Dynamic Resource Provisioning for Container-based Virtualization Application using Hybrid Model Approach

Habtamu Shiferaw Adugna

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September, 2022

Adama, Ethiopia

Dynamic Resource Provisioning for Container-based Virtualization Application using Hybrid Model Approach

Habtamu Shiferaw Adugna

Advisor: Dr. Ravindra Babu Bellam

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September, 2022

Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis entitled "**Dynamic Resource Provisioning for Container-based Virtualization Application using Hybrid Model Approach**" is my own work and has not been submitted to any university for similar purposes. The references used in this thesis are duly recognized by proper citations.

Habtamu Shiferaw Adugna _____ _____

Name of student Signature Date

Recommendation of Advisor

I, the major advisor/supervisor of this research, hereby certify that I have closely advised/supervised the student while developing this thesis and read the thesis entitled "**Dynamic Resource Provisioning for Container-based Virtualization Application using Hybrid Model Approach**" prepared by Habtamu Shiferaw Adugna. Therefore, I recommend the submission of the thesis to the department for further review and evaluation.

Dr. RavindraBabu.B RavindraBabu.B _____

Major Advisor Signature Date

Approval Page

I hereby certify that the recommendation and suggestion given by the thesis review committee are appropriately incorporated into the final thesis entitled "**Dynamic Resource Provisioning for Container-based Virtualization Application using Hybrid Model Approach**" by Habtamu Shiferaw Adugna.

Dr. RavindraBabu.B RavindraBabu.B _____

Major Advisor/Supervisor Signature Date

We, the undersigned, members of the Board of Reviewers of the thesis open defense by Habtamu Shiferaw Adugna have read and evaluated the thesis entitled "**Dynamic Resource Provisioning for Container-based Virtualization Application using Hybrid Model Approach"**, therefore, to certify that the thesis is accepted.

_____ _____ _____

Chairperson Signature Date

_____ _____ _____

Internal Examiner Signature Date

_____ _____ _____

External Examiner Signature Date

Finally, approval and acceptance of the thesis is contingent upon the submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and

_____ _____ _____

Department Head Signature Date

_____ _____ _____

School Dean Signature Date

_____ _____ _____

Office of Postgraduate Studies, Dean Signature Date

ACKNOWLEDGMENT

Above all, I express my heartfelt gratitude to the Almighty God for providing me with health and strength throughout the thesis journey. I would like to thank my advisor, Dr. Ravindra Babu Bellam, for his consistent guidance, support, and suggestions throughout the thesis work, from beginning to end.

I would like to thanks Dr.-Ing. Frezewd Lemma for his comments and suggestions starting from the beginning of my proposal title up to the end my final work thesis. I would like to thank the Computer Vision and Robotics SIG members for their help.

Last but not least, my sincere thanks also go to all my family for their support and advice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SAMPLE CODES

LIST OF ACRONYMS

ARPA	Advanced Research Project Agency
API	Application Programming Interface
AR	Auto-Regression
ARIMA	Auto-Regression Integrated Moving Average
ARMA	Auto-Regression Moving Average
CDT	Cool Down Time
CLI	Command Line Interface
CPU	Central Processing Unit
DCRNN	Diffusion Convolutional Recurrent Neural Network
DL	Deep Learning
ES	Exponential Smoothing
GPU	Graphical Processing Unit
GRU	Gate Recurrent Unit
HTC	High Throughput Computing
IBM	International Business Machine
ICT	Information and Communication Technology
IOT	Internet of Things
IT	Information Technology
LSTM	Long-Short Term Memory
MA	Moving Average
MAPE	Monitoring Analysis Planning Execution
ML	Machine Learning
NN	Neural Network
RAM	Random Access Memory
RNN	Recurrent Neural Network
SDR	Scale Down Ratio
SLA	Service Licence Agreement
SVM	Support Vector Machine
SVR	Support Vector Regression
VM	Virtual Machine

LIST OF NOTATIONS AND SYMBOLS

€n	Elasticity Speed
θo	Over-provision Metrics
θu	Under-Provision metrics
CPUallocated	Allocated CPU usage assigned container
CPUavg	Average CPU usage of container
CPUcurr	Current CPU usage of container
CPUlower_threshold	Lower bonds of CPU assigned at a given time
CPUmax	Maximum CPU utilized by the container application
CPUmin	Minimum CPU utilized by the container application
CPUpred	Predicted CPU at a given time plus one
CPUupper_threshold	Upper bound CPU utilized by the container application
HAproxy	High Availability Proxy
HTTP	Hyper Text Transfer Protocol
MAE	Mean Average Error
MAPE	Mean Average Percentage Error
MSE	Mean square Error
R^2	Coefficient determination
Ref.	Reference
RMSE	Root Mean Square Error
RQ	Research Question
To	Time over provision
Tu	Time under Provision

ABSTRACT

Container-based virtualization is a novel technology that cloud providers are using to provide end-users with cloud services. This technology has many advantages for executing an application (for example, it is lightweight, quick to deploy, and resource-efficient). Cloud providers use cloud technology to offer almost unlimited computing resources. Dynamic resource provisioning is managed effectively by using virtualization-based containerization technologies. However, container-based cloud applications need advanced auto-scaling techniques that swiftly and automatically provision and de-provisioning cloud resources in response to dynamic workload fluctuations without human intervention. This thesis presented a hybrid approach with a deep learning-based method to do auto-scaling of containers in response to dynamic workload changes during run-time to address this difficulty. The four steps of monitoring, analyzing, planning, and executing the control loop are followed by the proposed auto-scaler architecture. To establish the proper scaling measures throughout the analysis and planning phase, the monitor component continuously gathers several sorts of data (hypertext transfer protocol request statistics, central processing unit, and memory consumption). They use a Deep long short-term memory-based prediction model based on long short-term memory (LSTM) to predict future hypertext transfer Protocol request workload and estimate the number of containers required to handle requests in advance, preventing delays brought on by starting or terminating running containers. From both the provider and consumer viewpoints, the suggested method improves resource provision and reduces costs. The experimental findings demonstrate that the hybrid approach model dynamically provisions resources to an application quickly and maintains higher resource utilization than both horizontal and vertical elasticity. Furthermore, when the long short-term memory model is used, the predicted workload aids in using the least number of replicas and, central processing Unit (CPU) utilization to handle the future workload. The proposed deep LSTM framework approach improves CPU utilisation values from 0.999997 to 0.999999, as well as elasticity speedup time values from 1.170 to 1.529.

***Keywords**: Container, Virtualization, Resource, Computing, Workload, Provisioning, Hybrid, Central Processing Unit utilization, Auto scaling.*

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

In today's technological environment, cloud computing has profoundly changed how end-users use computers and other IT resources. During the last few years, the cloud computing industry has expanded in popularity. This is because it offers pay-per-use, always-on, cost-effective, and virtualized access to a large pool of resources. All of these objectives can be achieved without having to worry about the cost of maintaining actual hardware. Leonard Kleinrock, the principal scientist of the initial Advanced Research Project Agency (ARPA) project, presented a concept for offering "computing as a utility" in 1969. Computer networks, according to Kleinrock, will be used as a "utility"(Buyya & Vazhkudai, 2001). Information and communication technology (ICT) has made numerous improvements in various fields since 1969 to help realize this vision (Buyya et al., 2009). Computing has been transformed by developments in networked computing environments into a paradigm of services that may be commoditized and distributed similarly to utilities like water, electricity, gas, and telephony (Buyya & Vazhkudai, 2001). Consumers can use utility computing to get services on-demand based on their needs, regardless of where they are hosted. The utility computing model can be used as a new outsourcing service model that can provide ICT users with numerous options and benefits. The key benefit is that IT-related costs and complexities are reduced because businesses no longer need to invest extensively in or maintain their computing infrastructure and they are no longer bound to certain computing service providers. Indeed, businesses just have to pay for the resources they use, as determined by the computing service providers.

Cloud computing is seen as a solution to problems with traditional IT infrastructure, software sales, and deployment strategies, such as licensing, distribution, configuration, and management of enterprise applications(Bentaleb et al., 2022). Cloud computing has arisen in recent years as an internet-based solution for realizing the utility model of computing for providing compute-intensive applications. Under the cloud computing paradigm, IT and business resources such as servers, storage, networks, and applications can be dynamically allocated to cloud workloads submitted by end-users.

Modern scientific research difficulties need new technologies, integrated tools, and reusable and intricate investigations on distributed computing infrastructures. Above all, computational capacity is required for data processing and analysis to be effective.

Virtualization is a common method for reducing complexity and increasing system flexibility (Bhardwaj & Krishna, 2021) (Bentaleb et al., 2021). Virtualization is a common technique for integrating servers into data and computer infrastructures. The use of virtualization improves resource use and allocation. It serves as a fundamental building block for cloud computing implementations (Bhardwaj & Krishna, 2021). Virtualization is implemented using two key technologies virtualization-based hypervisors and virtualization-based container technologies. Figure1.1. shows Hypervisor-based virtualization.

