assigned on the same host, then the VM resources may complement one another. The resource-based scaling chooses unallocated VM resources to scale up a VM residing on a particular host [41]. For instance, a host with rock-bottom CPU and memory usage can assign these VM resources to another VM residing on it, known as scaling up of a resource. Figure 4.2 shows the proposed framework for auto-scaling and VM migration in the Cloud. Cloud tenants use different resources to deploy various applications and services on Cloud. There may be additional applications running on the Cloud Virtual Data Centre and each of the applications may run on one or more VMs and physical machines. The other major components and their working in the proposed framework are as follows:

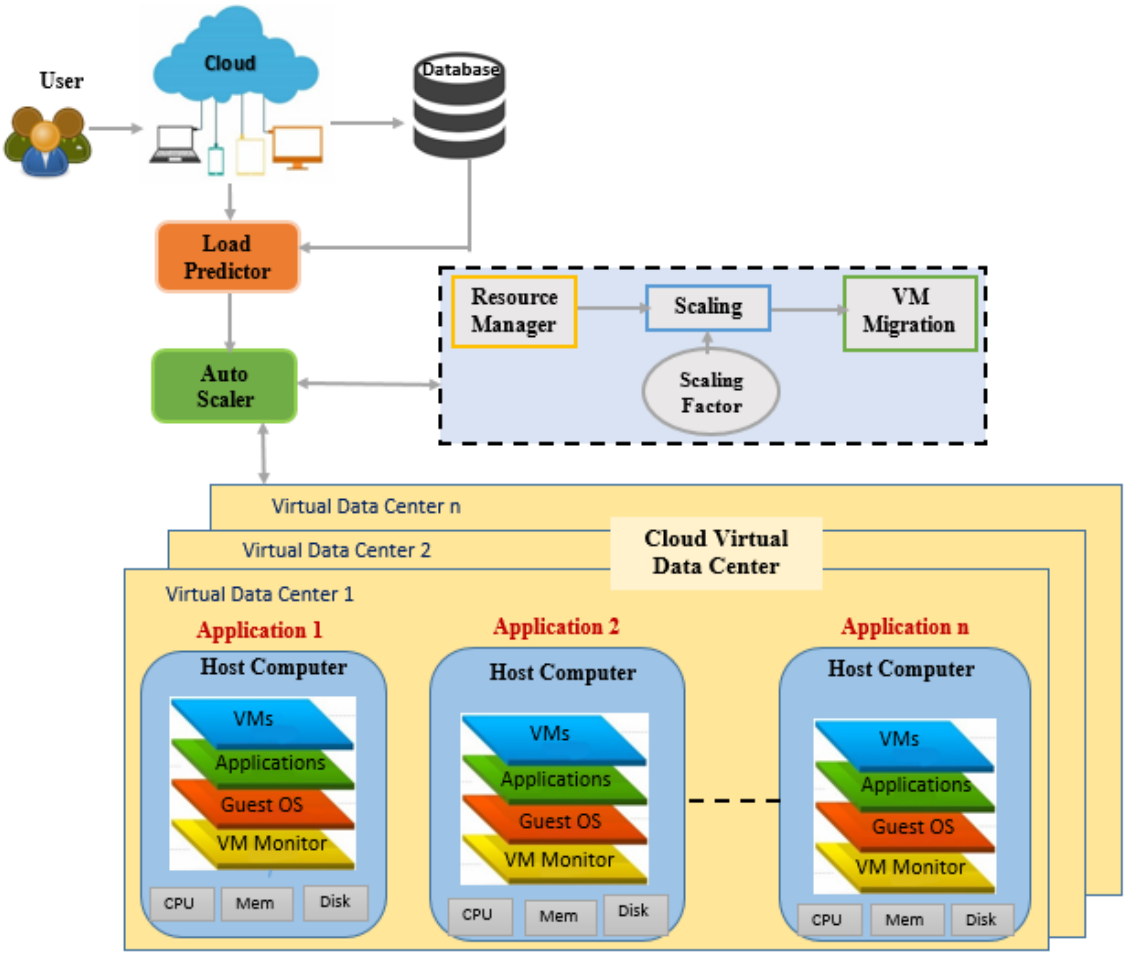


Figure 4.2: Proposed Auto-Scaling Framework

- **Database:** The database component acquires the analytical values required to assess VMs from the expert system. These past values indicate the allocated memory, CPU, response time, number of migrations and SLA violations of associated VMs. This database properly maintains all collected data as historical records. These records are used when training the prediction model to enhance its accuracy.

- **Load Predictor:** It monitors the load on each host at regular time intervals. It analyzes the history of each host load in the upcoming and past time intervals. It compares the expected number of resources with the provisioned resources and publishes scaling instructions to the auto-scaler. This paper proposes an ensemble approach to predict host load, which is more elaborated in the previous chapter.

- **Auto-Scaler:** This module is responsible for scaling up and down the resources per the host load and user requirements. Auto-Scaler is further comprised of resource manager, scaler and VM migration components.

- **Resource Manager:** This component manages the resources allocated to virtual machines, such as CPU & memory utilization. It gathers and monitors the need for resources from time to time. It also contains the log of information required to detect overloaded and under-loaded hosts.
- **Scaling:** By gathering the resource requirement from the resource manager and using the prediction results from the load predictor component, the scaling unit compares the predicted value with the scaling factor. It decides whether a resource needs to be migrated or not to balance the host load. If the predicted load is equal to or greater than the scaling factor/threshold, the host will be considered overloaded; if the load is less than the threshold, the host will be marked as under-loaded. After that, the VM migration component will perform the migration process of VMs.
- **VM Selection & Migration:** This module examines the status of each VM based on its RAM size and CPU utilization and then migrates the selected VMs of the overloaded or under-loaded hosts to the appropriately selected hosts.

Table 4.1: Notations used in the chapter

Component	Description
C_i	Clock cycles per instruction
IC	Instruction count
CR	Clock rate
V_n	No. of VMs ruuning on Nth node
$totalCU_N$	Total CPU utilization
$avgCU_N$	Average CPU utilization
$hUtil$	Host utilization
$hTotalCap$	Total capacity of a host
$hOverLoadList$	List of overloaded hosts
$LThresh$	Lower threshold value
$hUnderLoadList$	List of underloaded hosts
S_z	Size of VM
$MigVM$	VM to be migrated

4.3 Prediction of Host Load

With the advancement of private and public cloud data centers, leasing virtual machines to host applications is quite common. Usually, Cloud tenants do not choose to pay for the resources they attain but are not consumed when the load is light. This concern would be feasible only if the upcoming load can be predicted earlier before validating Quality of

Service (QoS) parameters [77]. Furthermore, there can be a probability of performance degradation in case of heavy load [110].

Though much research has been done on load prediction and time-series models, there is a need for an inclusive methodology to combine different types of prediction models. More profoundly, one prediction model would effectively predict some trends, but the same would be imprecise in other trends due to its dynamic performance [109]. Therefore, an ensemble technique has been deployed in this research work to predict future load more efficiently.

4.3.1 Dynamic Resource Prediction through proposed Ensemble Method: ETSA-LP

This research proposes an innovative Ensemble Time-Series Approach for Load Prediction (ETSA-LP) that integrates the top prediction results from compound models to forecast CPU utilization, as discussed in Section 3.3.1. This ensemble approach employs the functioning of the load predictor module, as shown in Figure 4.3.

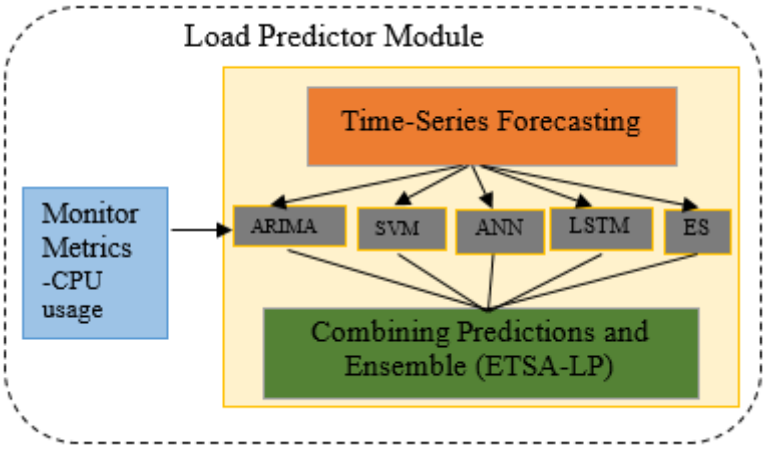


Figure 4.3: Load Predictor Module

According to the literature reviewed, efficient forecasting works with clean, time-stamped data and can identify genuine trends and patterns in historical data. Also, the choice of algorithms in time-series techniques differs entirely from those in static. Most static machine learning algorithms, like linear regression, do not have this capability as they generalize the training space for any new prediction [89]. In this context, machine learning-based models (such as SVM, LSTM, ANN, ES) are attaining inclusive attention in forecasting load nowadays because of their introspection aspect for multiple applications Hence, most state-of-the-art ensemble approaches are productive in a particular constituent model [118].

So, the primary focus of this proposed technique is to evaluate different statistical, neural, and ensemble models in their ability to predict resource load. In this chapter, an initiative has been taken to propose a more precise ensemble prediction model ETSA-LP that integrates the prediction values of five base models by giving weight to each prediction. Table 4.2 describes five base models and their tuning parameters. In this proposed ensemble model, the weights of the best model predictions from ARIMA, ANN, SVM, ES and LSTM models have been regulated using the proposed exponential weighting algorithm given in Algorithm 3.2.

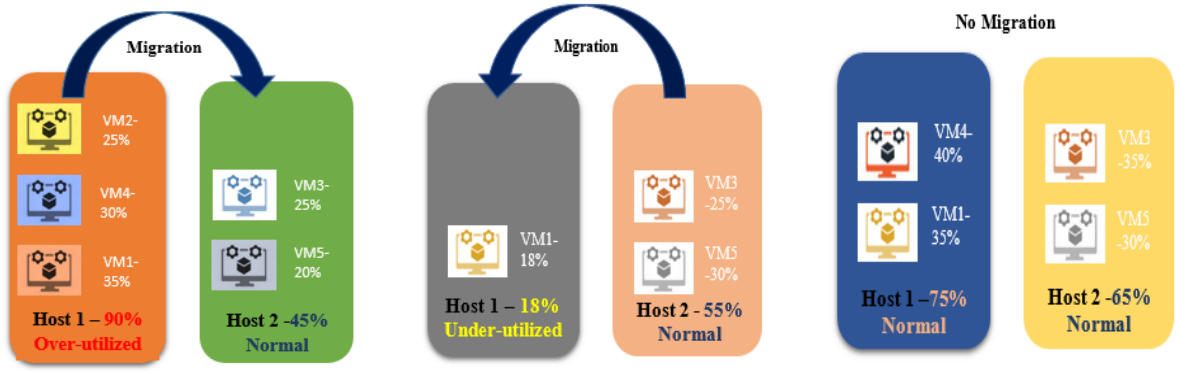
Table 4.2: Description of Base Models for Ensemble

Model name	**Method/Package**	**Tuning parameters**
ARIMA	arima()	p=2, d=1, q=2
SVM	*svm*(), kernlab	kernel="rbfdot", *type* = "nu_svm"
ANN	nnet()	maxit=100, size=5, decay=5e-4
LSTM	tslstm, keras	*hiddenlayer* = 1, neuron=4, *batchsize* = 20, epochs=8
ES	ses()	alpha=0.2, h=100

4.3.2 Dynamic Host Load Prediction Methods

In the Cloud environment, numerous VM migration approaches have been used to boost the performance of different resources by predicting the over-utilized or under-utilized state of the host. The VMs are migrated from one host to another using various Cloud resources such as CPU and memory. The migration process reassigns the workload equally between all the hosts with a defined threshold or static value. In this work, the host with a CPU utilization of 80% is considered over-utilized; therefore, its load should be divided among the other under-utilized hosts. Contrarily, the host utilizing 20% or below is assumed to be under-utilized. According to these defined criteria, the three possible conditions can occur at different time intervals (t):

- At the time (t), the three VMs are running on Host 1 and two VMs are running on Host 2, as shown in Figure 4.4. The total utilization of Host 1 becomes 90% which is an over-utilized state, but on the other hand, the total utilization of Host 2 is 45% which is a normal state. So, the VM with minimum utilization from Host 1 will be migrated to Host 2.



(a) Over-utilized Host (b) Under-utilized Host (c) No Migration

Figure 4.4: Different Scenarios of Host Load Prediction

- At the time (t+1), one VM is running on Host 1 and two VMs are running on Host 2, as depicted in Figure 4.4. Host 1 has total utilization of 18% and Host 2 has 55% of total utilization. As per the above criteria, Host 1 is in an under-utilized state. So, a VM needs to be migrated on Host 1.

- At the time (t+2), two VMs are running on Host 1 and Host 2 in a contrary manner, as presented in Figure 4.4. The total utilization of Host 1 and Host 2 is 75% and 65% respectively. Both of the hosts are in the normal state of their execution and load. Therefore, there is no need for migration in this type of scenario.

4.3.3 Host Overload Detection Algorithm

As stated earlier, when the allocation process of VMs has been initialized, the next step is to examine and analyze the hosts that are overloaded. Algorithm 4.1 represents the steps involved in host overload detection running on the Cloud. The algorithm selects the overloaded hosts and places them into an overload list and this list will act as an input for the VM selection and migration process. First of all, the input parameters required for the host overload detection algorithm are initialized to 0, i.e. hUtil (utilization of a host), hTotalCap (total capacity of a host), and hOverLoadList (List of all overloaded hosts). In the algorithm below, all the hosts are evaluated one by one to analyze whether they are overloaded or not (line 4). The total capacity of a host is considered in MIPS, which implies all the MIPS allocated to a host in the Cloud data center (line 5). Then, the current utilization of a MIPS of a host is derived (line 6). The next step is to check if the present utilization of a host is equal to or greater than the total capacity of a host in MIPS, then that host will be placed in hOverLoadList (lines 7 & 8). The final list of all detected overloaded hosts will be carried forward to the VM selection and migration

algorithm.