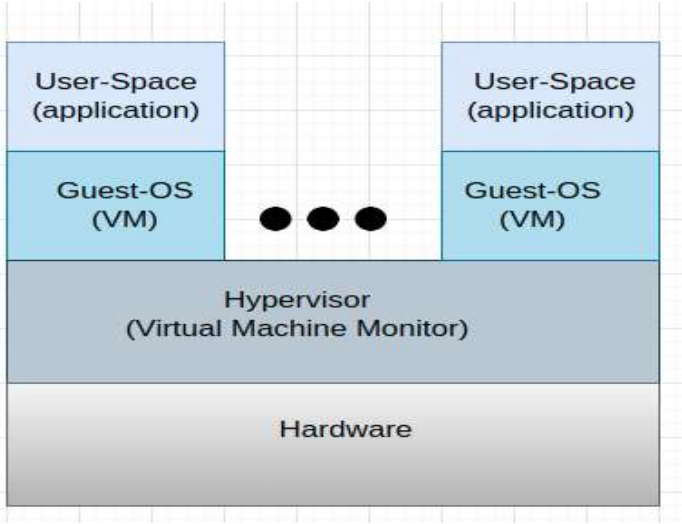


Figure 1: Hypervisor-based Virtualization

Container-based virtualization is a novel technology that cloud providers are using to provide end-users with cloud services(Bentaleb et al., 2022). This technology has several advantages for executing an application (for example, it is lightweight, quick to deploy, and resource-efficient) (Yadav, Pal, et al., 2021a). They are more acceptable because of the ease with which they may be deployed in a reasonable amount of time and the low processing resources required. Containers have evolved as a low-cost alternative to virtual machines that better supports microservice architectures (Bhardwaj & Krishna, 2021). They're frequently utilized by businesses to deploy increasingly diversified workloads resulting from modern applications like big data, IoT, and edge/fog computing in either private or public cloud data centers. Figure 1.2. below shows the container-based virtualization.

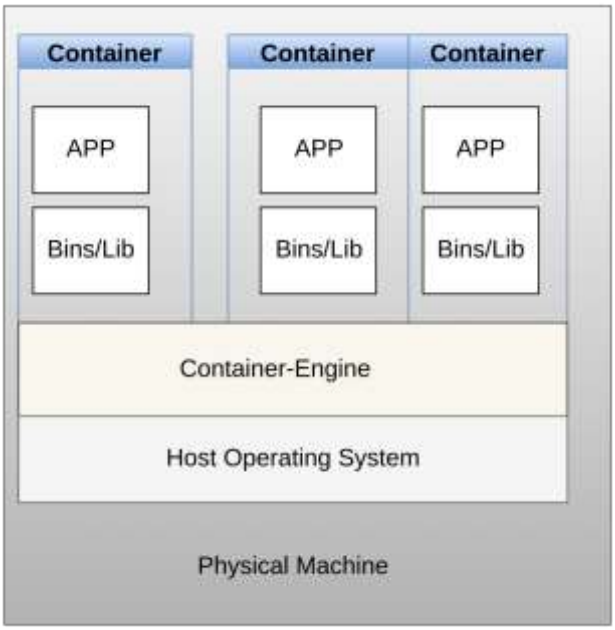


Figure 2: Container-based Virtualization

Containers were introduced by Docker as a free-source platform for users to develop their software. Docker has the advantage of grouping applications into packages that may be moved between any two systems running the Linux operating system. By using Docker, you may deploy software more quickly by separating the infrastructure and applications. The client-server applications in the docker architecture are handled by the docker engine. There are a few key elements in the engine, such as the daemon process, a type of zero-downtime running program, and the Representational State Transfer (REST) application programming interface (API), which specifies screens that allow programs to communicate with the daemon and send commands for what to do. The Docker environment must be configured, built, and maintained using a command line interface. The Docker REST API can control or interact with the Docker daemon via scripting or direct CLI commands thanks to the command line interface (CLI). When clients deploy a web application on a virtual machine, resources are squandered, resulting in inefficient resource utilization. Depending on the number of users, the demand for running an application can increase or decrease. When the number of users grows, the customer requests a "flash purchase" of virtual machines from the cloud provider. In this scenario, a VM-based elastic cloud will not function well since it will not be able to scale up or down resources rapidly, nor will it be able to start or stop VMs quickly. Due to the limitations of VM-based elasticity, they employed the Docker container elasticity. Cloud providers utilize auto-scaling technologies to implement elasticity in the cloud.

Table 1.1: Differences between hypervisor-based and container-based Virtualization

Hypervisor-based Virtualization	**Container-based Virtualization**
A virtual machine (VM) contains a full operating system with its loaded memory management and mixed overhead of device drivers.	The computational engine executes containers. Containers provide a quick startup with optimal execution and are more portable and smaller than VMs.
All virtual machines have their services and private libraries.	The same application libraries can be shared when a single kernel is used by several containers.
It is fully loaded with resources, heavy on speed, and lighter on weight resources, and takes time to develop and launch a virtual machine.	Faster and lighter resources that are extremely dependable for quick start-up and launch

Elasticity is one of the basic characteristics of cloud computing resources offered based on auto-scaling techniques and also manages the workload by dynamically provisioning or de-provisioning computing resources like CPU utilization, memory, and disk storage(Yadav et al., 2022; Yadav, Pal, et al., 2021b; Yadav, Rohit, et al., 2021c). They are numerous advantages of elasticity services those are increased Quality of Service (QoS), reduced power consumption, maximum throughput, and satisfying SLA between cloud providers and end-users. Elasticity overcomes the limitations of physical resources, such as cost, computational power, and storage capacity. When the workload fluctuation on a machine increases or decreases, elasticity can help scale up or down the various computing resources of the system. The occurrence of high variability within workloads can be efficiently addressed using an intelligent resource allocation system. It must avoid over-provisioning or under-provisioning, that is, the distribution of more or fewer resources based on their requirements. Resource provisioning approaches are often divided into two groups, namely reactive and proactive(Yadav et al., 2022). Reactive methods are used to assign resources after the arrival of demands, while proactive methods are used to assign resources first. Proactive methods include tools provided by predicting workloads that would be projected using historical knowledge. Elasticity can be implemented horizontally or vertically or in a combination of both (hybrid elasticity)(Yadav, Pal, et al., 2021a).

Dynamic Resource provision for container-based virtualization for container applications with a different model has a lot of advantages such as, to reduces operating costs, increasing satisfaction between cloud provider and end-user, carbon emissions, and increasing QoS(Yadav, Pal, et al., 2021a). On the other hand, having a minimum of computing resources may not be enough to meet peak workload demands, resulting in poor system performance, lower throughput, slower response times, and service level violations. As a result, it becomes difficult to allocate an ideal number of resources to meet the SLA requirements for the applications. To address the issues raised above, the proposed a hybrid approach elasticity that uses a reactive and proactive technique to provide a resource for container-based virtualization in real-time applications in response to changing workloads. The proposed auto-scaler uses the IBM computing model MAPE control loop to perform CPU usage provisioning(Nikravesh et al., 2017; Yadav et al., 2020).

Existing methods frequently use predefined sets of rules to add or delete resources according to the needs of the application(Yadav, Rohit, et al., 2021b). The proposed approach uses a reactive and proactive (hybrid) approach to perform auto-scaling of CPU usage in response to dynamic traffic changes. It uses Long Short-Term Memory (LSTM) to predict the accurate number of requests at the next time and applies a hybrid elasticity approach to obtain the optimal action to scale in or scale out container-based virtualization applications.

DL algorithms have gained a lot of attention in recent years and are becoming popular in cloud computing. The NN, which is brain-inspired, is one of the most powerful and diverse DL approaches. As characteristic approximates, NNs are widely applicable to a wide range of problems, from regression to robotics(Sharma, n.d.-a; Xu et al., 2022). RNN is interested in this research because of its ability to hold time series analysis, which makes it a suitable choice for predicting cloud resource use with greater precision than conventional approaches.

The RNN frequently fails to solve the predictive issue of highly variable workloads due to its excellent sequential processing power. However, training an efficient RNN for workload prediction could be a significant challenge. The normal RNN cannot learn long-term memory dependencies efficiently due to the gradient vanishing problem. Some variants of RNNs, such as LSTM and GRU, are particularly adept at dealing with this problem(Janardhanan & Barrett, 2018a; Yadav, Rohit, et al., 2021c).

Several approaches are proposed to overcome this problem, including developing a hybrid approach for resource prediction. The majority of the models, and thus the algorithm, have low performance and accuracy and do not take into account the correlation among other multivariate metrics. In this study, an LSTM-based algorithm for predicting CPU utilization for container applications is proposed. The developed framework is used to predict the CPU utilization of container applications.

1.2 Motivation of the Study

Dynamic resource provisioning is managed effectively by using Container-based virtualization technologies, that predict the workload for the entire system and provision the resource efficiently for application(Sharma, n.d.-b; Xu et al., 2022; Yadav, Pal, et al., 2021b). Resources provision and de-provisioning to containers increase the performance of the cloud computing application systems. So, increasing resource utilization increases the quality of services and elasticity of the system. In dynamic provision, add or remove resources dynamically when the resource is required it will allocate, and when it's not required it remove. To accommodate the real-time variable workload, cloud providers use cloud technology to offer almost unlimited computing resources. The advantage of Container-based virtualization is that it allows you to add/remove computer resources at peak times to manage real-time fluctuating workloads without having to turn off the system. Software must be able to accommodate a wide range of clients with varying wants. Resource provisioning can be costly when dealing with a variety of traffic workloads with virtualized resources to meet shifting workload demand during peak hours (threshold values). Dynamic resource provisioning enables us to scale up or scale-down virtualized computing resources in real time based on the actual workload. Instead of using the reactive mode to accomplish the auto-scaling, execute the workload prediction in proactive mode. The proactive workload prediction is used to determine the number of containers or resources required to manage the workload. The hybrid model improves end-user QoS and performance while lowering customer costs and maximizing resource utilization. While scaling up resources and containers, there is extremely little downtime for the application.