Algorithm 4.1 Algorithm for Overload Host Detection

Require: List of all hosts $(h_1, h_2 \dots h_n)$

1. $hUtil \leftarrow 0$
2. $hTotalCap \leftarrow 0$
3. $hOverLoadList \leftarrow 0$
4. For all hosts
   $hTotalCap.get.TotalMIPS[host]$
5. $hUtil.get.Util[host]$
6. $if(hutil >= hTotalCap)then$
7. Add host to $hOverLoadList$
8. endif
9. endfor
10. return $hOverLoadList$

4.3.4 Host Underload Detection Algorithm

As per the literature study, the majority of the researchers only focus on host overload detection. But, for efficient VM migration and scaling, it is recommended to find out whether a host is in the underloaded state (in case the host is not overloaded). In the host underload detection algorithm, the hosts running below a lower threshold are detected and placed in the underloaded list of hosts, which will be further passed to the VM selection and migration algorithm. In the proposed approach, the lower threshold is assumed as 20% of CPU utilization of a host in MIPS. According to Algorithm 4.2, the input parameters required for the host underload detection algorithm are initialized to 0, i.e. hUtil (utilization of a host), hTotalCap (total capacity of a host), and LThresh (lower threshold value). For each host running on the Cloud data center, current utilization in MIPS (hUtil) and the total capacity of a host (hTotalCap) are analyzed (lines 5 & 6). Then, the current utilization of a host is compared with the lower threshold defined (line 7). If the hUtil is less than or equal to LThresh, then the host is placed in the hUnderLoadList and if it is greater than the defined threshold, then it will pass on to the next host running on the data center (lines 8 & 9). The final list of all detected underloaded hosts will be carried forward to the VM selection and migration algorithm.

4.3.5 VM Selection Approach

The significant VM selection approach helps in improving overhead cost, energy consumption, resource utilization and SLA violation [12], [119]. This approach is basically

Algorithm 4.2 Algorithm for Underload Host Detection

Require: List of all hosts $(h_1, h_2 \dots h_n)$

1. $hUtil \leftarrow 0$
2. $hTotalCap \leftarrow 0$
3. $LThresh \leftarrow 0$
4. For all hosts
5. $hTotalCap = hTotalCap.get.TotalMIPS[host]$
6. $hUtil = hUtil.get.Util[host]$
7. $LThresh = hTotalCap * 20/100$
8. $if(hUtil <= LThresh)then$
9. Add host to $hUnderLoadList$
10. endif
11. endfor
12. return $hUnderLoadList$

applied to choose the list of VMs on a host that has been declared as overloaded or underloaded. In this research, the Minimum Size Maximum Utilization (MSMU) policy has been introduced in which a VM having minimum size but maximum resource utilization is selected. When the VM is chosen with higher resource utilization than other VMs on the host, the risk of over-utilization and under-utilization is reduced.

The complete working of the proposed MSMU algorithm is presented in Algorithm 4.3. As mentioned earlier, the list of overloaded and underloaded hosts has been forwarded to the VM selection and migration algorithm. Therefore, hOverLoadList and hUnderLoadList obtained from Algorithms 1 and 2 will act as input in Algorithm 4. After identifying the hOverLoadList, the further step is to choose the appropriate VM from the overloaded host and migrate it onto a relevant host. The beginning step is to sort the list of VMs in the descending order of their resource usage (line 1). Then, for each VM, the size of the VM and utilization have been analyzed (lines 3 & 4). The MSMU factor has been derived by calculating the current utilization of VM and the size of VM (line 5). After calculating MSMU, it has been compared to choose the finest VM with a higher MSMU value than other VMs on the overloaded host (lines 6-8). In the end, the algorithm will return the list of appropriate VMs to be migrated to some other host. The complexity of this policy is correlative to the number of VMs leased by the overutilized hosts.

4.3.6 Proposed Auto-Scaling Algorithm

The proposed auto-scaling strategy decides on the scale-out or scale-in of the resources focusing on the regulation and utilization of resources and the upper threshold value/ scaling factor. As stated earlier, the scaling factor of each host is defined on the basis of the amount of resource utilization such as the CPU usage above the upper threshold is

Algorithm 4.3 VM Selection Algorithm using MSMU

Require: List of all VMs $(vm_1, vm_2 \dots vm_n)$

1. Sort list of VMs in decreasing order of their utilization
2. For each vm in list of VMs do
3. $S_z \leftarrow RSize_{vm}$
4. $Util_{vm} \leftarrow CPUutilizationinMips/VMMips$
5. $MSMU = currentCPUutilizationofVM/(S_z + currentCPUutilization)$
6. $if(MSMU > MinMetric)then$
7. $MigVM \leftarrow MSMU$
8. $MigVM \leftarrow VM$
9. endif
10. endfor
11. return $MigVM$

considered as the over-utilized host/VM and the CPU usage less than the lower threshold is considered as the under-utilized host/VM. The proposed algorithm for the Auto-scaling policy is depicted in Algorithm 4.4. The core objective of this algorithm is, to sum up, the number of active hosts which are either over-utilized or under-utilized at time t (lines 4-6). The monitoring phase performs the scale-in and scale-out decision at time t if partial host loads are beyond the overlying threshold value (line 8) or lower than the bottom threshold value (line 9).

Algorithm 4.4 Proposed Algorithm for Auto-Scaling Policy

Require: List of all VMs and hosts $(vm_1, vm_2 \dots vm_n), (h_1, h_2 \dots h_n)$

1. Set lower and upper threshold for CPU utiization
2. Set minMetric=1 & maxMetric=6
3. For each vm in list of VMs do
4. Calculate Average CPU utilization at time t
   $avgCU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} CU_x}{totalCU_N} * 100$
5. Calculate number of overutilized hosts
   $CallAlgorithm1$ ▷ Algo for Host Overload Detection
6. Calculate number of underutilized hosts
   $CallAlgorithm2$ ▷ Algo for Host Underload Detection
7. Update minResourceCount, maxResourceCount
8. $if(minResourceCount < minMetric)then$
   $CallAlgorithm3$ ▷ Scale In Decision
9. $if(maxResourceCount > maxMetric)then$
   $CallAlgorithm3$ ▷ Scale Out Decision
10. Else System load is in normal mode
11. endif
12. endfor

4.3.7 Evaluation Metrics

The evaluation parameters used for the formulation of the proposed model are the RMSE (Root Mean Square Error), MAPE Mean Absolute Percentage Error), Average CPU utilization, response time, number of virtual machine migrations, accuracy and scaling overhead.

a) **RMSE and MAPE**: For experimental evaluation, RMSE and MAPE are evaluated to calculate the percentage error. RMSE is defined in Equation 4.1 where X_t is the actual output, x_t is the predicted output and n specifies the total sum of observations in the dataset. The mathematical formulation to calculate MAPE is given in Equation 4.2. The lower value of RMSE and MAPE validates to be a more precise prediction technique.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(x_t - X_t)^2} \tag{4.1}$$

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{(x_t - X_t)^2}{x_t} \tag{4.2}$$

b) **Accuracy:** Accuracy is the degree to which the outcomes of an actual calculation or measurement are grouped around the accurate value. The mathematical notation to compute accuracy is given below in Equation 4.3:

$$Accuracy = \frac{tp + tn}{tp + fp + tn + fn} * 100 \tag{4.3}$$

Here, tp, tn, fp and fn denote the count of true positives, true negatives, false positives and false negatives, respectively.

c) **CPU utilization**: The CPU utilization is quoted as the amount of work managed and utilized by the CPU of VMs residing on the host in the Cloud. In other words, it is the present CPU usage of processed user requests in percentage. The performance of the proposed auto-scaling approach has been validated with the CPU utilization metric. The very first step is to calculate the total CPU utilization and then the average CPU utilization is calculated as per the given formulation:

$$totalCU_N = \frac{C_i * IC}{CR} \tag{4.4}$$

In Equation 4.4, C_i refers to the clock cycles per instruction, IC is the instruction count and CR denotes the clock rate. By taking this total CPU utilization into account, the average CPU utilization is calculated as follows:

$$avgCU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} CU_x}{totalCU_N} * 100 \tag{4.5}$$

In above equation 4.5, $avgCU_N$ is calculated for Nth node at a particular time interval. considering V_N as the number of VMs running on the Nth node and J_N is the number of jobs assigned to V_N VMs. CU_x is the CPU utilization of K jobs running on M VMs on the Nth node.

d) **Response Time**: Response time is the period of time between the request submitted to the time of response. The response time of any job, task, VM, or application must be quick in a Cloud environment. It is one of the crucial metrics to validate user QoS and the minimum is the response time; the better is the accuracy of the system [41].

e) **No. of Migrations and Scaling Overhead**: The performance of the proposed efficient auto-scaling approach can be analyzed with the discrepancy that occurred between the number of estimated resources and the number of consumed resources. The VMs are rented and released according to demand with mitigation of host load. When there is a need for VM to be migrated, there should not be many occurrences. It affects the execution behavior of the host/VM regarding cost, energy and scaling overhead. The scaling overhead may upsurge with the rise in the resource count (VMs) consumed and therefore, it must be minimized to improve efficiency and performance.

f) **SLA Violations**: The SLA agreement is an important aspect that ensures the maximum availability of cloud services to the end users. The SLA violation is calculated as per the following equation 4.6. RT_t denotes the response time at time t, S^{rt} is the response time as per the SLA, and nr is the count of the requests received at time t.

$$SLAV = \sum_{i==1}^{nr} RT_t - S^{rt} \tag{4.6}$$

4.4 Experiments and Evaluations

This section describes the dataset in more detail and evaluates the suggested method against the state-of-the-art methods for optimal results.

4.4.1 Dataset Description

A real cloud workload trace from Google has been used as a baseline benchmark to validate our proposed approach with host load prediction effectively. As discussed above, in the Google dataset, the CPU usage traces represent the CPU usage in the Google cluster, as shown in Table 3.2 [114]. All these measurements are regulated between 0 and 1 by the relative maximum values for which the machine has been furnished.

4.4.2 Result Analysis

The experimental analysis of the proposed auto-scaling approach is highlighted in the present section.

a) ***Prediction Accuracy of Ensemble Model:*** As described earlier, thorough testing has been conducted prudently to assess the efficiency of the proposed approach on the Google cluster. Firstly, the performance of five time-series models, statistical ARIMA, ANN, SVM, LSTM and ES is dignified and then with their resulting best predictions, the proposed ensemble model is evaluated. Table 4.3 and Figure 4.5 depict the comparative results of the five base models tested on the Google cluster for analyzing CPU usage. RMSE and MAPE are used to calculate the accuracy of the prediction model.

Table 4.3: Comparison of different models tested on Google Traces

Metrics	**ARIMA**	**SVM**	**ANN**	**LSTM**	**ES**	**Proposed**
R	0.88	0.85	0.72	0.77	0.9	0.65
R^2	0.77	0.72	0.52	0.59	0.81	0.43
RMSE	0.46	0.4	0.43	0.53	0.75	0.39
Accuracy (%)	0.78	0.91	0.84	0.89	0.9	0.97
Total Time (ms)	0.52	0.64	0.51	0.44	0.43	0.39

b) ***Error rate of Ensemble Model:*** The error values corresponding to RMSE and MAPE are shown in Figure 4.6. Regarding Table 4.3, the RMSE value for ES is maximum (2.82%), whereas the proposed model for load prediction has a minimum

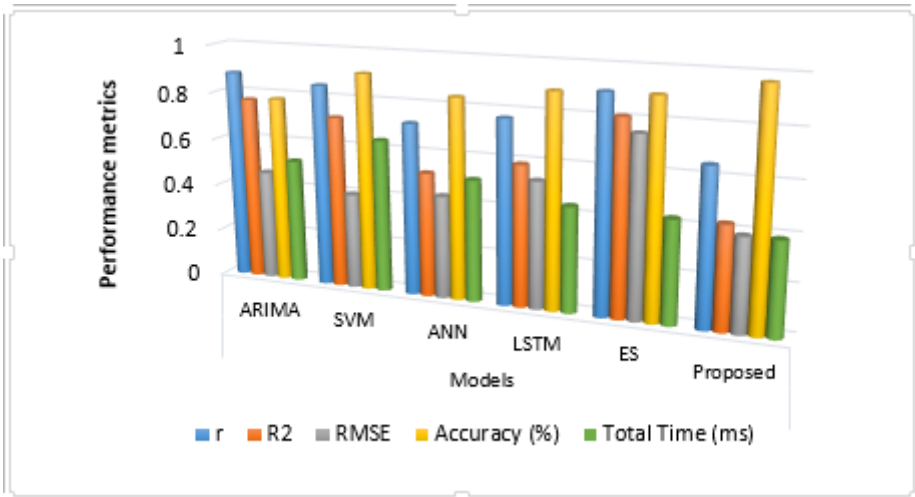


Figure 4.5: Performance Metrics of different models on Google traces

value of error (0.39%). The accuracy measure of ARIMA is minimum (0.78%) and the proposed approach has maximum accuracy among other models (0.97%). Also, the time taken for execution of the SVM model is the highest (0.64 ms) but the proposed model again outperforms other models with the minimum time for execution (0.39 ms).