1.3 Statement of the problem

When a client hosts an application and its popularity rises or falls over time, the workload on a virtual machine rises or falls as well. Different kinds of applications have different types of clients, each with its own set of needs. An application can be costly to manage the various forms of traffic (workload) with appropriate computing resources to meet workload demand during peak times (the highest numbers of viewers of the system/busy time). Alternatively, keeping the fewest computing resources to save money does not meet the need during peak hours, resulting in poor performance (e.g., longer response times and lower throughput) and perhaps violating SLA requirements (Yadav, Pal, et al., 2021a). Dynamic resource provisioning enables us to scale up or down computing resources in real time based on the actual workload. Cloud providers host client services using container-based virtualization to manage the real-time variable workload. Elasticity refers to the ability to dynamically add or remove resources based on application demand via the cloud. Elasticity is implemented by cloud providers via the auto-scaling technique(Yadav, Rohit, et al., 2021b). Due to the real-time variable workload, a system that hosts an application can be either overloaded or underloaded. For managing real-time variable workloads, cloud providers use an auto-scaling technique to dynamically scale up or down computing resources at the appropriate time. SLA violations, service unavailability, customer loss, power consumption, minimum throughput, and maximum reaction time occur when resources are not allocated/deallocated at the appropriate moment. It can be accomplished by employing an effective deep learning technique to predict the optimal processing of future workload on a server. Based on the predictable load, the required number of containers that may be employed to sustain the varying workload can be calculated, allowing for efficient job completion in a dynamic context(Yadav, Rohit, et al., 2021b).

Challenges in container-based aapplications resource provisioning(Bentaleb et al., 2022a; Yadav et al., 2022; Yadav, Pal, et al., 2021b; Yadav, Rohit, et al., 2021c)**:** To manage the applications in a cloud environment, we consider three major aspects:

- Workload prediction
- Auto-Scaling, and
- Provisioning.

Workload prediction is a process that uses historical data to forecast incoming workloads. Furthermore, the scaling decisions were planned by the auto-scaler based on workload prediction or resource utilization. Finally, provisioning executes the auto-scaling decision to scale up/down or do nothing with the resources in the cloud environment while taking hardware and application provider constraints into account.

Challenges in Workload Prediction: Workload prediction challenges cloud application workload patterns are constantly changing. The predictive workload model is essential for scaling autonomous cloud resource provisioning. This technique has the potential to reduce costs and make better use of resources. The application workload is typically combined into various patterns over time. A single predictive model cannot predict different types of workload patterns.

Challenges in Auto-scaling: Scalability enables application providers to dynamically provision computing power and storage capacity for their applications on demand. The carefully designed auto-scaling approach can save the application provider cost and maintain service quality (QoS). Understanding when to scale up and down resources is critical. Most cutting-edge algorithms are either reactive or proactive.

Challenges in Resource Provisioning: Dynamic resource provisioning dynamically provision resources based on the application workload. Resource over-and under-utilization can be avoided with autonomic resource provisioning. Resource provisioning mechanisms have the following challenges:

- **Under-provisioning:** The application does not have enough resources to handle all incoming server requests(Yadav, Pal, et al., 2021). It could be due to a large number of people or events, or a faulty auto-scaling algorithm. This situation results in an SLA violation, and application providers must pay a penalty to users as well as maintain their reliability. It takes some time for the servers to return to normal operation. This is known as the "cool down period." The auto-scaling policy must allocate sufficient resources based on the application requirements to ensure the quality of service to end users.

- **Over-provisioning** occurs when the number of resources required to process the application exceeds the required resources. In this case, the service provider's dependability is increased(Yadav, Pal, et al., 2021). To a certain extent, SLA violations are minimal in over-provisioning and, to a certain extent, beneficial in dealing with the fluctuating workload. It has an impact on the service provider's profit, and on-demand services become more expensive for clients. There is no such thing as a perfect solution in automatic or autonomous scaling.

1.4 Research Question

This study intended to answer the following four significant research questions.

RQ1: How to dynamically provision resources to avoid over-and under-provisioning?

RQ2: How does the auto-scaler calculate the number of resources required to handle the workload?

RQ3: How to provide the resource Dynamically to container-based virtualization by deep learning technique?

RQ4: How to formulate the Hybrid Approach elasticity to handle resource provisioning problems?

1.5 Objective of the Study

The study's overall purpose is to present a hybrid model technique for provisioning and de-provisioning of resources for container-based virtualization applications that integrates elasticity and workload prediction.

1.5.1 General Objective

The general objective of this study is to design dynamic resource provisioning for container-based virtualization application using hybrid model approach.

1.5.2 Specific Objectives

The following specific objectives will be addressed to achieve the general objective.

- To design a Deep learning prediction model for workload prediction
- To implement the architecture of a hybrid approach model.
- To train the LSTM model with the collected cloud dataset.

- To evaluate the performance of the proposed solution framework approach model
- To scale container-based virtualization applications based on a hybrid approach to reactive threshold policies and
- To predict the future cloud computing resources (CPU utilization) utilizing the proactive deep learning LSTM model.

1.7 Scope and Limitation of the study

1.7.1 Scope of the study

This study focuses on dynamic resource provisioning for container-based virtualization applications using a hybrid auto-scaler and LSTM mechanism to Provision resource usage in cloud computing. The study covers workload prediction and resource provisioning and de-provisioning.

1.7.2 Limitation of the study

The study we proposed does not cover the following:

- The planned work does not include a live container-based virtualization migration.
- We have assumed that sufficient resources are available on cloud provider servers for the proposed work.
- This work focuses only on CPU usage prediction metrics

1.8 Significance of the study

Organizations all over the world are quickly adopting containers. Over 3.5 billion applications are currently running in Docker containers, according to Research and Markets, and 48% of organizations manage containers at scale with Kubernetes. Cloud-based virtualization is the current technology that provides service to end-users by cloud providers. This new technology provides a lot of advantages, such as lightweight, rapid deployment, and efficient resource utilization of an application. Some container-based virtualization applications are big data processing, High-performance computing, Grid computing, High throughput computing (HTC), DevOps, Internet of Things “IoT” and Scientific computing (Nguyen et al., 2020) (Bentaleb et al., 2021). Mostly significant for a cloud provider, Cloud users, and Researchers(Bentaleb et al., 2022c).

1.9 Organization of the Thesis

The thesis is structured as follows:

Chapter One provides background information and motivation for the study; a statement of the problem and the research question to be addressed; a technique to accomplish the study's goal; its scope and constraints; and how the results will be applied. Chapter Two provides a summary of cloud computing, cloud resource provisioning, cloud resource utilization prediction, various architectures and techniques, prior research on the prediction of cloud computing resource utilization, proactive and reactive auto-scaler, and comparison of various strategies used by related work. Chapter Three presents the dataset, the data exploration, and the preparation for building the deep learning models. Chapter Four presented the proposed solutions method, which is used for resource provisioning. Chapter five presented the experiments, results and discussions proposed framework approach of thesis work. Chapter sex conclude the result and it also presents future work.

CHAPTER TWO

LITERATURE REVIEW AND RELATED WORKS

2.1 Introduction

In this chapter, the background and basic concepts used in this thesis and related work are introduced. Section 2.3 is an overview of resource provisioning management in the cloud, Container-based virtualization, auto-scaling techniques, and deep learning time series analysis. Section 2.8 provides the related works.

2.2 Background

Cloud computing is an Internet-based technology that uses a pay-as-you-go paradigm to deliver hosted services over the Internet(Bentaleb et al., 2022c). Cloud computing is a paradigm that allows for simple, on-demand access to a variety of readily available, configurable computing resources with little need for service provider intervention or downtime (e.g., networks, servers, storage, apps, and services). A large number of virtualized ICT tools are exposed as web utilities in the cloud computing paradigm of today, where they can be called upon and released as needed by web-based application programmers.

2.3 Resource Provisioning Management in Clouds

Resource Provisioning management is the process of assigning resources to a collection of apps in a way that satisfies their performance requirements as well as those of cloud service providers and cloud customers(Vinothiyalakshmi & Anitha, 2021). A cloud provider, a cloud user, and an end-user are the three main participants in a cloud environment. They play various roles and have various goals. The goals of the cloud provider are to focus on the effective and efficient use of resources while staying within the bounds of SLAs with cloud users. The application's performance, availability, and cost-effective scaling are becoming increasingly important to cloud users. When it comes to managing the resources in the cloud, the cloud provider does so by allocating them in a way that satisfies SLAs. Applications are hosted by the cloud user using the resources. The workloads that cloud resources process are produced by the end user.

2.4 Container-based virtualization

A recent technology utilized by cloud service providers to offer cloud services to end users is container-based virtualization. This technology is useful for running applications since it is lightweight, easy to deploy, and resource-efficient. Many container-based technologies, including Kubernetes, LXC, and Docker, have been utilized by cloud service providers to build, ship, and run their applications(Bentaleb et al., 2022a). Containers have gained a lot of traction in recent years and are being widely used by the most well-known cloud providers, like Amazon, Microsoft, Alibaba, Google, etc. Among the most well-known services built on containers are Google Container Engine, Azure Container Service, and Amazon ECS.

2.5 Auto-scaling Techniques

To handle a changing workload, elastic apps dynamically provision and release computer resources. A system for automatic resource modifications is required to handle the burden of the applications and enable elasticity. Without human input, the auto-scaler decides on dynamic scaling. Auto-scaling techniques can be divided into two types: (a) vertical scaling, which includes adding/removing resources to a single node in the system, and (b) horizontal scaling, which involves adding/removing nodes.

The following are the four stages of the auto-scaling procedure (MAPE loop):

- Monitoring phase: During the monitoring phase, performance indicators for the system's operation and the state of its application are gathered at various time granularity.
- Analysis phase: collecting current system utilization data and/or predictions of future demand.
- Planning phase: Resource modification actions (such as adding or removing a container-based virtual machine or upgrading memory) can be planned in this phase while maintaining a healthy balance between cost and SLA compliance.
- Execution phase: execute the action.

2.5.1 Auto-Scaler implementations Method

- **Threshold-based rules:** Choosing how to scale the threshold-based rules is determined by certain performance measures (such as the average input request, the average reaction time, typical CPU usage, and a predetermined threshold(Yadav, Pal, et al., 2021b). This approach is said to be straightforward, but it has the drawback of being difficult to select the appropriate performance measurements.