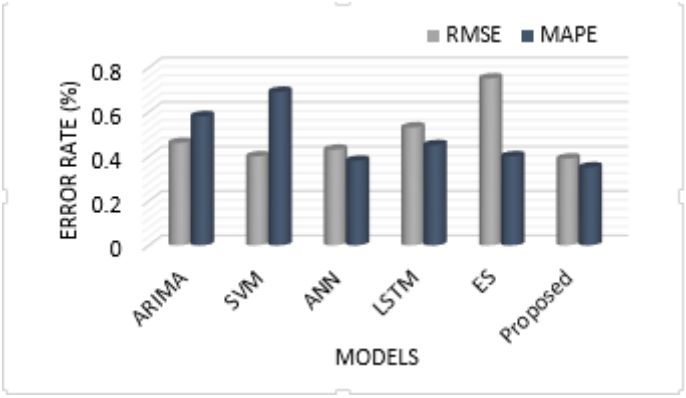


Figure 4.6: RMSE and MAPE

c) ***Average CPU utilization:*** The CPU utilization is the number of jobs addressed by the CPU of VMs on a particular host., The technique proposed in this research combines both reactive and proactive techniques, giving a persistent performance with average CPU utilization. As discussed earlier, if the CPU utilization exceeds the defined threshold, the host will be considered over-utilized and must be migrated to another host. Also, if it is less than the minimum utilization, then a VM should be placed onto this host to balance the load. The proposed auto-scaling approach has been compared with existing scaling approaches as shown in Figure 4.7. For

instance, the average CPU usage of other current techniques comes out to be lower as well as higher at different time intervals. Contrarily, the average utilization of the proposed approach is maintained between upper (80%) and lower (20%) bounds. It has come out to be in the normal range for most of the time. The results prove that if the load can be effectively predicted with former awareness of resource utilization, then the over-utilization or under-utilization of resources can be either handled commendably.

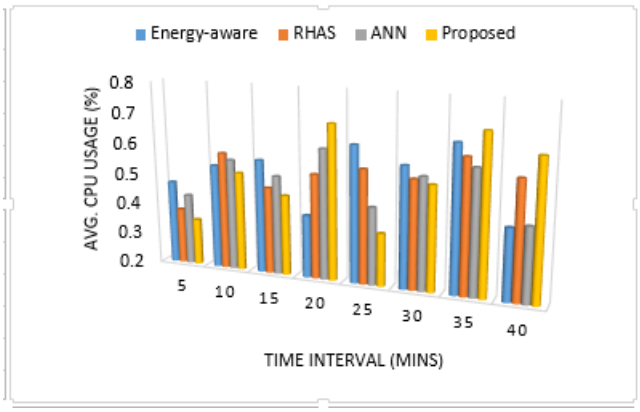


Figure 4.7: Average CPU utilization

d) ***Response Time:*** As discussed above, the response time of an application needs to be quick and must be agreed to 1 as per SLA. Figure 4.8 shows the comparative outcome of response time for the proposed and existing approaches. It is evident from the results that the proposed auto-scaling method has a relatively low response time (2.2%) within five minutes of interval. It has been improved by (3.7%), and relatively the accuracy is increased by (5.67%) compared to other existing models. Therefore, it has been proven that the proposed efficient auto-scaling approach can work well for dynamic host load prediction in the Cloud having maximum accuracy and minimum response time.

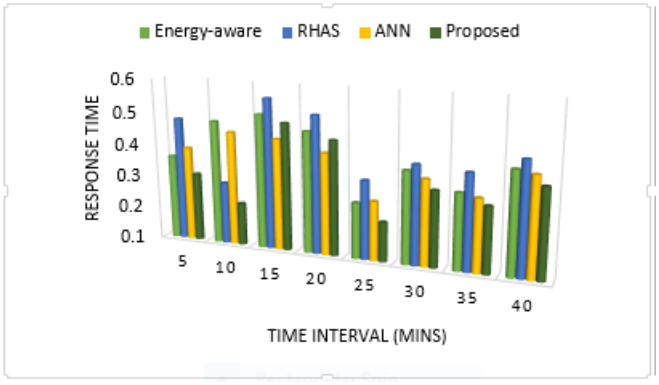


Figure 4.8: Response Time

e) ***No. of Migrations and Scaling Overhead:*** The accuracy analysis clearly proved the efficient performance of the prediction model and originated that the proposed approach gives superior conduct for host load prediction in comparison with existing works. Also, our proposed prediction model is able to fulfill the resource demand in normal as well as peak hours.

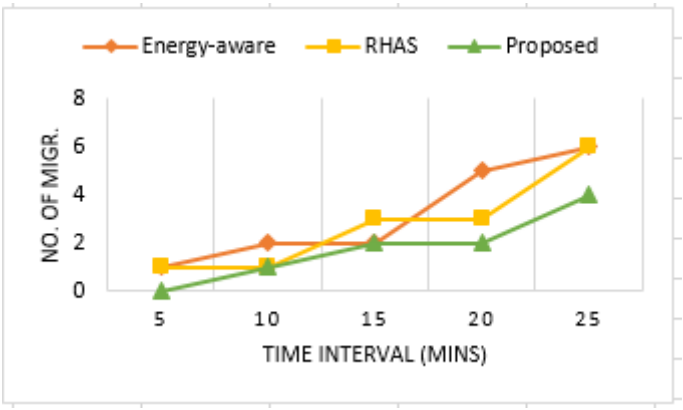


Figure 4.9: No. of Migrations

It can be seen in Figure 4.9 that the proposed auto-scaling approach has the least number of VM migrations within different time intervals, as compared with other approaches. In the RHAS technique, the number of migrations was approx. (1.12%) at the initiation time, but as the time interval increases, the number of migrations also rises by (5-6%), leading to low accuracy and poor performance. Contrarily, the average number of migrations for the proposed approach is monitored between (2-4%), which is comparatively less. Moreover, the experimental results for scaling overhead have been shown in Figure 4.10 and are the lowest in the proposed auto-scaling approach. The number of migrations has been improved by 2% and scaling overhead by 2.6% compared to existing auto-scaling approaches.

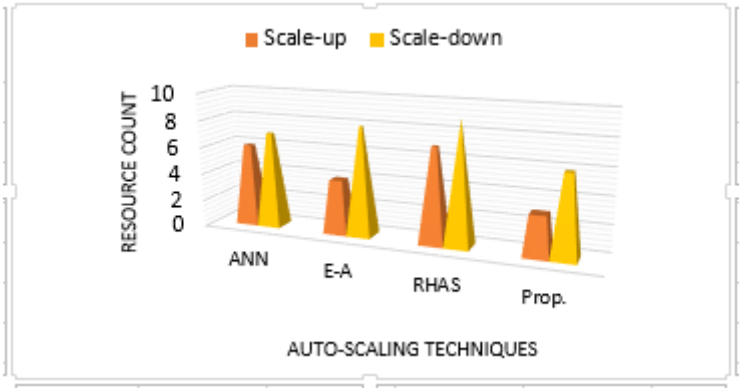


Figure 4.10: Scaling Overhead with various Auto-Scaling techniques

f) ***SLA Violations:*** The SLA compliance is a crucial aspect that certifies the utmost accessibility of various services and applications to Cloud users. If any SLA violation occurs, the tenants must pay a penalty. The SLA violation is examined depending on an application's major parameter response time. Figure 4.11 shows that the proposed technique has been evaluated with approximately 1% SLA violation and this results in the efficient accessibility of resources with better QoS.

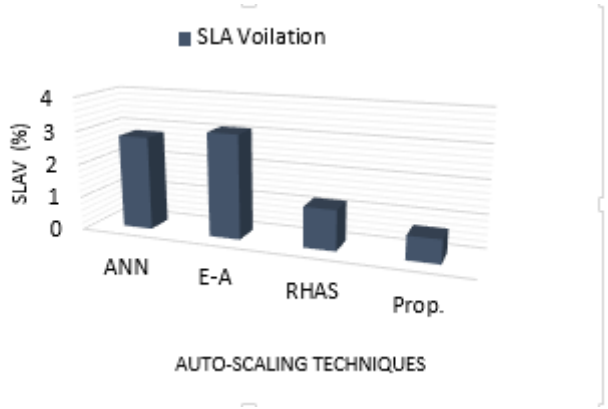


Figure 4.11: SLA Violations

As discussed earlier, if the CPU utilization comes out to be greater than the defined threshold, the host will be over-utilized and must be migrated to another host. Also, if it is less than the minimum utilization, then a VM should be placed onto this host to balance the load. The proposed auto-scaling approach has been validated on Google Cloud for host load prediction in terms of CPU and compared with existing scaling approaches.

It is clear from the above-stated experimental results that the proposed auto-scaling technique is an efficient and dynamic approach that provides the requisite amount of QoS to the end users and the application providers. For example, in any social media web application such as Flipkart, Amazon, etc., the network traffic may vary due to the dynamic number of users and resources consumed. If the load fluctuates frequently and exceeds the defined threshold, the system may be over-utilized or under-utilized, leading to data loss and poor performance. Moreover, even a few milliseconds of delay in response time can cause the whole system to cease functioning. So, to avoid such circumstances, an auto-scaling approach can be a solution to scale up and down the resources automatically. Moreover, the proposed auto-scaling approach using host load prediction can minimize response time and increase the system throughput for large-scale applications.

4.5 Conclusion

The tremendously increasing use of Cloud services and applications raises the issue of non-uniform load variations, which results in ambiguous scaling decisions to a greater extent. In this research, an initiative has been proposed to design and implement an efficient auto-scaling approach for host load prediction through VM migration in the Cloud. The intended plan yields the merits of the auto-scaling feature with host load prediction by deploying an ensemble method using different time-series forecasting models. The simulation results have improved accuracy by 5.7% and response time by 3.7%. It also consistently reduces error rate by 2.7%, no. of migrations by 2% and scaling overhead by 2.6% compared to existing auto-scaling approaches. Moreover, the proposed technique has been evaluated with minimum SLA violation, and this results in the efficient accessibility of resources with better QoS.

The upcoming chapter presents the exploratory outcomes attained by affirming the proposed ETSA-LP and auto-scaling approaches on the actual cloud platform. Firstly, the outcomes of the load prediction technique have been discussed. Next, the results of the prediction-based auto-scaling approach are detailed, and the performance is validated using different evaluation metrics. Also, a case study of a fog-based framework has been introduced to enhance the requisite performance of the proposed auto-scaling approach.

Chapter 5

Verification and Validation of the proposed approaches

The previous chapter explained the host load prediction-based auto-scaling approach for the IoT-based cloud application. An actual cloud testbed is set up for conducting the experiments and the results are evaluated based on various performance metrics. This chapter deals with verifying and validating the proposed load prediction and auto-scaling approaches on the Cloud.

This chapter elaborates on testing and validating the proposed load prediction approach using the Google Cloud Platform (GCP). The chapter also provides the load variations of different VMs deployed on GCP. Finally, the results of the proposed auto-scaling approach are validated on the AWS (Amazon Web Services).

Furthermore, to validate the effectiveness of the proposed auto-scaling approach, the case study of a fog-based IoT application has been employed. A Fog-Enabled Auto-Scaling Technique (FEAST) has been proposed to predict resource usage concerning CPU, memory, disk, and network utilization using the LSTM model. Using the prediction results from the LSTM model, the proposed auto-scaling approach has been deployed to detect overload and underload hosts. Also, VMs have been migrated based on the host load status. Comparing the suggested method to the current fog-based auto-scaling techniques, the experimental findings demonstrate that it improves cost, resource utilization, SLA violation, response time, and delay violation.

5.1 Validation of Proposed ETSA-LP Approach

This research proposes a dynamic Ensemble Time-Series Approach for Load Prediction (ETSA-LP) that combines the best prediction results from compound models (ARIMA, ANN, SVM, LSTM and ES) to predict CPU and memory utilization. A dynamic exponential weighting algorithm has been proposed, in which a distinct model's best prediction outcome selected from diverse models has been dynamically weighted. In the previous section 3.4.2, the base predictors ARIMA, ANN, SVM, ES and LSTM have been compared with the proposed ETSA-LP to assess the best performance using the RMSE, accuracy and execution time metrics, using the Google traces and PlanetLab datasets.

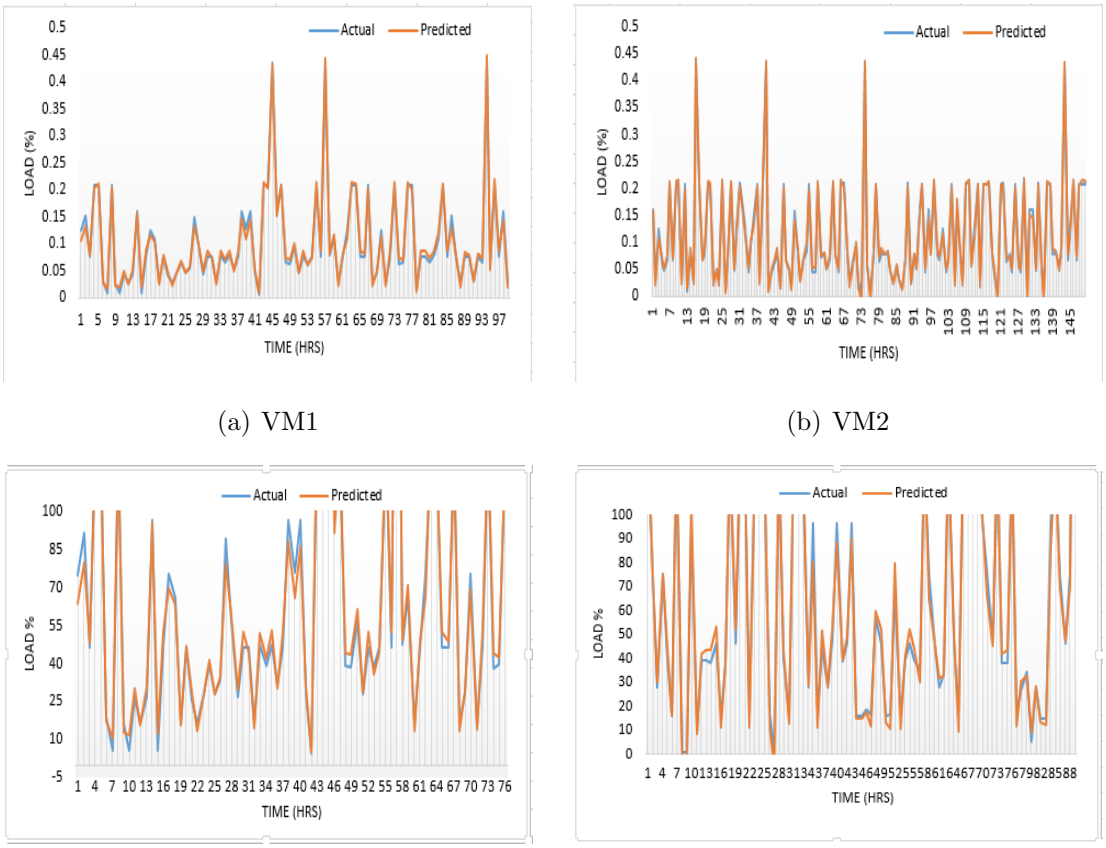
5.1.1 Experimental Setup: ETSA-LP approach

For proper validation and experimentation, it is always recommended to use real workflow traces. Google offers a Cloud Platform (GCP) suite of Cloud Computing services deployed on the same infrastructure used by Google Search, Gmail and YouTube. Besides the management tools, GCP offers integrated and innovative services such as computing, data storage and analytics and machine learning. In this research work, four heterogeneous VMs are created for parallel execution and validating the performance of ETSA-LP in a cloud environment as shown in Table 5.1. All four VMs have diverse configurations to work in a distributed manner for executing applications.

Table 5.1: Selected VMs and their average load

VMs	VM Name	Load Type	Avg Load%	SD%
VM1	Gcpvm1	Low	38.31	3.32
VM2	Gcpvm2	Low	35.45	12.78
VM3	Gcpvm3	High	78.71	6.90
VM4	Gcpvm4	High	72.56	14.55

The average load changes at different time intervals based on the accomplished process. As observed practically and depicted in Figure 5.1, VM1 and VM2 have low loads as compared to VM3 and VM4. The reason behind this is dynamic resource allocation and the requirements of the users.



(a) VM1

(b) VM2

(c) VM3

(d) VM4

Figure 5.1: Actual and predicted load of different VMs

Based on heterogeneous virtual machines, the average load variations at different time intervals on GCP have been shown in Figure 5.2. Here, one of the VM instances has been monitored as having a high load, but the same VM may have a low load at another time period. Similarly, different VMs have been deployed, and their load has been monitored at different intervals.

As per the experiments, it has been observed that the proposed ensemble model ETSA-LP gives the best accuracy (97.45%) among all other base models with a total execution time (0.39 ms) for the Google traces dataset. Similarly, in the PlanetLab dataset, ETSA-LP gives superior accuracy (95.78%) among the existing models with a total execution time (0.40 ms). The overall accuracy is improved by approximately 3% and execution time is reduced by 12.2%. Also, the RMSE value for ES is maximum (2.82%) whereas ETSA-LP has a minimum error value (0.39%). Hence, it is substantiated that the proposed ETSA-LP ensemble approach performs better than the existing individual time-series-based models.

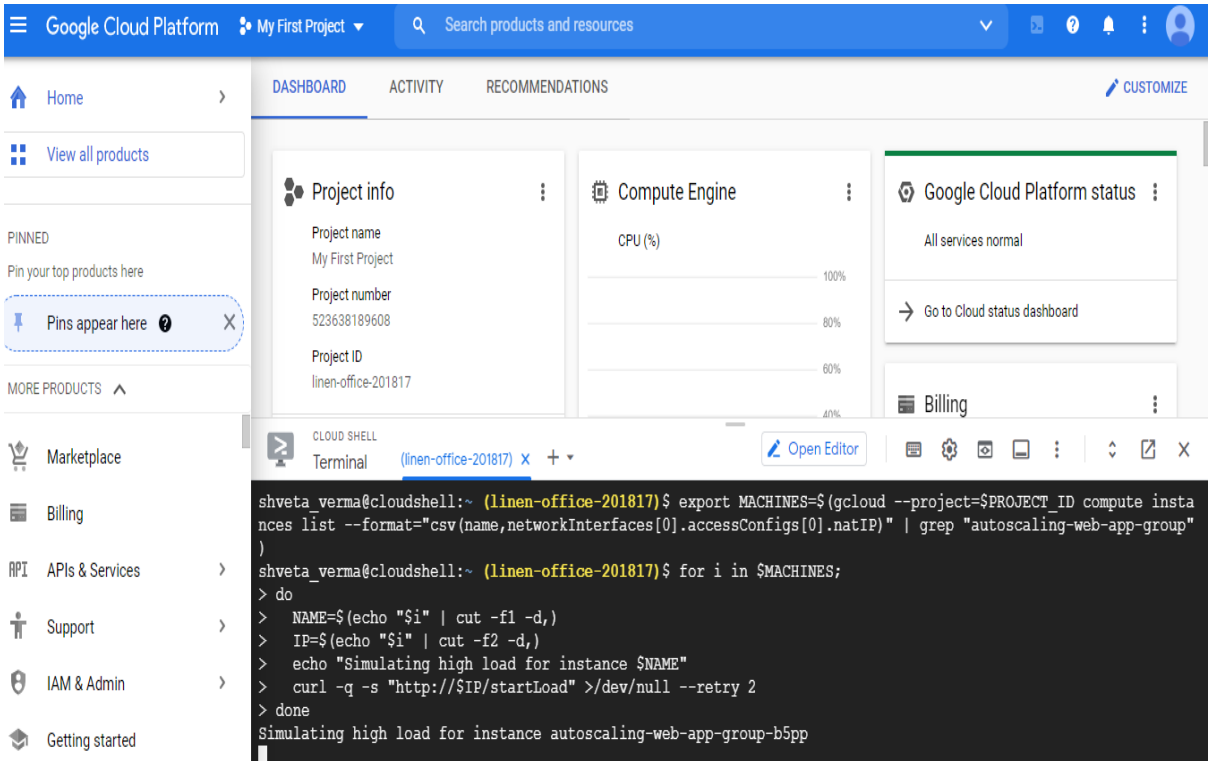


Figure 5.2: Load variations on GCP Cloud

5.2 Validation of proposed auto-scaling approach

An efficient auto-scaling approach for predicting host load through VM migration has been proposed in this research work. The ensemble method using different time-series forecasting models has been submitted to forecast the approaching workload on the host. Based on this predicted load, different algorithms have been devised to detect over-utilized and under-utilized hosts and VMs can be migrated. The designed approach has been validated by experimentation on a real-time Google cluster dataset. The results have been compared with existing techniques to assess the best performance using the error rate, accuracy, no. of migrations, and scaling overhead under Section 4.4.2.

5.2.1 Experimental Setup: Auto-Scaling approach

Amazon Web Services (AWS) offers metered, pay-as-you-go cloud computing platforms and APIs to consumers, businesses, and governments. This is frequently used in conjunction with auto-scaling, which enables clients to consume more processing power during periods of high application usage and scale back to lower expenses during periods of lower traffic. In August 2006, AWS introduced the Amazon Elastic Compute Cloud (EC2) service, enabling programmers to start and stop instances (machines) automatically.

For the validation of the proposed auto-scaling approach on the Cloud, four heterogeneous VMs have been created with diverse configurations. As per the host over-utilization and under-utilization scenarios, the upper threshold has been set to 80%, and the lower threshold has been set as 20% for CPU utilization. According to these CPU utilization factors, the auto-scaling action has to be performed whether to scale up or scale down the VMs. In other words, there may be a need to launch a new VM or terminate the existing VM. As shown in Figure 5.3, the termination and launching of VMs have been monitored on AWS Cloud as the load variations occur on the host.

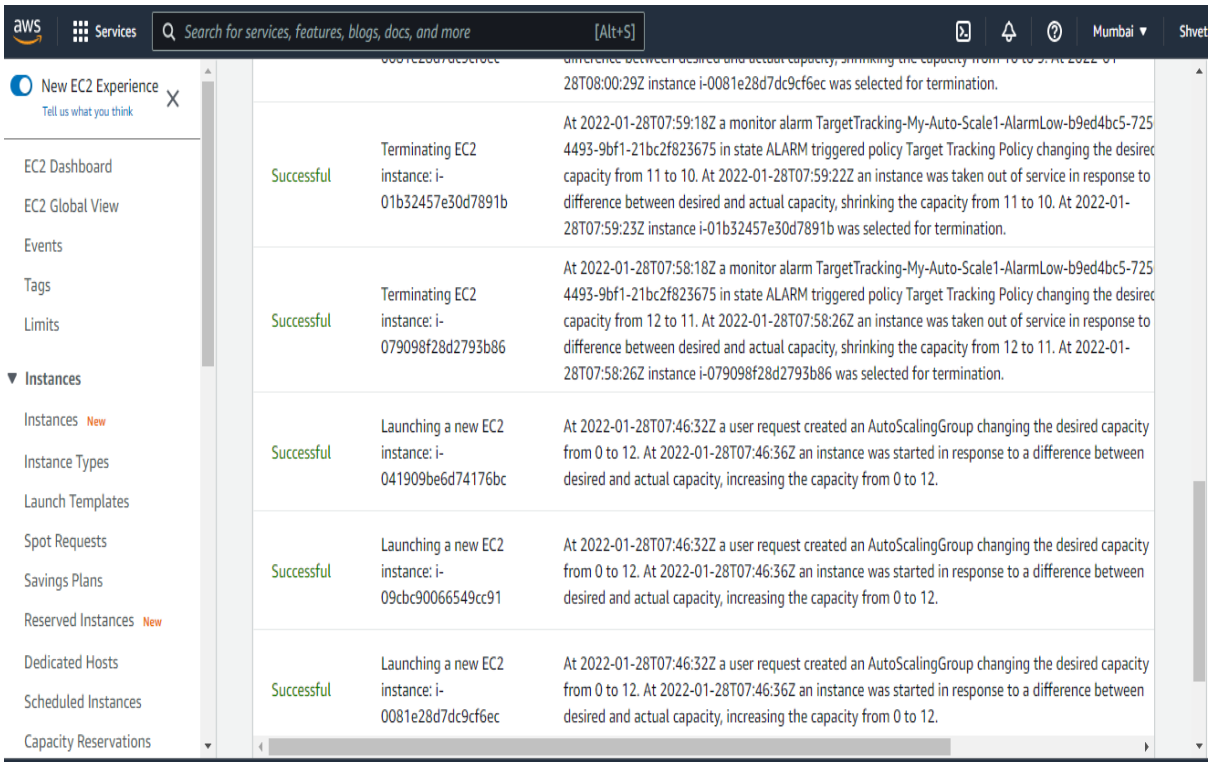


Figure 5.3: Launching and termination of VMs on AWS

Regarding the experimental results, the proposed auto-scaling approach has been validated on AWS Cloud for host load prediction in terms of CPU and compared with existing scaling approaches. The simulation results have improved accuracy by 5.7% and response time by 3.7%. It also consistently reduces error rate by 2.7%, no. of migrations by 2% and scaling overhead by 2.6% compared to existing auto-scaling approaches. Moreover, the proposed technique has been evaluated with minimum SLA violation, resulting in efficient accessibility of resources with better QoS.

5.3 Fog-based IoT Application as a Case Study

The Internet of Things (IoT) is the network of billions of intelligent devices. It offers the ability to transform simple items into real-time smart devices that can run independently without human intervention. All information obtained from the installed IoT devices is processed, stored, and analyzed before being made available to users in an understandable manner [120]. Since IoT technology is expanding quickly, the fog computing model is beginning to take shape as an intriguing approach to deliver networking, computing, and storage capabilities. A unique layered distributed computing architecture called fog computing mediates between cloud servers and Internet of Things devices [121]. To satisfy the processing needs of latency-sensitive Internet of Things applications, fog nodes can be intelligent gateways, switches, routers, base stations, set-top boxes, proxy servers, or other fog devices [122].

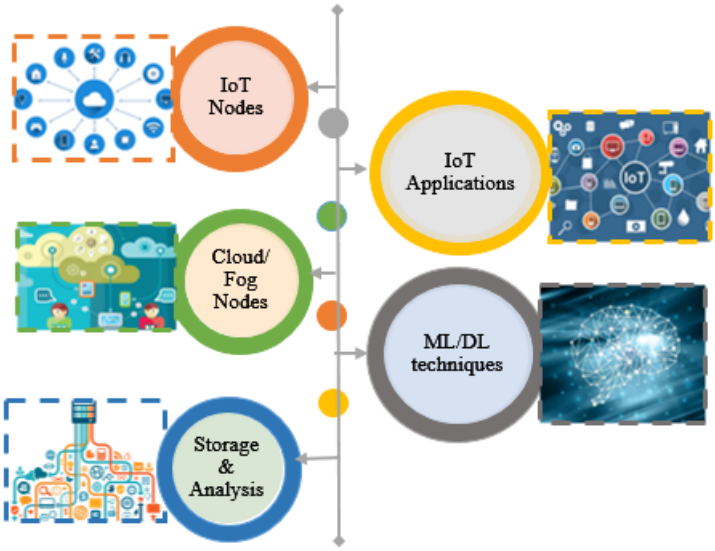


Figure 5.4: Key elements of basic IoT structure

The recent growth of real-time IoT has triggered the emergence and enlargement of concurrent data analytics, such as Machine Learning (ML), Deep Learning (DL), and computing infrastructures [123]. Figure 5.4 shows the key elements of basic IoT structure where IoT nodes, represented as IoT devices such as sensors and actuators, lie at the network's edge. The Cloud/Fog nodes assist IoT devices by facilitating advanced computing, storage, and processing capabilities. These nodes comprise data centers, which can handle and analyze intense data using ML/DL techniques, data sharing, etc. The data from various Cloud and IoT applications are clustered and explored to generate services for the end user[124]. IoT applications use this information with different middleware

IoT platforms, reinforcing the elements in several ways throughout the infrastructure [125].

In order to meet the QoS requirements, the majority of research works concentrate on machine learning-based methods for serving heterogeneous IoT workloads. However, using machine learning-based techniques can be sometimes insufficient to achieve high performance since IoT workloads are heterogeneous and dynamic. Based on the findings of this literature, decision-making and resource scaling can be enhanced by the deep learning technique with higher accuracy compared to the other approaches examined [74]. The outcomes show that these raise the performance standards. More work is needed to efficiently handle IoT workloads, even if several research works have already used machine learning-based strategies to address resource auto-scaling mechanisms.