- **Machine Learning:** To dynamically build the resource consumption model for a given workload, machine learning resource estimation approaches are used (online learning)(Duc, 2019; Goli et al., 2021; Imdoukh et al., 2020a; Zhong et al., 2022). Different applications can use the auto-scalers in this way without requiring special setups or preparations. Additionally, because the learning algorithm can self-adaptively modify the model in response to any noteworthy events, they are more resistant to adjustments made during production.
- **Reinforcement learning:** This approach discovers the optimum course of action to be followed in each specific situation based on previous experience (trial and error method)(Abbasi et al., 2021; Rossi et al., 2019). The Auto-scalers receive input from the system after doing each action, which demonstrates how well the action was executed. As a result, the Auto-scalers have a constant tendency to conduct the best actions. This technique's primary drawbacks are its lengthy learning curve and poor initial performance.
- **Queuing theory:** Queuing theory uses system modeling to predict future resource requirements(Jafarnejad Ghomi et al., 2019; Yadav et al., 2022). It entails establishing connections between jobs entering and exiting the system. A straightforward queuing model that represents the load balancer (LB) as a single queue that distributes requests among many container-based virtual machines (VMs) can be used to create elastic applications. The queuing models are frequently solved by simulation. Average response time and average waiting time were two performance metrics that were used. The non-flexibility of the queuing theory model's assumptions means that the parameters and metrics must constantly be recalculated.
- **Control theory:** Developing an application model is necessary for control theory. It entails creating a controller that modifies the resources required by the application demand. There are various kinds of control systems, such as the open-loop controller, which makes use of data from the present state is used to calculate the system's input without requiring feedback to determine whether an objective is achieved. The feedback controller monitors the output of the system and fixes any deviations from the desired value. Lastly, a feed-forward controller forecasts and response before mistakes are made. The most common type of control system is a feedback controller. Input variables are modified following the desired value of the output variable.

But the issue of developing a trustworthy performance model that links input to Output variables is still difficult.

- **Time series analysis:** A certain performance metric is sampled repeatedly at set intervals to Predict future values. To Predict future values, repeated patterns in the input workload can be found using time series analysis.

2.6 Time Series Analysis and Predicting

A certain performance metric is sampled repeatedly at set intervals to forecast future values(Fenila Naomi & Roobini, 2019; Struhár et al., 2020; Wang, 2021). To forecast future values, repeated patterns in the input workload can be found using time series analysis. Time series are frequently used in many fields to depict how measurements change over time. A time series is a collection of data points that are measured at regular intervals of time. A variety of techniques are used in time series analysis to find patterns and predict future values.

2.6.1 Moving Average (MA) methods:

The technique is used to make predictions or to remove noise. The weighted average of the most recent history window of the subsequent values is computed by the prediction formula. There are several variations on this theme, such as the simple moving average that gives each observation the same weight. Forecasting only works when there is no trend. Each observation value is given a different weight by the weighted moving average. wherein the most recent observations are given more weight. When predicting, moving average approaches are still unable to handle big trends.

2.6.2 Exponential smoothing (ES)

Like MA, ES is employed to determine the weighted average of previous data. However, it takes the time series' earlier history into account(Fenila Naomi & Roobini, 2019). It designates weights that get smaller and smaller with time. ES comes in a number of variations, including Single ES, Double ES, and Triple ES. In the case of time series with shifting trends, a single (i.e., simple) exponential smoothing is insufficient. However, double ES is a wise choice when there is a linear trend.

2.6.3 Auto-regression (AR)

The method for making predictions is determined by a linear weighted sum of previous values. Getting the best values for the weights or auto-regression coefficients is the key goal. The AR coefficients are calculated using a variety of methods. The most popular ones use Yule-Walker equations and the determination of the auto-correlation coefficient.

2.6.4 Auto-Regressive Moving Average (ARMA)

Moving average and auto-regression are combined in ARMA. It is appropriate for stationary time series, which lack both trend and seasonality(Fenila Naomi & Roobini, 2019). For non-stationary time series, the Auto-regression Integrated Moving Average (ARIMA) model, an extension of the ARMA model, might be employed.

2.6.5 Machine learning-based techniques

Regression and neural networks are the two-machine learning-based methods that are most widely used(Al-Dhuraibi, Zalila, et al., 2018; Janardhanan & Barrett, 2018a, 2018b; Yadav, Rohit, et al., 2021c). Regression is a popular method for estimating the associations between variables and is based on a statistical model. A neural network is a collection of interconnected nodes (neurons) that have multiple layers, including input, output, and hidden layers. The neural network, in comparison to MA and simple ES approaches, reportedly produces more accurate findings. The second type of time series analysis focuses on spotting time series trends and then employs those patterns to forecast future demand. It typically consists of four types of elements: trends, seasonality, cyclicity, and randomness. In general, it is thought that time series analysis is the key to proactive auto-scaling.

2.7. Time Series Predicting with Deep-LSTM

Deep learning is a group of machine learning approaches that "leverage many non-linear information processing layers for supervised or unsupervised feature extraction, pattern analysis, and classification(Abbasi et al., 2021; Aggarwal, n.d.; Sharma, n.d.-a, n.d.-b; Xu et al., 2022). “When creating predictions and training the model, time series forecasting differs from conventional regression predictive modeling in that the sequence adds an order dependence on the observation that must be carefully kept. In the literature, various deep learning architectures that could be applied to time series forecasting were discovered. Multilayer Perceptron and Long Short-Term Memory are the two most popular among them.

2.8 Related Work

In this section, we summarize recent research on resource provisioning and de-provisioning for managing time-series data workload fluctuations. The goal of the related work is to review the existing approaches in the literature that address time series prediction in the context of cloud computing using traditional and deep learning approaches. Prediction of cloud resource utilization would improve the efficiency of cloud auto-scaling systems. The majority of techniques found in the literature are aimed at predicting workload, performance, and SLA metrics.

Mahendra et al.(Yadav, Pal, et al., 2021a) introduce s container-based virtualization to allocated/deallocated computing resources of the running system(container). Elasticity is an important feature of cloud computing that allocated/deallocated computing resources automatically and provisions and de-provisioning computing resources timely when the workload has fluctuated. It helps to scale up/ down resource when the workload is increase or decrease over the machine. In this study container-based virtualization is used as a hybrid model to provision and de-provisioning computing virtual resources and also uses a muti-criteria decision-making procedure for allocation and d-allocation of the resource. This model used a reactive approach that uses both horizontal and vertical elasticity mechanisms to perform hybrid elasticity through the MAPE model. The proposed approach also minimizes the response time and provides better resource utilization and better quality of service. The experiment results show that hybrid model elasticity is dynamically allocated and deallocated VM computing resources more than horizontal elasticity, vertical elasticity, and docker.

Mahendra et al.(Al-dhuraibi et al., 2017) introduce an ML algorithm for container-based virtualization (docker container) for auto-scaler with dynamic workload changes for the server. The proactive approach of the SVR model performs horizontal elasticity for Docker container in response to workload fluctuating of real-time application. The author uses the IMB model, and MAPE-P principles to perform elasticity using workload prediction using the SVR model. In this study, container auto-scaling is accomplished through the use of a controller that adjusts the number of containers based on the MAPE control loop. The monitor collects various types of data that the prediction model requires to estimate future workloads, such as incoming HTTP requests, CPU utilization, and memory utilization, and stores it in a time series database.

Based on historical time series data, the analyzer employs the SVR-RBF model to forecast future workloads. Based on the predicted workload, the planner decides whether to scale up or down several container replicas to meet the predicted future workload. The Executer is the final phase that reduces or increases the number of container replicas based on the results of the planner phase. The experiments compare the SVR prediction model with some classical models (i.e., Auto regressive, Moving average, ARIMA, etc.)

The basic taxonomy of elasticity was established by Al-Dhuraibi et al.(Al-Dhuraibi, Paraiso, et al., 2018). The authors have discussed several research challenges connected to container-based elasticity, including container monitoring, container-based elasticity, and integrated elasticity between VMs and containers. Al-Dhuraibi et al(Al-Dhuraibi, Paraiso, et al., 2018) proposed the ELASTICDOCKER auto-scaler to conduct vertical elasticity in the reactive technique. In addition, when container resizing is not possible, the authors have executed live migration from one host to another. They also cover themes like resource availability, interoperability, granularity, hybrid solutions, threshold definition, start time, the best trade-off between user needs and vendor interests, prediction estimation mistakes, a unified platform for elastic applications, and assessment methodology. The authors discussed the many research issues related to container-based elasticity, such as container monitoring, coupled elasticity between VMs and VMs, container-to-VM hybrid solutions, and VM elasticity. They've also discussed issues including start-up time, the optimal trade-off between the needs of the user and the interests of the supplier, and prediction estimating inaccuracy.

Al-Dhuraibi et al. presented (Al-Dhuraibi, Zalila, et al., 2018) a React-based approach to perform vertical scaling for the virtual machine as well as for the container. They also allocated compute resources to containerized applications through autoscaling on the virtual machine or the data center. The recommended method of allocating compute resources for both the container and the virtual machine. Use the elastic controller as required by real-time workload fluctuations. The Elastic Controller directly modifies the running Docker container file system through cgroups to increase/decrease computing resources. The recommended auto-scaler automatically scales based on the average CPU or memory usage after a fixed time and then is compared to its under/upper threshold value (i.e., 70% or 90%). When the system statistics reach a certain predefined threshold, the elastic controller immediately increases or decreases the computing resources, as defined in the predefined SLA rules.