5.4 Proposed Framework: FEAST (Fog Enabled Auto-Scaling Technique)

The Fog-Enabled Auto-Scaling Technique (FEAST) for resource scaling, as seen in Figure 5.5, is thoroughly explained in this section. The proposed FEAST architecture is comprised of three levels: the cloud, fog, and IoT device tier. In this three-tier architecture, the IoT device tier sits at the bottom and is made up of IoT devices (such as sensors, smartphones, tablets, and other gadgets) that are used to retrieve the necessary data from the IoT ecosystem and transmit it to the top layer via the access point.

The cloud layer and the IoT device tier are separated by the fog tier, which is the intermediate tier. This is in charge of receiving requests from the lower tier (i.e., IoT device tier) and either serving them through the cloud gateway to the upper tier (i.e., cloud tier) or employing fog nodes situated/settled at this tier. IoT queries that require minimal processing, networking, and storage resources as well as those with quick response times (i.e., real-time requests) are often handled at this tier, with the remaining requests being forwarded to the cloud tier. The cloud tier, which is the top layer, consists of numerous cloud servers with substantial processing and storage capacities to support pay-per-use IoT applications.

a) **Admission Manager**: After the IoT device layer receives the IoT requests, the admission manager is in charge of recognizing real-time requests based on their deadline parameter and forwarding them to the fog tier or the cloud tier. The admission control component verifies the deadline parameter of each IoT request before accepting it from the IoT device tier. If the deadline is below the threshold,

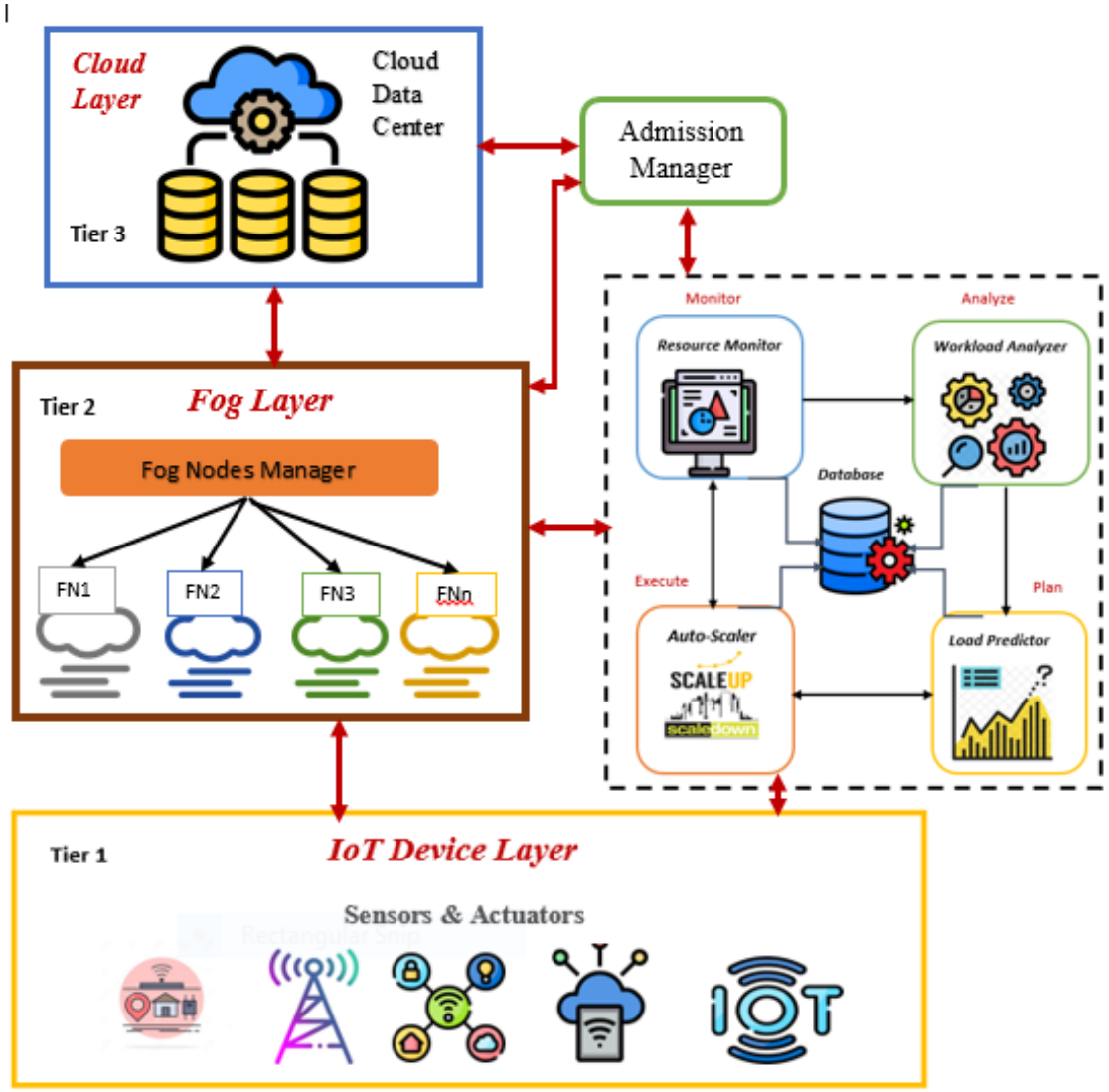


Figure 5.5: Proposed Auto-Scaling Framework: FEAST

the IoT request will be fulfilled at the fog tier otherwise it will be moved to the cloud tier. Lower threshold values result in fewer requests being sent to the fog tier. Conversely, a threshold with a higher value transfers more requests into the fog tier. Consequently, a precise threshold value is required.

b) **Fog Nodes Manager**: A collection of fog nodes serves as the computing resources of the fog layer. The real processing power, including CPU, RAM, and storage capacity, is supplied by the fog nodes in order to deploy and run IoT services. With expanded functionality to execute IoT services and manage fog resources, the fog nodes manager is in charge of the fog nodes. Requests to use IoT services can be sent to it by admission control. Additionally, in cases where the current fog node is unable to perform IoT services or fails, the fog control node has the capability to

forward IoT requests to other fog colonies or the cloud tier.

c) **MAPE Module**: The resource provisioning module, which is based on the MAPE control loop, is the primary element of the proposed FEAST framework. A brief description of the four parts of the MAPE module- monitor, analyzer, planner, and executor, is given below:

- Monitor: This module gathers data regarding IoT requests and fog resources. The resource monitor sub-component gathers data regarding the fog nodes set up at the fog tier. In order to accomplish this, the monitor component makes reference to the fog control node and keeps track of the necessary data regarding the fog resources (such as the CPU, RAM, disk and network usage of the fog nodes) as well as the IoT requests (such as the cost and response time). Lastly, the data is entered into the database so that other parts may access it.

- Analyze: This component processes the data collected by the monitor component to produce estimates of future demands. The system's future requirements are estimated based on the evolving needs of IoT users. The history of IoT requests from IoT devices is used to perform analysis. The outcomes are then kept in the knowledge base so the planner component can use them later.

- Plan: This part serves as the core decision-maker for the resource provisioning procedure. When the analyzer predicts the system status, this component goes to the database and decides when and how to allocate fog nodes to the IoT requests. Based on the anticipated outcomes of the LSTM model, the load predictor will route the Internet of Things requests to the appropriate fog nodes.

- Execute: It distributes and reallocates fog nodes according to the planner component's choices. This is done by sending the decisions made to the fog nodes manager and making the required adjustments. It adds or removes fog nodes based on the analyzer component's predictions. Lastly, the database stores the effective scaling choices so that the auto-scaling component can utilize them.

The flowchart of the proposed approach is shown in Figure 5.6. The very first step is to gather IoT requests and accordingly check for the resources from the resource manager. To predict resource utilization, the LSTM prediction model has been applied. next, the upper and lower thresholds have been set to detect the over-load and under-load status of the host. Accordingly, scale-in or out decisions have been performed by migrating the

appropriate VMs.

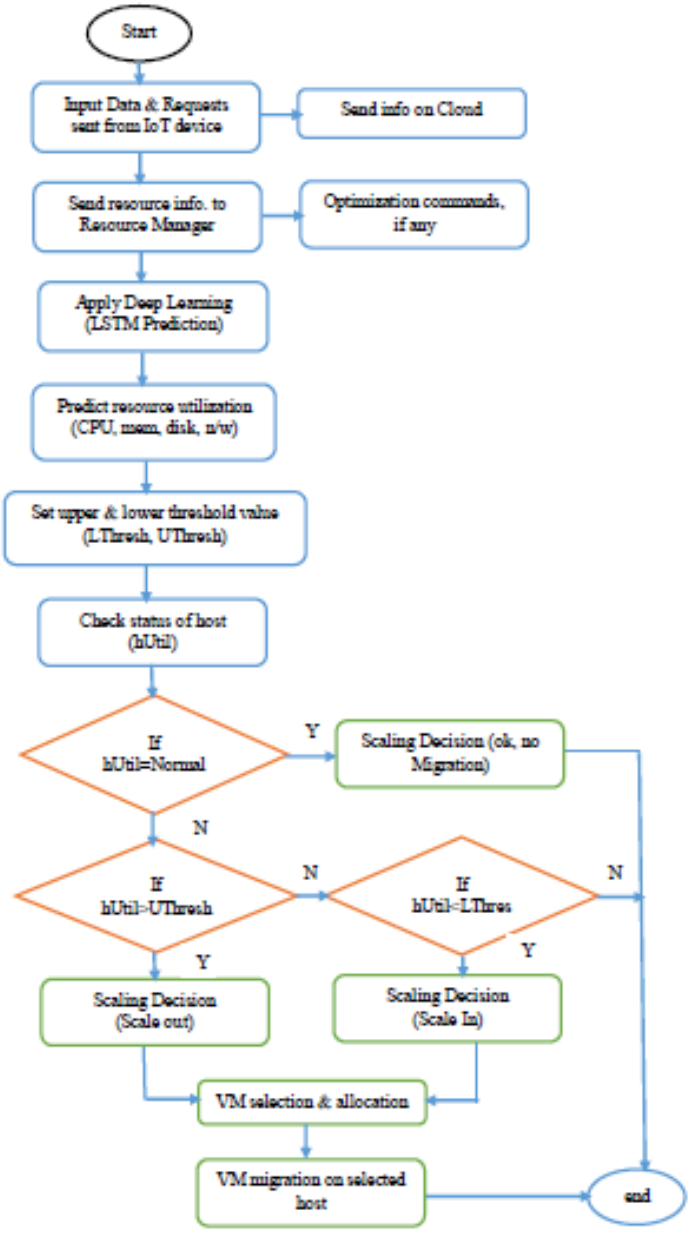


Figure 5.6: The flowchart of proposed model

5.4.1 FEAST: Prediction using LSTM Model

The RNN model's ambiguity has been suggested to be resolved via Long Short-Term Memory (LSTM). The Back-propagation Through Time (BTT) algorithm is used to train the network, giving RNN the capacity to learn arbitrary length relationships [126]. However, because of the gradient's vanishing and expanding nature, applying this approach to the collection of data for long-term reliance frequently fails. For tasks whose input/output

sequences include temporal contingencies across extended periods of time, the training step consequently becomes challenging. The LSTM uses a set of gates—input, forget, and output—to solve the gradient vanishing and exploding problem. These gates keep an eye on who has access to the memory cells and guard against irrelevant inputs disturbing them. The model's capacity to retain information for longer periods of time and forget irrelevant values is enhanced by these gates. These gates create a sigmoid layer, which has an output value ranging from 0 to 1 [54].

The final stage is where the components of the cell state will be sent to output and are defined at the output gate. Additionally, the sigmoid and the tanh are the two layers of this gate. The portions of the cell state that should be sent for output are determined by the sigmoid layer. After that, the tanh layer is applied to the cell state to provide normalized values between -1 and 1. Lastly, the sigmoid layer's output is multiplied by the values that were obtained. The RNN maintains and overrides the state of the cells; by applying these processes, the LSTM adds and removes information from the cell state. The LSTM and RNN have ruined many applications' exceptional performance. However, while processing long-dependence data, LSTM models outperform RNN.

5.4.2 FEAST: Evaluation Metrics

Numerous servers are connected in a Cloud Data Centre (CDC) and offer cloud consumers a range of cloud services. Within a data center, these servers use their resources in different ways. In this research work, the different resources considered to calculate resource utilization are CPU, memory, disk and network/bandwidth usage. The very first step to calculating CPU utilization is to initialize the value of the number of jobs (J) and number of VMs as 1. Then, for each Kth job running on the Nth node on Mth VM, CPU and memory utilization are calculated as shown in step 2. Further, the total CPU utilization is predicted by the formula given below:

$$totalCU_N = \frac{C_i * IC}{CR} \tag{5.1}$$

The clock cycles per instruction are shown by C_i in Equation 5.1, the instruction count is indicated by IC, and the clock rate is indicated by CR. The average CPU use is computed as follows by accounting for this total CPU utilization:

$$avgCU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} CU_x}{totalCU_N} * 100 \tag{5.2}$$

In the above equation 5.2, $avgCU_N$ is calculated at any given time for the Nth node. considering V_N as the number of VMs running on the Nth node and J_N as the number of jobs assigned to V_N VMs. CU_x is the CPU utilization of K jobs running on M VMs on the Nth node. Similarly, the average memory utilization is calculated by the formula:

$$avgMU_N = \frac{\sum_{M=1}^{V_N} \sum_{K=1}^{J_N} MU_x}{totalMU_N} * 100 \tag{5.3}$$

Equation 5.3, where V_N is the number of virtual machines (VMs) running on the Nth node and J_N is the number of jobs allocated to V_N VMs, can be used to determine the average memory utilization $avgMU_N$ for the Nth node at any given time. On the Nth node, $totalMU_N$ represents the total memory utilization for the node, and MU_x represents the memory utilization of K tasks running on M VMs.

$$TD_i = D_{rb} + D_{wb} + D_{rr} + D_{wr} \tag{5.4}$$

$$avgDU_N = \frac{TD_i}{AU * SS} * 100 \tag{5.5}$$

Also, at a given time, for the Nth node, the average disk utilization $avgDU_N$ can be given in Equation 5.5, where TD is the total disk used as per the formula given in Equation 5.4 and AU and SS are the allocated units and storage size of the disk respectively. For computing total disk used TD, the parameters required to be measured are disk read bytes D_{rb}, disk write bytes D_{wb}, Disk read requests D_{rr} and disk write requests D_{wr}.

$$avgNU_N = \frac{NT_{ib} + NT_{ob}}{NT_{wp}} * 100 \tag{5.6}$$

Similarly, to compute network/bandwidth utilization, the formula is given in Equation 5.6, where avg_NU denotes the average network/bandwidth utilization, NT_{ib} and NT_{ob} are the total network in and out bytes respectively and NT_{wp} denotes the total network window time period.

First, it has been considered that an IoT application consists of a collection of IoT requests that fog nodes in the fog tier process. r is the total number of IoT requests, and u is the number of users, according to Equation 5.7. The maximum time is included in the RQ_{ur} as the deadline for completing the rth request belonging to the uth user, i.e.,dl_{ur}, the required CPU, memory, disk and network utilization that is denoted by CU_{ur}, MU_{ur}, DU_{ur} and NU_{ur} respectively.

$$RQ^{ur} = dl_{ur}, CU_{ur}, MU_{ur}, DU_{ur}, NU_{ur} \tag{5.7}$$

Additionally, the admission control component verifies if an IoT request is for a transfer to the cloud or fog layer after receiving it from an IoT device. An IoT request is processed at the fog tier if its deadline is under the delay threshold; if not, it is forwarded to the cloud tier. The fog layer comprises many fog nodes fn, which are indicated in Equation 5.8, depend on resource estimation potential CPU, Memory, Disk and Network utilization:

$$fn = CU_fn, MU_fn, DU_fn, NU_fn \tag{5.8}$$

Assuming that TC denotes the overall total cost to carry through the IoT user's requests and includes the cost of computation $Cost_cmp$ and cost of communication $Cost_cmm$, as is expressed by Equation 5.9. The computation cost is the resources assigned to handle IoT users' requests. The inter-fog communication estimates the communication cost.

$$TC = Cost_{cmp} + Cost_{cmm} \tag{5.9}$$

The following Equation 5.10 calculates the communication cost between the fog nodes, either inter-fog or fog-fog communication by including the CPU, memory, disk and network costs along with the communication cost of a fog node $Cost_{fn}$, length of a request len_r and the transmission rate of an IoT request from fog nodes tr_{fn}, as follows:

$$Cost_{cmm} = \sum_{fn}^{ur} CU_i + \sum_{fn}^{ur} MU_i + \sum_{fn}^{ur} DU_i + \sum_{fn}^{ur} NU_i + Cost_{fn} + len_r + tr_{fn} \tag{5.10}$$

Now, the Computation cost $Cost_{cmp}$ is calculated by the total resource cost and the penalty cost, where resource cost depends on the different resource capabilities. The penalty cost is calculated based on the IoT request's delay violation as served by the fog node. When a fog node fails to meet the delay threshold specified in the QoS requirements for the IoT request, there is a delay violation. Equation 5.11 illustrates that a delay violation happens when the delay time of the IoT request exceeds the delay threshold.

$$dv_r = \frac{1}{r}\sum_{i=1}^{r} {fn_r}^t * 100 \tag{5.11}$$

5.4.3 FEAST: Auto-scaling

In this fog-based case study, the proposed auto-scaling approach has been deployed. In the proposed auto-scaling, firstly, the host over-utilization (Algorithm 4.1) and under-utilization (Algorithm 4.2) algorithms are devised to detect the host's status. If the CPU utilization of the host is above the upper threshold limit (80%), then appropriate VMs will be scaled out to maintain the load. Similarly, if the utilization is below a lower threshold (20%), then more VMs will be appended on that host. It is also recommended that appropriate VMs be chosen for migration to avoid other consequences and overhead. In the proposed approach, the Minimum Size Maximum Utilization (MSMU) policy has been applied in which a VM having minimum size but maximum resource utilization is selected (Algorithm 4.3). When the VM is chosen with higher resource utilization than other VMs on the host, the risk of over-utilization and under-utilization is reduced.

5.5 FEAST: Experimental Setup

The iFogSim toolbox, an extension of CloudSim, has been used to create the experiments in this section. It models and simulates IoT services and heterogeneous fog computing infrastructures. The iFogSim toolkit enables the user to manage Internet of Things (IoT) services, describe fog nodes, and assess performance metrics pertaining to fog environments, such as energy consumption, latency, cost, and utilization. It consists of several classes: AppModule, AppEdge, and Tuple to model IoT services; FogDevice (also known as Fog node), Sensor, and Actuator to model fog infrastructure. In this study, an Intel Core i5 CPU, 250 GB disk, 4 GB RAM, and Windows 10–64bit were used for the experimental investigation, which was conducted using the iFogSim toolkit.

Four heterogeneous virtual machines (VMs) are created in this research case study in order to analyze data concurrently and validate how the suggested approach operates in a cloud context. Practical observations show that the procedure being carried out determines the distinct time periods at which the CPU demand varies. In order to evaluate all potential consequences of load prediction with over-utilization or under-utilization elements of CPU consumption, which might either directly or indirectly influence the performance aspects of the suggested approach, an attempt has been made in this proposed work.

5.5.1 FEAST: Result Evaluation

In this part, the experimental study of the suggested approach is emphasized.

a) **Average CPU utilization**: The amount of work that a virtual machine's CPU on a specific host is able to handle is known as CPU utilization. Figure 5.7 displays the anticipated and actual CPU consumption for several virtual machines. For the given approaches EA, CE, AFED-EF and proposed, it comes out to be (69.34%, 57.5%, 60.9% and 55.84%) respectively [124], [123], [121]. The findings show that the proposed approach gives better utilization than other approaches.

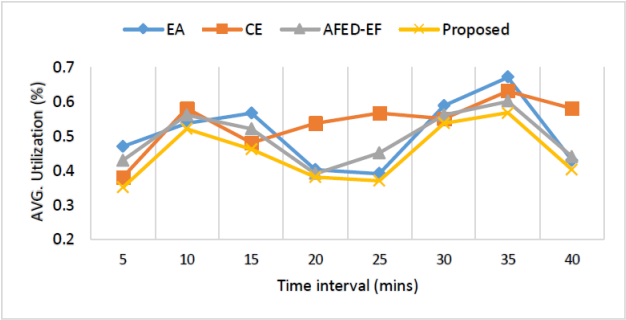


Figure 5.7: Average CPU utilization

b) **Response Time**: As was previously said, an application's response time needs to be prompt and must match 1 in accordance with the SLA. Figure 5.8 presents the response time comparison between the suggested and current methods. For instance, at 15-minute time intervals, the EA, CE, and AFED-EF approaches give a response time of (4.45%, 5.2%, and 4.16%) respectively, whereas the proposed approach has a response time of 3.67%. Overall, the response time has been reduced by 2.4% approximately. The data clearly show that the suggested auto-scaling mechanism responds quickly—within five minutes of interval.

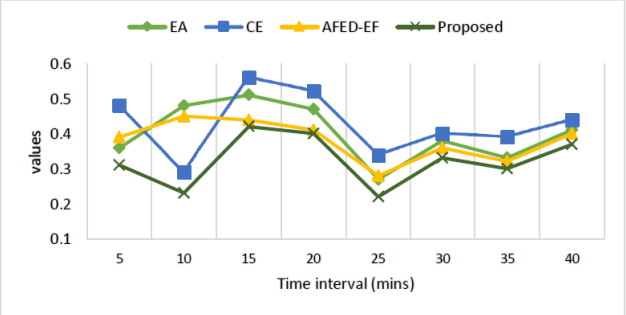


Figure 5.8: Response Time

c) **Average Total Cost**: The average total cost of various approaches at each interval is shown in Figure 5.9. Among all the approaches, the CE approach has a higher cost (67.14%) than others, EA (54.8%), AFED-EF (52.9%), and proposed (45.4%). According to the simulation results, the suggested solution has lower resource usage costs by 5.3% compared to alternative methods. The results obtained further demonstrate the scalability and ability of the suggested approach to schedule large jobs in a heterogeneous environment while incurring the minimum execution costs. This is made possible by the suggested solution's ability to employ the LSTM technique to identify the appropriate quantity of fog resources based on the kind of IoT request received from IoT devices. Since the suggested method uses just the necessary fog resources, the remainder is turned off. As a result, the overall expense of carrying out the IoT requests drops.

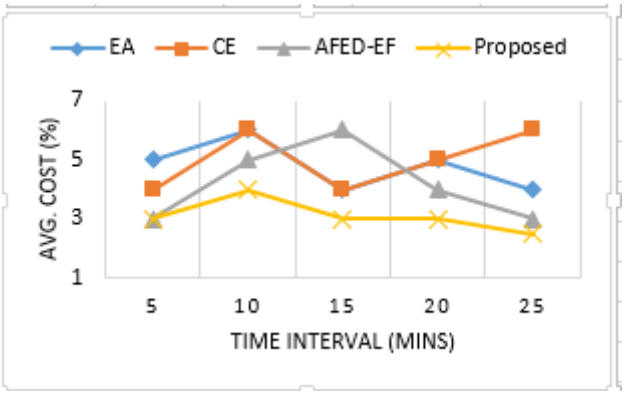


Figure 5.9: Average Total Cost

d) **SLA Violations**: One of the most important factors in certifying the optimum accessibility of different services and apps to Cloud users is SLA compliance. Tenants are required to pay a penalty in the event of any SLA violation. Depending on the major parameter response time of an application, the SLA violation is investigated. As can be seen in Figure 5.10, the suggested method has been tested with approximately 1% SLA violation, which leads to more effective resource accessibility and improved QoS.

e) **Delay Violation**: The delay violation criteria of recent EA, CE, and AFED-EF approaches are shown in Figure 5.11 at various time intervals. Delay violations occur when the amount of resources is less than what the requests need. While having fewer resources can lead to under-provisioning and the inability to respond to some incoming requests, it is also desirable in terms of energy consumption and CPU utilization. Delay violations and user attrition may result from this. Consequently,

the suggested methodology considerably decreases the rate of delay violation by 3.2% in contrast to the current techniques.

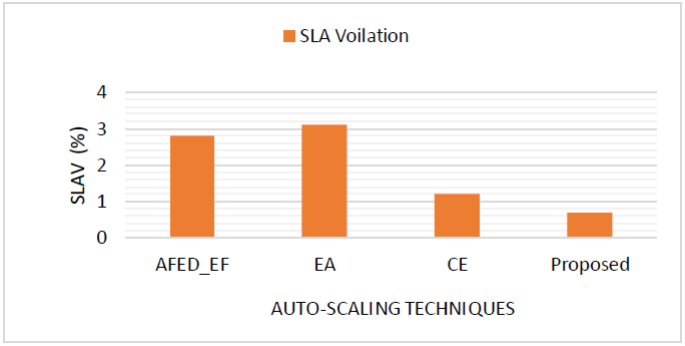


Figure 5.10: SLA Violations

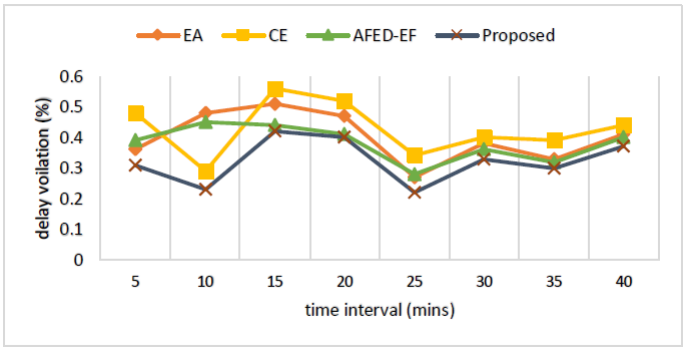


Figure 5.11: Delay Violation

When applied to real-world workloads, prediction models generally yield more accurate prediction results than when applied to workloads with large swings. Ultimately, out of all the prediction methods, the suggested FEAST model yields the prediction results with the lowest error rate. The trial results mentioned above also make it evident that the suggested technique is a dynamic and effective method that gives application providers and end users the necessary level of quality of service.

5.6 Conclusion

In order to prevent problems with fog resource overuse and underuse, a dynamic resource auto-scaling technique has been presented in this research for unpredictable IoT queries. A deep learning method, LSTM, has been developed to address the issue of variable load. The suggested three-tier fog architecture for auto-scaling shows how cloud, fog, and IoT resources interact at the same time. In comparison to the current models,

the testing results demonstrated the superiority of the suggested technique FEAST in terms of improved cost by 5.31%, reduced response time by 2.4%, and delay violation by 3.2%. Furthermore, the suggested methodology has been evaluated with our proposed auto-scaling using host load prediction and VM migration. For this fog-based model, proposed auto-scaling enhanced resource allocation, improved accuracy and reduced service level agreement violations, resulting in efficient and dynamic resource attainability with enhanced quality of service.

The next chapter summarizes the research effort presented in the thesis by drawing conclusions from all the chapters and presents future directions.

Chapter 6

Conclusion and Future Scope

As cloud computing service providers work to successfully organize, arrange, and schedule the resources for executing an IoT application, efficient and dynamic resource scaling continues to be a critical challenge. Because different application types require varying amounts of resources at different times, the problem becomes even more complex. The current cloud systems must be sufficiently effective to automatically scale the resources for IoT-based cloud applications following the anticipated resource utilization requirements.

This thesis addresses the need for prediction-based auto-scaling of resources for IoT-based Cloud applications. The work proposes an Ensemble Time-Series Approach for Load Prediction (ETSA-LP) for predicting resource usage, such as CPU and memory. Further, it implements an auto-scaling approach based on host load prediction using VM migration.