The authors simulated the proposed approach, when the average memory usage of a running container was greater than the predefined upper threshold in the last 16 seconds, the auto-allocator allocated immediately increase the size of 256MB memory. It also waits 10 seconds before performing the next auto-scaling action. Recommended elasticity controller passes 18.3% and 70%, for vertical elasticity in container and VM respectively.

Al-Dhuraibi et al.(Al-dhuraibi et al., 2017) proposed a rule-based reactive technique to achieve elasticity. The authors presented an automated metric labeled "Elastic Docker" to create vertical elasticity in a containerized application using this method. The current upper and lower threshold values for the operating container are adjusted based on the various test results, and the value with the shortest response time is chosen. Cloud providers utilize a predefined policy, described in the SLA for resource allocation, to allocate or allot IT resources in a running container. When memory utilization exceeds the current upper limit in a running container, the automated allocator allocates 256 MB of RAM immediately. The auto-allocator, on the other hand, immediately decreases 128MB of RAM if the bucket's memory usage is less than the current threshold. In addition, the authors undertake a direct migration forward from the host to the process.

Taherizadeh et al(Taherizadeh & Stankovski, 2019) published a paper about reactive auto-scaling systems. The proposed autoscaling mechanism employs a multi-level approach that includes infrastructure-level CPU and memory utilization, as well as application-level response time and throughput data that is used to dynamically alter service thresholds based on container workload status at runtime. The experimental results suggest that the strategy provided outperforms rule-based auto-scaling systems that use fixed thresholds under varying workloads.

Imdoukh et al(Imdoukh et al., 2020b) introduce a machine learning algorithm for docker-container auto-scaling with dynamic workload changes. The LSTM prediction model is used to forecast HTTP request workloads to reduce or increase container numbers in the following time window. LSTM, a type of RNN neural network, overcomes drawbacks such as vanishing gradients, making it effective at detecting time-series sequences such as workload over time. The manager nodes and worker nodes comprise the system architecture based on the Docker Swarm framework. The scaling process is carried out by the manager node, which also runs the ML model. Worker nodes, run the container, with one of them acting as a load balancer, receiving HTTP requests and distributing them among worker nodes.

In this study, container auto-scaling is accomplished through the use of a controller that adjusts the number of containers based on the MAPE control loop. The monitor collects various types of data that the prediction model requires to estimate future workloads, such as incoming HTTP requests, CPU utilization, and memory utilization, and stores it in a time series database. Based on historical time series data, the analyzer employs the LSTM neural network model to forecast future workload. Based on the predicted workload, the planner decides whether to scale up or down several containers to meet the predicted future workload. The Executioner is the final phase that reduces or increases the number of containers based on the results of the planner phase. The experiments compare the LSTM prediction model to the ARIMA and ANN models. On the one hand, it matches the accuracy of the ARIMA model with over 600 times faster prediction time; on the other hand, it outperforms ANN in auto scalar metrics related to provisioning and elastic speedup. This study will be expanded to include vertical scaling in the future.

Maenhaut et al(Maenhaut et al., 2020) is introduces the predictive analysis of time series forecasting using the deep learning method LSTM to forecast future server load. The LSTM prediction model is used to analyze and interpret real-time data. The analysis of time series data forecasting using LSTM to predict resource utilization, response time, and network traffic over a cloud server. The LSTM model predicts accuracy using RMSE, MSE, and MAE. The experiments compare the LSTM prediction model to conventional prediction models. The LSTM Deep Learning-based approach outperforms traditional prediction models for time series forecasting. This study will initially focus on cloud server resource utilization and response time via horizontal scaling but will be expanded to include hybrid scaling in the future (horizontal and vertical scaling).

Tang et al. (Tang et al., 2018)proposed a container load prediction model based on the Bi-LSTM approach, which predicts future load based on the container's past CPU utilization load. When compared to the ARIMA and LSTM models, the proposed model had the lowest prediction error. The authors, however, did not explain how to configure the parameters of the proposed model. Furthermore, the paper focuses solely on predicting future load and does not apply it to autoscaling problems.

2.8.1 Summary of Related Works

The following tables illustrate previous work on a cloud resource utilization prediction. The selected papers are from 2017 onwards.

Table 2.2: Summary of related works

Ref.	Monitored Resources	Resource Allocation Technique	Auto-scaling Method	Gap
(Yadav et al., 2022)	HTTP request,	LSTM model to predict server processing load (request) Provision/de-provision predicts server processing load	M/M/c model	Didn't predict the computing resource Only predict the incoming request (server processing load)
(Imdoukh et al., 2020)	HTTP user request, CPU and memory	LSTM model to predict future HTTP workload of container application	Horizontal	Didn't consider vertical scaling
(Yadav, Pal, et al., 2021)	User request, CPU and Memory	Reactive-based approach to the provision and de-provision of computing resources to container application	Hybrid approach model	Doesn't predict computing resources in an advanced way. Doesn't perform the live migration

(Yadav, Rohit, et al., 2021)	CPU, Memory	SVR model to predict future workload and maintain container sustainability	Horizontal	Didn't consider vertical scaling
(Al-dhuraibi et al., 2017)	CPU, Memory	The rule-based approach is based on threshold-based rules to per-form scaling action	ELASTICDOCKER	Limited resources for hosting the machine capacity
(Al-Asaly et al., 2021)	CPU	DCRNN-based deep learning model to predict CPU demand usage for cloud services	Proactive-based Autonomic	Didn't consider randomly Fluctuating workload
(Tao et al., 2018)	CPU, Memory, and HTTP request	Fuzzy-logic approach to provi-sion container resources for big data application	User preference strategy	Complex communication mechanism workflow Selection preference is limited to 3 criteria
(Shahidinejad et al., 2021)	CPU, Memory, and HTTP request	Hybrid approach model (Imperialist computing algorithm and K-Means) and Decision Tree for resource pro-vision to serve heterogeneous cloud workloads	Decision Tree	Complex architecture and didn't consider container virtualization technology

As shown in the tables above, different auto-scaling techniques and approaches are used to predict the utilization of cloud resources, including rule-based (threshold policies), queuing methods, machine learning, and neural network approach models(Al-Dhuraibi, Zalila, et al., 2018; Yadav et al., 2022; Yadav, Pal, et al., 2021b; Yadav, Rohit, et al., 2021b, 2021c). Some deep learning techniques, such as RNN, reinforcement learning, and LSTM, are used to forecast future workloads. Those approaches and auto-scaler techniques are used to provision and de-provision computing resources like an HTTP request, CPU, and memory for container applications(Yadav, Pal, et al., 2021b). To address the gaps listed in the above table, they propose a reliable, fast, and adaptable model that predicts computing resources ahead of time for container applications that must be built with greater accuracy than existing approaches.

This study attempts to solve this problem by expanding on previous solutions and utilizing current and new trends.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Overview

This chapter gives details about the methods, tools, and procedures used to analyses the research problems. It provides a brief overview of the methodology that can be applied to the design, implementation, and evaluation of the proposed architecture. Finally, this chapter described the tools and evaluation methods used to access the results of the study. Figure 3.1 describes the overall research workflow.

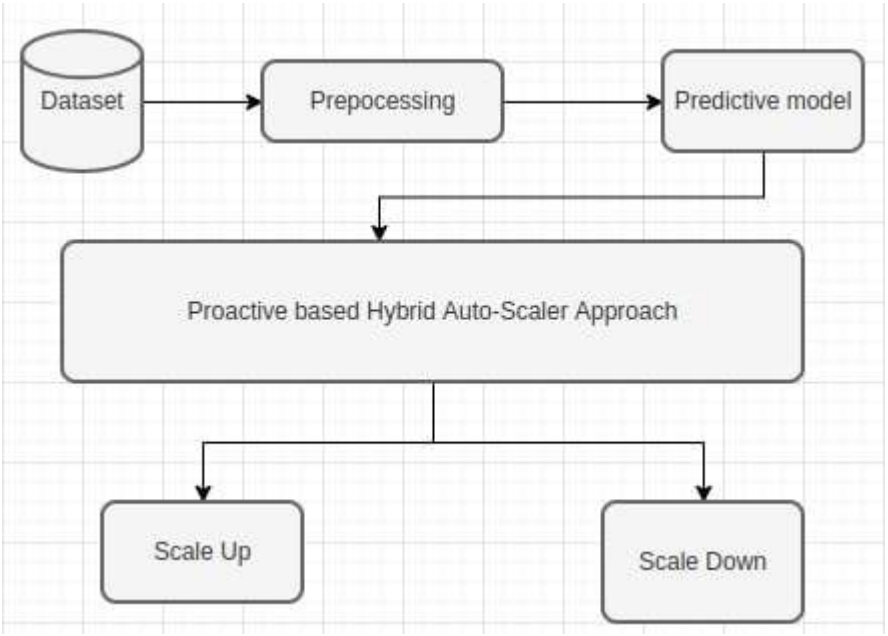


Figure 3:1 The Research work flow

3.2 Data Collection Method

The Azure cloud platform includes over 200 products and cloud services that are designed to assist you in bringing new solutions to life to solve today's challenges and create the future. Build, run, and manage applications across multiple clouds, on-premises, and at the edge using your preferred tools and frameworks.

Azure is a publicly accessible cloud computing platform that provides Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) solutions for

analytics, virtual computing, storage, networking, and other services. It can take the place of your on-premise servers.

- Flexible – Move compute resources up and down as needed
- Open – Supports almost any OS, language, tool, or framework
- Reliable – 99.95% availability SLA and 24×7 technology support
- Global – Data housed in geo-synchronous data centers
- Economical – Only pay for what you use

Microsoft Azure have a best performance than Amazon Web Service and Google Cloud Platform in terms of integration with open-source and on-premise systems.