Finally, this chapter summarizes the research effort presented in the thesis by illustrating the steps taken to achieve the study's goals in terms of dynamic load prediction, VM migration and resource scaling. This chapter also suggests some avenues for further investigation.

6.1 Conclusion

This research work aims to develop and implement an effective auto-scaling strategy using load prediction for IoT-based cloud applications. Auto-scaling based on host load detection, virtual machine migration, and the suggested ensemble approach for prediction have addressed this goal. Additionally, the suggested auto-scaling approach has been used to improve and evaluate the performance utilizing the fog-based FEAST approach. The following is a summary of the main conclusions drawn from this study:

- A detailed survey has been conducted to explore various IoT-based Cloud applications, such as healthcare, smart homes, smart cities, automotive, smart energy, *etc.*

- A thorough survey has been conducted for various auto-scaling techniques, including threshold-based, reinforcement learning, control theory, time-series forecasting techniques, *etc.* In addition, essential QoS metrics have been evaluated to implement a viable auto-scaling approach.

- Further, a comprehensive investigation has been conducted to study the existing load prediction and VM migration approaches, including their merits and demerits. Finally, based on the studied literature, the challenges related to auto-scaling in IoT-based Cloud applications are discussed.

- An Ensemble-based framework ETSA-LP has been designed and implemented, integrating various time-series analysis techniques for predicting CPU and memory utilization. The overall accuracy is improved by approximately 3% and execution time is reduced by 12.2% with a minimum error value (0.39%).

- Based on the predicted load, an auto-scaling approach has been developed using VM migration for the Cloud environment. Different algorithms have been devised to detect over-utilized hosts, under-utilized hosts, and VM migration.

- The suggested auto-scaling approach has been validated by experimentation on a real-time Google cluster dataset. The simulation results have improved accuracy by 5.7% and response time by 3.7%. It also consistently reduces error rate by 2.7%, no. of migrations by 2% and scaling overhead by 2.6% compared to existing auto-scaling approaches.

- The proposed load prediction and auto-scaling approaches have been validated on GCP and AWS Cloud. The load variations, initiation and termination corresponding to different heterogeneous VMs have been monitored and analyzed at different

time intervals.

- Finally, a Fog-Enabled Auto-Scaling Technique (FEAST) has been proposed and deployed as a case study to predict resource usage concerning CPU, memory, disk, and network utilization using the LSTM model. Compared to the current models, the testing results demonstrated the superiority of the suggested technique FEAST in terms of improved cost by 5.31%, reduced response time by 2.4%, and delay violation by 3.2%.
- Last but not least, the proposed auto-scaling approach will help in dynamic load prediction, avoiding over and under-provisioning problems and simultaneously satisfying SLA and QoS requirements in the context of IoT-based Cloud applications.

6.2 Future Scope

Through its significant contributions, this thesis advances our understanding of load prediction and auto-scaling for IoT-based cloud applications. This thesis has identified new Cloud Computing research areas that need more investigation. Below are some ideas for future study directions:

- The proposed prediction approach can be extended to predict anomalies and peak resource usage periods to improve the scheduling, load balancing, and resource scaling. Also, the auto-scaling approach can be further tested on IoT-based cloud applications with consideration of power consumption so that energy efficiency can be improved in a cloud environment.
- The proposed prediction technique can be used to predict the security threats to upgrade the performance of various applications such as risk assessment, project management, *etc.* Most of the existing load prediction schemes ignore network resources and bandwidth, leaving an opportunity to improve the adequate bandwidth of parallel clustered applications.
- The suggested research work can also come up with desired benefits to IoT-based Cloud applications in terms of managing load caused by various sensors installed and help in lowering their operational cost.
- Recently, a new data-driven paradigm, termed digital twin (DT), has emerged and received increasing attention in Cloud. Therefore, the proposed approach can be simulated using the real-time DT, increasing reliability, and maintenance and extending its service life.

References

[1] Robert Dukaric and Matjaz B Juric. Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems*, 29(5):1196–1210, 2013.

[2] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, 56:684–700, 2016.

[3] Mahmoud M Badawy, Zainab H Ali, and Hesham A Ali. Qos provisioning framework for service-oriented internet of things (iot). *Cluster Computing*, 23:575–591, 2020.

[4] Anisha Gupta, Rivana Christie, and R Manjula. Scalability in internet of things: features, techniques and research challenges. *Int. J. Comput. Intell. Res*, 13(7):1617–1627, 2017.

[5] Omer F Rana and Kate Stout. What is scalability in multi-agent systems? In *Proceedings of the fourth international conference on Autonomous agents*, pages 56–63, 2000.

[6] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12:2012, 2012.

[7] Mohammad S Aslanpour, Sukhpal Singh Gill, and Adel N Toosi. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things*, 12:100273, 2020.

[8] Hanieh Alipour, Yan Liu, and Abdelwahab Hamou-Lhadj. Analyzing auto-scaling issues in cloud environments. In *CASCON*, volume 14, pages 75–89, 2014.

[9] Parminder Singh, Pooja Gupta, Kiran Jyoti, and Anand Nayyar. Research on auto-scaling of web applications in cloud: survey, trends and future directions. *Scalable Computing: Practice and Experience*, 20(2):399–432, 2019.

[10] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 500–507. IEEE, 2011.

[11] Paulo Pereira, Jean Araujo, and Paulo Maciel. A hybrid mechanism of horizontal auto-scaling based on thresholds and time series. In *2019 IEEE international conference on systems, man and cybernetics (SMC)*, pages 2065–2070. IEEE, 2019.

[12] Anju Bala and Inderveer Chana. Prediction-based proactive load balancing ap-

proach through vm migration. *Engineering with Computers*, 32:581–592, 2016.

[13] Fei Zhang, Guangming Liu, Xiaoming Fu, and Ramin Yahyapour. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys & Tutorials*, 20(2):1206–1243, 2018.

[14] Mohammad Masdari and Hemn Khezri. Efficient vm migrations using forecasting techniques in cloud computing: a comprehensive review. *Cluster Computing*, 23(4):2629–2658, 2020.

[15] Rishabh and TP Sharma. Device classification based data encryption for internet of things. *International Journal of High Performance Computing and Networking*, 16(1):36–42, 2020.

[16] Mohammed Ibrahim, Susan Gauch, Omar Salman, and Mohammed Alqahtani. An automated method to enrich consumer health vocabularies using glove word embeddings and an auxiliary lexical resource. *PeerJ Computer Science*, 7:e668, 2021.

[17] Shruti Kaushik, Abhinav Choudhury, Pankaj Kumar Sheron, Nataraj Dasgupta, Sayee Natarajan, Larry A Pickett, and Varun Dutt. Ai in healthcare: time-series forecasting using statistical, neural, and ensemble architectures. *Frontiers in big data*, 3:4, 2020.

[18] Chanapha Butpheng, Kuo-Hui Yeh, and Hu Xiong. Security and privacy in iot-cloud-based e-health systems—a comprehensive review. *Symmetry*, 12(7):1191, 2020.

[19] Navod Neranjan Thilakarathne and WD Madhuka Priyashan. An overview of security and privacy in smart cities. *IoT and IoE Driven Smart Cities*, pages 21–44, 2021.

[20] Mahdi Kasmi, Faouzi Bahloul, and Haykel Tkitek. Smart home based on internet of things and cloud computing. In *2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, pages 82–86. IEEE, 2016.

[21] Jie Yang, Jinbao He, and Xiongwei Wang. Design of intelligent parking system based on internet of things and cloud platform. *International Journal of Grid & High Performance Computing*, 15(2), 2023.

[22] Mohammed Lawal Ahmed, Rahat Iqbal, Charalampos Karyotis, Vasile Palade, and Saad Ali Amin. Predicting the public adoption of connected and autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):1680–1688, 2021.

[23] Irene Yuan-Ying Hsu, Michał Wódczak, Robert G White, Tao Zhang, and T Russell Hsing. Challenges, approaches, and solutions in intelligent transportation sys-

tems. In *2010 second international conference on ubiquitous and future networks (ICUFN)*, pages 366–371. IEEE, 2010.

[24] Richa, TP Sharma, and Ajay Kumar Sharma. Heterogeneous-internet of vehicles (het-iov) in twenty-first century: A comprehensive study. *Handbook of computer networks and cyber security: principles and paradigms*, pages 555–584, 2020.

[25] Sumedha Sharma, Yan Xu, Ashu Verma, and Bijaya Ketan Panigrahi. Time-coordinated multienergy management of smart buildings under uncertainties. *IEEE Transactions on Industrial Informatics*, 15(8):4788–4798, 2019.

[26] Simar Preet Singh, Rajesh Kumar, Anju Sharma, Jemal H Abawajy, and Ravneet Kaur. Energy efficient load balancing hybrid priority assigned laxity algorithm in fog computing. *Cluster Computing*, pages 1–18, 2022.

[27] Minal Patel, Sanjay Chaudhary, and Sanjay Garg. Performance modeling and optimization of live migration of virtual machines in cloud infrastructure. *Research advances in cloud computing*, pages 327–350, 2017.

[28] Adel Nadjaran Toosi, Rodrigo N Calheiros, Ruppa K Thulasiram, and Rajkumar Buyya. Resource provisioning policies to increase iaas provider's profit in a federated cloud environment. In *2011 IEEE International Conference on High Performance Computing and Communications*, pages 279–287. IEEE, 2011.

[29] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12:559–592, 2014.

[30] Pranali Gajjar and Brona Shah. Survey on different auto scaling techniques in cloud computing environment. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(12):2278–1021, 2015.

[31] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Application management in fog computing environments: A taxonomy, review and future directions. *ACM Computing Surveys (CSUR)*, 53(4):1–43, 2020.

[32] Anton Beloglazov and Rajkumar Buyya. Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds. *Concurrency and Computation: Practice and Experience*, 27(5):1310–1333, 2015.

[33] Anshuman Biswas, Shikharesh Majumdar, Biswajit Nandy, and Ali El-Haraki. A hybrid auto-scaling technique for clouds processing applications with service level agreements. *Journal of Cloud Computing*, 6(1):1–22, 2017.

[34] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

[35] Gaopan Huang, Songyun Wang, Mingming Zhang, Yefei Li, Zhuzhong Qian, Yuan

Chen, and Sheng Zhang. Auto scaling virtual machines for web applications with queueing theory. In *2016 3rd International conference on systems and informatics (ICSAI)*, pages 433–438. IEEE, 2016.

[36] Javad Dogani, Reza Namvar, and Farshad Khunjush. Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey. *Computer Communications*, 2023.

[37] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[38] Manasi Patil. *Enhancing Static Auto-scaling Approach to Mitigate Resource Over-Provisioning in Cloud Computing*. PhD thesis, Dublin, National College of Ireland, 2019.

[39] Mohammad Sadegh Aslanpour, Mostafa Ghobaei-Arani, and Adel Nadjaran Toosi. Auto-scaling web applications in clouds: A cost-aware approach. *Journal of Network and Computer Applications*, 95:26–41, 2017.

[40] Masum Z Hasan, Edgar Magana, Alexander Clemm, Lew Tucker, and Sree Lakshmi D Gudreddi. Integrated and autonomic cloud resource scaling. In *2012 IEEE network operations and management symposium*, pages 1327–1334. IEEE, 2012.

[41] Parminder Singh, Avinash Kaur, Pooja Gupta, Sukhpal Singh Gill, and Kiran Jyoti. Rhas: robust hybrid auto-scaling for web applications in cloud computing. *Cluster Computing*, 24(2):717–737, 2021.

[42] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[43] Monireh Fallah, Mostafa Ghobaei Arani, and Mehrdad Maeen. Nasla: Novel auto scaling approach based on learning automata for web application in cloud computing environment. *International Journal of Computer Applications*, 113(2), 2015.

[44] Trieu C Chieu, Ajay Mohindra, Alexei A Karve, and Alla Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *2009 IEEE International Conference on e-Business Engineering*, pages 281–286. IEEE, 2009.

[45] Jonathan Kupferman, Jeff Silverman, Patricio Jara, and Jeff Browne. Scaling into the cloud. *CS270-advanced operating systems*, 2009.

[46] Harold C Lim, Shivnath Babu, Jeffrey S Chase, and Sujay S Parekh. Automated control in cloud computing: challenges and opportunities. In *Proceedings of the 1st workshop on Automated control for data centers and clouds*, pages 13–18, 2009.

[47] Nishant Gupta, Naman Ahuja, Shikhar Malhotra, Anju Bala, and Gurleen Kaur. Intelligent heart disease prediction in cloud environment through ensembling. *Ex-*

pert Systems, 34(3):e12207, 2017.

[48] Yongyu Chen and John Carlo Vincent Cattaneo. Auto-scaling for allocation of cloud service resources in application deployments, June 30 2020. US Patent 10,698,735.

[49] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[50] Weichao Ding, Fei Luo, Liangxiu Han, Chunhua Gu, Haifeng Lu, and Joel Fuentes. Adaptive virtual machine consolidation framework based on performance-to-power ratio in cloud data centers. *Future Generation Computer Systems*, 111:254–270, 2020.

[51] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pages 137–146, 2009.

[52] Thomas Cooper. Proactive scaling of distributed stream processing work flows using workload modelling: doctoral symposium. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 410–413, 2016.

[53] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and computation: practice and experience*, 25(12):1656–1674, 2013.

[54] Jiang Zhong, Saisai Duan, and Qing Li. Auto-scaling cloud resources using lstm and reinforcement learning to guarantee service-level agreements and reduce resource costs. In *Journal of Physics: Conference Series*, volume 1237, page 022033. IOP Publishing, 2019.

[55] Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE, 2006.

[56] Jordi Vilaplana, Francesc Solsona, Ivan Teixidó, Jordi Mateo, Francesc Abella, and Josep Rius. A queuing theory model for cloud computing. *The Journal of Supercomputing*, 69:492–507, 2014.

[57] Jingwan Tong, Mingchang Wei, Maolin Pan, and Yang Yu. A holistic auto-scaling algorithm for multi-service applications based on balanced queuing network. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 531–540. IEEE, 2021.