3.2.1 Dataset

This study uses a hybrid approach and deep learning mechanism to solve resource provisioning issues for container applications, so data is an important part of the study. Microsoft Azure Trace 2017 is used for both training and testing. This dataset can be utilized in this work to conduct a CPU usage analysis on a container application since it is the most widely used dataset in VM workloads. They are freely available online and are accessible datasets. The Microsoft Azure dataset contains an HTTP user request, CPU minimum, CPU average, and CPU maximum, and it is used to train and test the model.

3.3 Microsoft Azure 2017 Dataset

Microsoft Azure 2017 is used to test the efficiency and effectiveness of the proposed process. Microsoft Azure-trace 2017 was used. The dataset used in this work is long-term and large-scale workload traces from a distributed cloud data center. The dataset contains performance metrics for 1,250 virtual machines. The traces include CPU, memory, disc I/O, and network I/O information for both requested and used resources. The virtual machines host mission-critical applications. Collects and analyses the dataset, which is then made publicly available at the Microsoft Azure 2017 dataset trace. The dataset is row-based and sampled every 5 minutes. Each row represents a performance metric observation. The dataset is very large in terms of size, and it contains workload data for 1250 virtual machines, the majority of which contain thousands of observations. To evaluate it in a reasonable amount of time, we must create a smaller sample to work with. The dataset should be small enough that we can build and test deep learning models in a reasonable amount of time and computation. Later, high-performing models can be scaled up and trained across all datasets.

A trend is a smooth pattern that represents data collected over more than a year. Seasonal time series are those with a frequency of one year or less. Cyclical time series, as the name implies, has a repeated pattern, but the frequency is greater than one year. A random series refers to anything that does not follow a specific pattern.

3.3 Description of Data Features

The following is a description of Microsoft Azure 2017 Trace description attributes:

- Timestamp in seconds (starting from 0) when the first VM created
- Count VMs created
- Deployment size (we define a "deployment" differently than Azure in our paper)
- Timestamp VM created
- Timestamp VM deleted
- Max CPU utilization
- Avg CPU utilization
- VM category
- VM virtual core count
- VM memory (GBs)
- Timestamp in seconds (every 5 minutes)
- Min CPU utilization during the 5 minutes
- Max CPU utilization during the 5 minutes
- Avg CPU utilization during the 5 minutes

3.4 Data Pre-processing

It is best practice to prepare the data in such a way that the structure of the problem is exposed to the deep learning algorithm that we will use. We discovered that the dataset we have is clean and has no missing values after exploring it. In this section, we will go over the steps we took to prepare the data for building the prediction models. Machine learning solves problems by learning from data. As a result, before working on deep learning models, it is necessary to first prepare and understand the dataset thoroughly. The dataset is examined using the Python libraries listed below: NumPy is an efficient multidimensional array computation language. Pandas are data structures that are quick and flexible for data analysis. Matplotlib is a two-dimensional plotting library. Seaborn: new plot types, plots that are both stylish and readable. The dataset is read into a data frame using Pandas. Following that, statistics and charts are generated to help the user better understand the dataset.

3.5 Visualization

Creating plots and charts is an excellent way to better understand the data. In general, data visualization aids in learning the data quickly and understanding each attribute independently.

3.6 Feature Selection

All of the features in the data set are not required for the Deep-LSTM model to use as input to accurately predict the adjusted CPU and memory resource utilization. The proposed model employed common features from the two datasets to forecast CPU and memory. The names of the features may differ depending on the provider. The proposed model incorporated the required features by analyzing the trace descriptions provided by providers such as Microsoft Azure (minimum CPU, maximum CPU, average CPU, CPU usage distribution, assigned memory, and cycles per instruction). It would take a long time and a lot of computation to train the model using all of the dataset's features that aren't necessary for prediction.

3.7 Define and Compile the Model

The traces will be converted to a list of time series (sequence) data using a time series generator, and the features and labels will be generated from the sequence. Following that, the Deep-LSTM model will be created by using the input shape of the input layer, regularization, recurrent, activation, and dense layer by setting different parameters and values to support the proposed framework of Keras' input format. As a result, after creating the model and network, the model must be compiled. After compiling the model, the loss function, metrics, and optimizes can be defined. The model compilation defines and binds the loss function in this step. The evaluation metrics are a necessary and obligatory step in the model training process.

3.8 Train the Model

After the network has been constructed, the training and validation generator will fit the constructed model. The model will assess the dataset's fit at this stage. The model will create and fit callbacks so that it can monitor the changes in loss and iteration throughout a predetermined number of epochs across the entire dataset and make dynamic modifications

to the learning rate and early stop. The returned history item records the loss values and metric values throughout training.

3.6 Development Tools

The proposed system is designed and implemented for this study using a variety of development tools. The development tools are summarized and justified in this section. These tools include platforms and methodologies for creating prototypes; UML modeling strategies; https://app.diagrams.net/ and Microsoft Visio research-related tools. A brief evaluation of these development tools is given in the sections that follow.

3.6.1 Design Tools

Design tools are the media that are used to create, communicate, and interpret design concepts. The suggested system's construction makes use of Edraw Max. It is a simple and effective visual design tool for creating flowcharts, network diagrams, UML diagrams, and other graphics with a polished appearance. This approach is preferred because it provides a large number of high-quality shapes, examples, and templates, readily visualizes complex information using a variety of diagrams, and it integrates nicely with MS Office 19 and other programs.

3.6.2 Hardware Tools

The hardware tool that was used during the investigation is shown in Table 3.1 along with its special purpose. Section 5.2.1 contains a description of the comprehensive hardware configuration tool.

Table 3:Research hardware tools are used for implementation.

Devices Name	**Required**	**The uses of these devices for research work**
Hard Disk	1TB	Used to store a large dataset
RAM	16GB	To work together with the GPU to speed up the training process
GPU	500Hz	To reduce the time, it takes to compute and train

3.6.3 Software Tools

Libraries, software tools, and programming tools have been utilized in the following ways:

- **Tensorflow** is a collection of free and open-source programs designed for high-performance numerical computation. Deployment options include desktop PCs, clusters, mobile devices, and edge computing devices. The Google Brain team, which consists of researchers and engineers, created this library. It supports flexible numerical computing, deep learning, and machine learning.
- **Keras** is a high-level API built on Tensorflow (and can be used on top of Theano too). Compared to Tensorflow, it is more approachable and simpler to use. Keras is simple to use, enables modularity, allows for simple expansion, and integrates with Python. These are some characteristics of Keras. It is easy to use, modular, and expandable. It enables quick and easy prototyping support for both CNs and RNs, as well as the hybrids of the two. On both CPU and GPU, it may operate flawlessly. According to the diagram above, Keras is the second most well-known and widely used deep learning framework.
- **Anaconda**: It's a program that installs the Python programming language and all of its modules, depending on the Python edition you have. It includes a navigator program for viewing various settings such as Jupiter notebook, spider, vs-code, and installed modules.
- **Jupyter notebook:** It is a web-based interactive program for configuring, loading Python APIs, and writing Python code.
- **Python:** They are used to run and demonstrate this thesis, and it requires many drivers to configure and package any device installed on the machine.
- **Mendeley Desktop:** It’s most known for its reference manager, which is used to manage and share research papers and generate bibliographies for scholarly articles.
- **Microsoft office packages:** It is used to write document thesis, design diagrams, flow charts, and mark image file names for the proposed system.
- **Docker** is a free and open platform for building, delivering, and operating apps. Docker allows you to keep your apps and infrastructure separate(Yadav, Rohit, et al., 2021b)(Yadav, Rohit, et al., 2021a). You may drastically minimize the time between writing code and executing it in production by leveraging Docker's approaches for shipping, testing, and deploying code quickly (Tao et al., 2018a)(Irufaan, 2021)(Ndamlabin Mboula et al., 2021)(Bankole & Ajila, 2013).

- **MATLAB Deep learning toolbox:** It is creating and deploying a Deep Neural Network algorithm-trained models, and applications. Pictures, time series, and text data can be classified and regressed using traditional neural networks (ConvNets, CNNs) and long-term memory (LSTM) networks.

3.7 Evaluation Method

The outcome will be analyzed to describe how the LSTM model performed on a test set of data. Using various test sizes, the dataset is divided into distinct training and testing sets. The method is evaluated using the test set. The RMSE, MAE, MAPE, and MSE are used as four metrics of evaluation to assess the effectiveness of the models used to test the proposed procedure. In the equations, y is the value of the measured time series, and y' is the value of the predicted time series. There are n samples in each time series.

- $RMSE = \sqrt{\sum_{i=1}^{n} \frac{(Yi-Yi))^2}{n}}$ Equation (1)
- $MAE = \frac{1}{n}\sum_{i=1}^{n} |(yi - yi)|$ Equation (2)
- $MAPE = \frac{1}{n}\sum_{i=1}^{n} |(yi - y'i)|X100$ Equation (3)
- $R^2 = 1 - \frac{\sum_{i=1}^{n}(yi-y'i)^2}{\sum_{i=1}^{n}(yi-y'i)^2}$ Equation (4)

RMSE and MAE are two popular metrics that are scale-dependent for calculating the average error between predicted and actual values. The MSE is a supplementary method for computing average error squares, or the average square difference between anticipated values and actual values, as well as the accuracy and error of the used predictive models. Given that it determines the average of the squared errors, the RMSE is more susceptible to outliers in the data. An average percentage error (MAPE) ranges from 0% to 100%. The result of the model is more robust for each metric when its value is lower. In Equation (4), the coefficient of determination, denoted as R^2, is shown as a goodness-of-fit measure for linear regression models. It is calculated as the square of the coefficients of multiple correlations between observed outcomes and observed predictor values. The greater the R^2, the better the fit of the model to the data.