[58] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. A reliable and cost-

efficient auto-scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications*, 65:167–180, 2016.

[59] T Vondra and J Šedivỳ. Cloud autoscaling simulation based on queueing network model. *Simulation Modelling Practice and Theory*, 70:83–100, 2017.

[60] Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Fourth International Conference on Autonomic Computing (ICAC'07)*, pages 27–27. IEEE, 2007.

[61] Iure Fé, Rubens Matos, Jamilson Dantas, Carlos Melo, Tuan Anh Nguyen, Dugki Min, Eunmi Choi, Francisco Airton Silva, and Paulo Romero Martins Maciel. Performance-cost trade-off in auto-scaling mechanisms for cloud computing. *Sensors*, 22(3):1221, 2022.

[62] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *2013 13th IEEE/ACM international symposium on cluster, Cloud, and Grid Computing*, pages 58–65. IEEE, 2013.

[63] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. Cloud resource auto-scaling system based on hidden markov model (hmm). In *2014 IEEE International Conference on Semantic Computing*, pages 124–127. IEEE, 2014.

[64] Sang-Min Park and Marty Humphrey. Self-tuning virtual machines for predictable escience. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 356–363. IEEE, 2009.

[65] Pradeep Padala, Kai-Yuan Hou, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 13–26, 2009.

[66] Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael I Jordan, and David A Patterson. Statistical machine learning makes automatic control practical for internet datacenters. *HotCloud*, 9:12–12, 2009.

[67] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, pages 117–126, 2009.

[68] Faiyaz Doctor and Rahat Iqbal. An intelligent framework for monitoring student performance using fuzzy rule-based linguistic summarisation. In *2012 IEEE International Conference on Fuzzy Systems*, pages 1–8. IEEE, 2012.

[69] Jing Xu, Ming Zhao, Jose Fortes, Robert Carpenter, and Mazin Yousif. On the use of fuzzy modeling in virtualized data center management. In *Fourth International*

Conference on Autonomic Computing (ICAC'07), pages 25–25. IEEE, 2007.

[70] Lixi Wang, Jing Xu, Ming Zhao, Yicheng Tu, and Jose AB Fortes. Fuzzy modeling based resource management for virtualized database systems. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 32–42. IEEE, 2011.

[71] Palden Lama and Xiaobo Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 151–160. IEEE, 2010.

[72] Negin Najafizadegan, Eslam Nazemi, and Vahid Khajehvand. An autonomous model for self-optimizing virtual machine selection by learning automata in cloud environment. *Software: Practice and Experience*, 51(6):1352–1386, 2021.

[73] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management*, pages 9–16. Ieee, 2010.

[74] Ishana Attri, Lalit Kumar Awasthi, Teek Parval Sharma, and Priyanka Rathee. A review of deep learning techniques used in agriculture. *Ecological Informatics*, page 102217, 2023.

[75] R Prashanth, Sumantra Dutta Roy, Pravat K Mandal, and Shantanu Ghosh. Automatic classification and prediction models for early parkinson's disease diagnosis from spect imaging. *Expert Systems with Applications*, 41(7):3333–3342, 2014.

[76] Paulo Roberto Pereira da SILVA. A hybrid strategy for auto-scaling of vms: an approach based on time series and thresholds. Master's thesis, Universidade Federal de Pernambuco, 2019.

[77] Jitendra Kumar and Ashutosh Kumar Singh. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41–52, 2018.

[78] Sukhpal Singh Gill, Peter Garraghan, Vlado Stankovski, Giuliano Casale, Ruppa K Thulasiram, Soumya K Ghosh, Kotagiri Ramamohanarao, and Rajkumar Buyya. Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge. *Journal of Systems and Software*, 155:104–129, 2019.

[79] Nasro Min-Allah, Muhammad Bilal Qureshi, Saleh Alrashed, and Omer F Rana. Cost efficient resource allocation for real-time tasks in embedded systems. *Sustainable Cities and Society*, 48:101523, 2019.

[80] Jing Chen and Yinglong Wang. A resource demand prediction method based on eemd in cloud computing. *Procedia Computer Science*, 131:116–123, 2018.

[81] Chunhong Liu, Chuanchang Liu, Yanlei Shang, Shiping Chen, Bo Cheng, and Junliang Chen. An adaptive prediction approach based on workload pattern discrimination in the cloud. *Journal of Network and Computer Applications*, 80:35–44, 2017.

[82] R Prashanth, Sumantra Dutta Roy, Pravat K Mandal, and Shantanu Ghosh. High-accuracy detection of early parkinson's disease through multimodal features and machine learning. *International journal of medical informatics*, 90:13–21, 2016.

[83] Valter Rogério Messias, Julio Cezar Estrella, Ricardo Ehlers, Marcos José Santana, Regina Carlucci Santana, and Stephan Reiff-Marganiec. Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. *Neural Computing and Applications*, 27:2383–2406, 2016.

[84] Jinhui Huang, Chunlin Li, and Jie Yu. Resource prediction based on double exponential smoothing in cloud computing. In *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 2056–2060. IEEE, 2012.

[85] Euclides C Pinto Neto, Gustavo Callou, and Fernando Aires. An algorithm to optimise the load distribution of fog environments. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1292–1297. IEEE, 2017.

[86] Ali Asghar Rahmanian, Mostafa Ghobaei-Arani, and Sajjad Tofighy. A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment. *Future Generation Computer Systems*, 79:54–71, 2018.

[87] Wei Fang, ZhiHui Lu, Jie Wu, and ZhenYin Cao. Rpps: A novel resource prediction and provisioning scheme in cloud data center. In *2012 IEEE Ninth International Conference on Services Computing*, pages 609–616. IEEE, 2012.

[88] Qingchen Zhang, Laurence T Yang, Zheng Yan, Zhikui Chen, and Peng Li. An efficient deep learning model to predict cloud workload for industry informatics. *IEEE transactions on industrial informatics*, 14(7):3170–3178, 2018.

[89] Javad Dogani, Farshad Khunjush, and Mehdi Seydali. Host load prediction in cloud computing with discrete wavelet transformation (dwt) and bidirectional gated recurrent unit (bigru) network. *Computer Communications*, 198:157–174, 2023.

[90] Liang Hu, Jia Zhao, Gaochao Xu, Yan Ding, and Jianfeng Chu. Hmdc: Live virtual machine migration based on hybrid memory copy and delta compression. *Appl. Math*, 7(2L):639–646, 2013.

[91] Shashank Sahni and Vasudeva Varma. A hybrid approach to live migration of virtual machines. In *2012 IEEE international conference on cloud computing in*

emerging markets (CCEM), pages 1–5. IEEE, 2012.

[92] Jihun Kim, Dongju Chae, Jangwoo Kim, and Jong Kim. Guide-copy: fast and silent migration of virtual machine for datacenters. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2013.

[93] Umesh Deshpande, Danny Chan, Ten-Young Guh, James Edouard, Kartik Gopalan, and Nilton Bila. Agile live migration of virtual machines. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1061–1070. IEEE, 2016.

[94] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, Hanhua Chen, and Xiaodong Pan. Mecom: Live migration of virtual machines by adaptively compressing memory pages. *Future Generation Computer Systems*, 38:23–35, 2014.

[95] Aidan Shribman and Benoit Hudzia. Pre-copy and post-copy vm live migration for memory intensive applications. In *Euro-Par 2012: Parallel Processing Workshops: BDMC, CGWS, HeteroPar, HiBB, OMHI, Paraphrase, PROPER, Resilience, UCHPC, VHPC, Rhodes Islands, Greece, August 27-31, 2012. Revised Selected Papers 18*, pages 539–547. Springer, 2013.

[96] Bane Raman Raghunath and B Annappa. Virtual machine migration triggering using application workload prediction. *Procedia Computer Science*, 54:167–176, 2015.

[97] Umesh Deshpande and Kate Keahey. Traffic-sensitive live migration of virtual machines. *Future Generation Computer Systems*, 72:118–128, 2017.

[98] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing*, pages 171–182, 2011.

[99] Zhong Wang, Daniel Sun, Guangtao Xue, Shiyou Qian, Guoqiang Li, and Minglu Li. Ada-things: An adaptive virtual machine monitoring and migration strategy for internet of things applications. *Journal of Parallel and Distributed Computing*, 132:164–176, 2019.

[100] Martin Duggan, Rachael Shaw, Jim Duggan, Enda Howley, and Enda Barrett. A multitime-steps-ahead prediction approach for scheduling live migration in cloud data centers. *Software: Practice and Experience*, 49(4):617–639, 2019.

[101] Saurabh Kumar Garg, Adel Nadjaran Toosi, Srinivasa K Gopalaiyengar, and Rajkumar Buyya. Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter. *Journal of Network and Computer Applications*, 45:108–120, 2014.

[102] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)*, 51(4):1–33, 2018.

[103] Emiliano Casalicchio. A study on performance measures for auto-scaling cpu-intensive containerized applications. *Cluster Computing*, 22(3):995–1006, 2019.

[104] Anju Sharma, Rohit Pandey, Simar P Singh, and Rajesh Kumar. A survey of load balancing and implementation of clustering-based approach for clouds. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, 14(3):669–677, 2021.

[105] R Raval and A Gonsai. Performance analysis and design of a mobile web services on cloud servers. *International Journal of Emerging Technology and Advanced Engineering*, 5(9):104–113, 2015.

[106] Samuel Adesoye Ajila and Akindele A Bankole. Using machine learning algorithms for cloud client prediction models in a web vm resource provisioning environment. *Transactions on Machine Learning and Artificial Intelligence*, 4(1):28, 2016.

[107] Jian Cao, Jiwen Fu, Minglu Li, and Jinjun Chen. Cpu load prediction for cloud environment based on a dynamic ensemble model. *Software: Practice and Experience*, 44(7):793–804, 2014.

[108] Gurleen Kaur, Anju Bala, and Inderveer Chana. An intelligent regressive ensemble approach for predicting resource usage in cloud computing. *Journal of Parallel and Distributed Computing*, 123:1–12, 2019.

[109] Ngoc-Tri Ngo, Anh-Duc Pham, Thi Thu Ha Truong, Ngoc-Son Truong, Nhat-To Huynh, and Tuan Minh Pham. An ensemble machine learning model for enhancing the prediction accuracy of energy consumption in buildings. *Arabian Journal for Science and Engineering*, pages 1–13, 2021.

[110] M Chandini, R Pushpalatha, and R Boraia. A brief study on prediction of load in cloud environment. *International Journal of Advanced Research in Computer and Communication Engineering.–2016.–5 (5).–pp*, pages 157–162, 2016.

[111] Zheyi Chen, Jia Hu, Geyong Min, Albert Y Zomaya, and Tarek El-Ghazawi. Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning. *IEEE Transactions on Parallel and Distributed Systems*, 31(4):923–934, 2019.

[112] Peyman Yazdanian and Saeed Sharifian. E2lg: a multiscale ensemble of lstm/gan deep learning architecture for multistep-ahead cloud workload prediction. *The Journal of Supercomputing*, pages 1–31, 2021.

[113] Rachael Shaw, Enda Howley, and Enda Barrett. An intelligent ensemble learning approach for energy efficient and interference aware dynamic virtual machine

consolidation. *Simulation Modelling Practice and Theory*, 102:101992, 2020.

[114] John Wilkes and Charles Reiss. Google Cluster data. `https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md`, 2011. [Online; accessed 19-Nov-2022].

[115] Anton Beloglazov. PlanetLab traces. `https://github.com/beloglazov/planetlab-workload-traces`, 2011. [Online; accessed 15-Nov-2022].

[116] Habte Lejebo Leka, Zhang Fengli, Ayantu Tesfaye Kenea, Negalign Wake Hundera, Tewodros Gizaw Tohye, and Abebe Tamrat Tegene. Pso-based ensemble meta-learning approach for cloud virtual machine resource usage prediction. *Symmetry*, 15(3):613, 2023.

[117] Amany Abdelsamea, Ali A El-Moursy, Elsayed E Hemayed, and Hesham Eldeeb. Virtual machine consolidation enhancement using hybrid regression algorithms. *Egyptian Informatics Journal*, 18(3):161–170, 2017.

[118] Liang Bao, Jin Yang, Zhengtong Zhang, Wenjing Liu, Junhao Chen, and Chase Wu. On accurate prediction of cloud workloads with adaptive pattern mining. *The Journal of Supercomputing*, 79(1):160–187, 2023.

[119] Shilpa Sunil and Sanjeev Patel. Energy-efficient virtual machine placement algorithm based on power usage. *Computing*, pages 1–25, 2023.

[120] Masoumeh Ayoubi, Mohammadreza Ramezanpour, and Reihaneh Khorsand. An autonomous iot service placement methodology in fog computing. *Software: Practice and Experience*, 51(5):1097–1120, 2021.

[121] Zhou Zhou, Mohammad Shojafar, Mamoun Alazab, Jemal Abawajy, and Fangmin Li. Afed-ef: An energy-efficient vm allocation algorithm for iot applications in a cloud data center. *IEEE Transactions on Green Communications and Networking*, 5(2):658–669, 2021.

[122] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–41, 2019.

[123] Masoumeh Etemadi, Mostafa Ghobaei-Arani, and Ali Shahidinejad. Resource provisioning for iot services in the fog computing environment: An autonomic approach. *Computer Communications*, 161:109–131, 2020.

[124] Masoumeh Etemadi, Mostafa Ghobaei-Arani, and Ali Shahidinejad. A cost-efficient auto-scaling mechanism for iot applications in fog computing environment: a deep learning-based approach. *Cluster Computing*, 24(4):3277–3292, 2021.

[125] Kun Zhang, Yu Zhou, Chaoyang Wang, Haizhuang Hong, Jing Chen, Qian Gao, and Mostafa Ghobaei-Arani. Towards an automatic deployment model of iot services in fog computing using an adaptive differential evolution algorithm. *Internet of*

Things, 24:100918, 2023.

[126] Nhat-Minh Dang-Quang and Myungsik Yoo. Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. *Applied Sciences*, 11(9):3835, 2021.