Evaluating the auto-scaler Provisioning and elastic speedup are employed. The under-provisioning metric, θu, determines how many containers are required to fulfill the demand, as shown in Eq. 4. The θo overprovisioning metric tracks the quantity of supplied containers that are more than the required number, as indicated in Eq. 5. According to Eqs. 6 and 7, the auto-scaler is under-and over-provisioned during the periods represented by T_U and T_O, respectively. Elasticity speedup (n) in Eq. 8 compares the performance for cases with or without auto-scaling, where subscript a denotes the case with the use of auto-scaling and n otherwise. This metric evaluates performance loss when the value is less than one, and gains when the auto-scaler is applied to the alternative case.

$$\theta u[\%] = \frac{100}{T}\sum_{t=1}^{T}\frac{max\ (demand(t) - supply(t), 0)\Delta t}{demant(t)} \quad \text{Equation (5)}$$

$$\theta o[\%] = \frac{100}{T}\sum_{t=1}^{T}\frac{max\ (demand(t) - demand(t), 0)\Delta t}{demand(t)} \quad \text{Equation (6)}$$

$$Tu[\%] = \frac{100}{T}\sum_{t=1}^{T}\max\ (sgn\ (demand(t) - supply(t)), 0)\Delta t \quad \text{Equation (7)}$$

$$To[\%] = \frac{100}{T}\sum_{t=1}^{T}\max\ (sgn\ (demand(t) - demand(t)), 0)\Delta t \quad \text{Equation (9)}$$

$$\epsilon n = \left(\frac{\theta u,n}{\theta u,a}.\frac{\theta o,n}{\theta o,a}.\frac{Tu,n}{Tu,a}.\frac{To,n}{To,a}\right)^{1/4} \quad \text{Equation (10)}$$

CHAPTER FOUR

PROPOSED DEEP LSTM AND HYBRID AUTO-SCALER

4.1 Overview

The proposed framework architecture of dynamic resource provisioning for container applications using the hybrid auto-scaler approach is detailed in this chapter. Ideally, this chapter can be divided into three sections: the first describes the proposed Framework approach; the second provides an overview of cloud resource managers, and the third provides an overview of the Deep-LSTM prediction model and the hybrid Auto-scaler approach. The final section explains how Pseudocode scales up and scales down CPU usage for container applications using a hybrid auto-scaler approach.

4.2 Proposed Framework Approach

The proposed framework is presented in Figure 4, it has mainly two components: Autonomic scaler manager and cloud resource manager. The autonomic scaler manager is responsible to estimate the required resources and triggering scaling action. Cloud resource Manager (CRM) is responsible to collect and control provisioned resources to the container. In this section, we describe our proposed framework architecture design, which is based on the Monitor-Analyze-Plan-Execute (MAPE) loop(Bentaleb et al., 2022). The proposed architecture is a combination of Hybrid elasticity and Deep learning techniques that perform resource provision for container applications based on workload prediction. In this section, we describe our proposed framework architecture design, which is based on the Monitor-Analyze-Plan-Execute (MAPE) loop as shown in the Figure below. The proposed solution framework architecture includes the following components:

- Autonomic Manager (AM)
  - ❖ Deep-LSTM Prediction Model
  - ❖ Hybrid Auto-Scaler Approach
- Cloud Manager Resource (CMR)

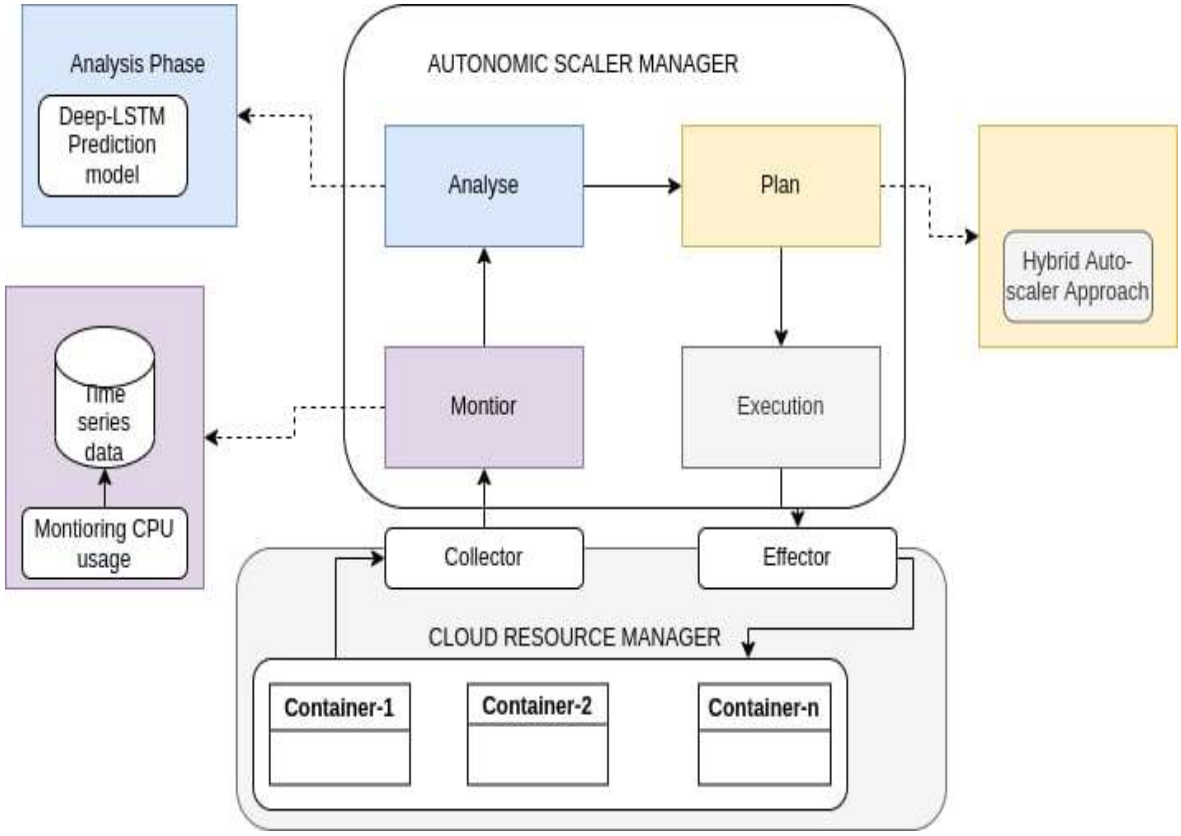


Figure 4:Proposed Framework architecture

4.3 Cloud Resource Manager

4.3.1 Effector

The effector receives and executes scaling action commands (from the autonomic auto-scaler manager). Furthermore, it validates the currently managed cloud resources after each scaling action to determine whether they meet the desired ones

4.3.1 Collector

In time series forecasting, a multivariate prediction model typically includes multiple feature values at each time step. The resource utilization data of various features, including the CPU, memory, disk I/O, and network received throughput, is collected by the collector component. The monitoring server of the autonomic auto-scaler manager receives all the collected data.

4.4 Autonomic Manager (AM)

The autonomic manager component is the grouping of a hybrid auto-scaler approach and Deep learning techniques to provision cloud computing resources for container applications and also use the IBM computing model MAPE control loop(Mostofi, 2021; Salmanian et al., 2020).

4.4.1 Hybrid Auto-Scaler Approach

This section presents two scaling methods, reactive and predictive approaches to realizing the hybrid scaling solution. Hybrid auto-scaling is capable of supporting horizontal and vertical scaling(Yadav, Pal, et al., 2021). Hybrid scaling techniques benefit from both vertical scaling fine-grained resource control and horizontal scaling high availability. As a result, hybrid scaling appears to be a promising solution for effective auto-scaling. As previously stated, the proposed framework approach architecture is based on the MAPE loop, and our main contributions are the Deep-LSTM prediction model (as the analysis phase) and the hybrid auto-scaler (as the planning phase), both of which are recognized as the auto-scaler core(Tang et al., 2018). Figure 5 below depicts the auto-scaler architecture, which consists of four major phases. Next, we provide a thorough explanation of each step in this auto-scaler.

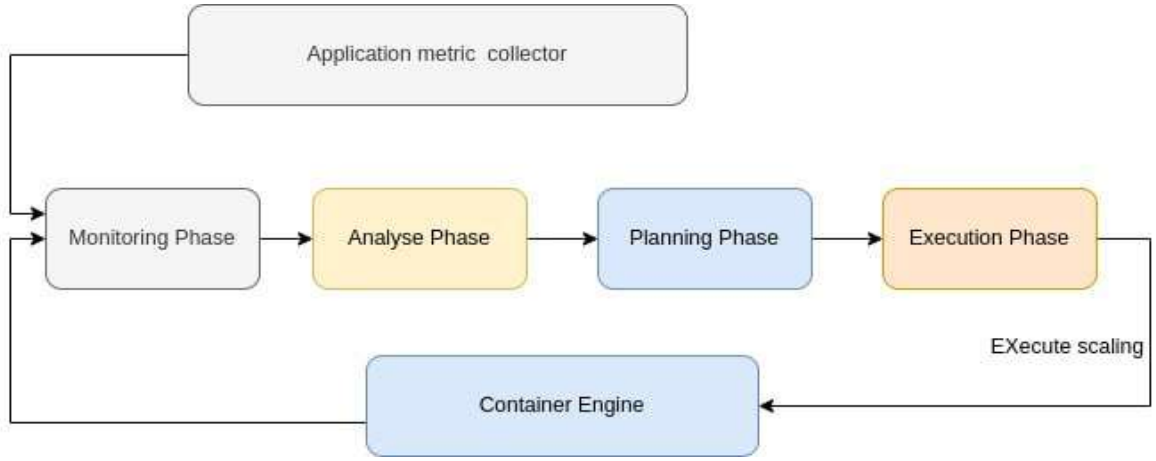


Figure 5: Autoscaler architecture adopted from (Imdoukh et al., 2020).

4.4.1.1 Monitor Phase

The monitor component is in charge of gathering the monitoring metrics required by the Auto-scaler. It polls for application status at regular intervals using interfaces exposed by cloud providers. A time window is set, and monitoring data collected during that period is saved locally for forecasting future trends.

4.4.1.2 Analysis Phase

The analysis component accepts monitored metrics as input. It builds a model based on historical data to forecast application resource demand soon. It updates its predictive model and makes predictions for the next time interval at the end of each time interval.

4.4.1.3 Planning Phase

This is the foundation of the entire auto-scaler. It determines optimal scaling strategies to meet SLAs based on real-time resource usage and predicted resource demand. It is also critical for planners to release the appropriate number of resources to avoid resource over-provisioning or under-provisioning.

4.4.1.4 Execution Phase

The final phase is in charge of carrying out the scaling strategy decided upon during the planning phase. The execution can be carried out using the cloud service provider's API. The complexity of this phase stems from the need to support a diverse set of APIs to communicate with multiple cloud service providers

4.4.1.5 Container Engine

A container engine is a piece of software that accepts user requests, including command line options, pulls images, and runs the container from the end user's perspective.

- Handle user inputs
- Handle inputs over APIs
- Pull the container image from a registry server
- Use your graphics driver to decompress and expand container images on disk
- Prepare mount points for containers, usually using copy-on-write storage
- Prepare meta-data to pass to the container run-time to launch the container correctly, based on container image defaults or user input that override the defaults
- Call the container run-time

4.4 Deep-LSTM Prediction model

The proposed Deep-LSTM prediction model is discussed in this section.

4.4.1 Neural Network Architecture

In this section, we will go over the proposed Prediction model, which makes use of Deep Learning LSTM models. Long short-term memory (LSTM), a successor to recurrent neural networks (RNN), is a special type of RNN known to overcome RNN's vanishing gradient drawback(Yadav, Pal, et al., 2021c) (Gao et al., n.d.; Yadav et al., 2022; Yadav, Rohit, et al., 2021a, 2021b, 2021c). Because of this capability, LSTM is well suited to predicting the next sequence in a time series dataset, such as workload over time. The Deep-LSTM network is made up of a series of LSTM units/cells that are linked together. Figure 6 depicts the proposed Deep-LSTM model, which has two layers that move forward and backward. Each hidden layer has an input layer with 10 neural cells for the last 54-time steps, each with 30 hidden units. The hidden layers' final outputs in both forward and backward directions are concatenated and used as input to the dense. Because the dense is a fully connected layer, each neuron in the dense receives input from all neurons in the previous layer. The number of neurons specified in the dense will affect the output shape. The dense is used to generate predictions in this study.

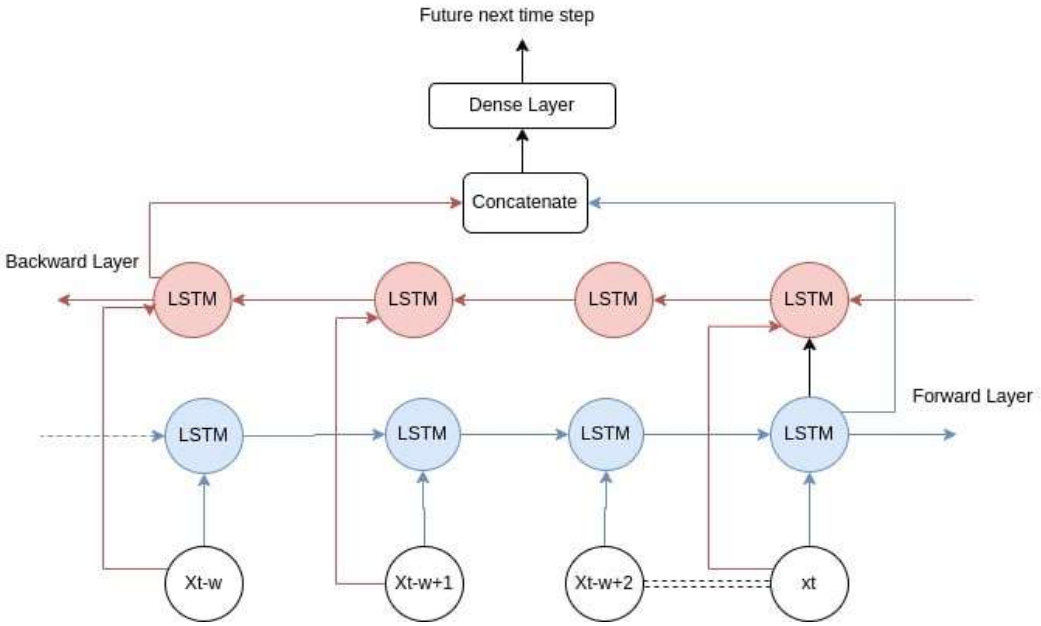


Figure 6: Proposed deep LSTM network architecture

4.5 Proposed Hybrid Auto-scaler Planning Phase Algorithm

After the analysis providing resource demand in the next time interval, we attempt to explain our scaling strategy in this section. Our proposed algorithm is depicted in Algorithm II. During the run time, the monitor gathers metrics from the container at each time interval. Scale CPU utilization based on current real-time metrics using a reactive-based approach and predict CPU usage. Using the predicted value as input and applying a hybrid auto-scaling decision to scale in and scale out CPU usage. The proposed hybrid auto-scaling algorithm solution has illustrated as follows:

Algorithm I: Planner Algorithm

Input: CPU_{cur}, CPU_{pred}, $CPU_{allocated}$, $CPU_{upper\_threshold}$, $CPU_{lowerr\_threshold}$

Output: scale Up, Scale Down

1. N=number_of_running containers
2. while a running_of_container **do**
3. if CPUcur > $U_{upper\_threshold}$ **then**

   Scale up vertically

4. else if CPUcur < $CPU_{lower\_threshold}$ **then**

   Scale down by decreasing container replicas

5. else if CPU_{pred} > CPUcur, **then**

   Increase container replicas

6. else if $CPU_{pred} - CPU_{allocated} > CPU_{cur}$ **then**

   Scale up vertically

7. end if

   update the Number of containers;

8. end while

4.6 Proposed Framework Flow Work

This section describes the overall process of the proposed framework pseudo code.

Algorithm II: Algorithmic Framework

1. Initial system deployment;
2. While the system is running do
3. For every time interval do
4. /* collecting the monitoring metric from container-based (CPU utilization and user request) */
5. Collect monitoring CPU usage of container metrics.
6. Scale Container-based application horizontal scaling) according to the current workload of CPU utilization of container
7. /* estimate the workload of CPU utilization in the analysis phase */
8. Calculate predicted CPU usage resource demand
9. Scale container-based application vertically scaling
10. Execution ();
11. End for
12. End while

In general, the main idea behind the Hybrid based approach scaling is that we use vertical and horizontal scaling to achieve CPU usage provisioning in the variations of workload(Imdoukh et al., 2020). Planner Scaling decisions algorithm based on the upper and lower thresholds predefined by the cloud provider and predict the future CPU usage workload demand. If an application's CPU usage exceeds a certain threshold, the scaler allocates more CPU to each container to quickly meet the burst of resource demand. When the scaling down criterion is triggered, the scaler simply de-provisioning resources by reducing the number of containers, i.e., scaling down horizontally. The proactive based approach to scaling is to meet resource demand soon. They decided to use horizontal scaling first to meet resource demand and then tune provisioning through vertical scaling. The Algorithm above depicts the entire procedure.

CHAPTER FIVE

EXPERIMENTS, RESULTS, AND DISCUSSIONS

5.1 Overview

This chapter describes the model's implementation, results, and discussions. The working environment and resource provision for container applications via Deep LSTM implementation are discussed. This section displayed important portions of the code, while the appendix section outlined the imperative detail codes. After having discussed the specifics of the implementations and experiments to be carried out, this chapter presents the results of the experiments. The performance of the proposed framework approach was evaluated using the evaluation metrics discussed in the third chapter. Furthermore, the performance of the proposed model is compared to existing resource provisions for container applications.

5.2 Working Environment

The hardware specification used to implement the Deep-LSTM experiment is described in this section.

Desktop Computer: used for implementing and training the CPU usage by the LSTM model.

- Model: DELL OptiPlex 9020
- Operating system: Windows 10
- Processor: Intel ® Core™ i5-4580 CPU @ 3.29GHz x 4
- Graphics Card Upgrade: GeForce RTX 2070 Super 8 GB RAM
- Memory Size: 12.00 GB
- System Type: 64-bit Operating System, x64-based Processor

Laptop computer: The laptop computer is utilized for creating a network architecture.

- Device name: DESKTOP-7UQ8161
- Operating system: windows 10 pro
- Processor: Intel ® Core™ i7-7500U CPU @ 2.70 GHz 2.90GHZ
- Memory Size: 8.00 GB
- System Type: 64-bit Operating System, x64-based Processor

5.3. Setup Environments

5.3.1. Integrated Development Environments and Editors

Jupyter Notebook: is the most widely used Block-wise programming IDE for editing Python code and visualizing diagrams. Jupyter Notebook 5.2.2 is used in this thesis work.

Colab is an Online Jupyter Notebook that includes several utilities for running code on the GPU, Tensor Processing Unit (TPU), and Random-Access Memory (RAM).

Visual-Studio-Code is a popular source code editor IDE that can be easily configured and used to edit a variety of programming languages, including C++, Python, Java, and others. For this work, Visual-Studio-Code version 17.3 was installed in addition to the Jupyter notebook.

5.3.2. Programming Language and Libraries Used

- Python 3.8.3 is used in this work because it has many packages and libraries for scientific computing as well as large community support.
- Keras 2.4.3 is a deep learning architecture designing framework that is used to apply various operations such as convolution, pooling, and other deep learning processes, as well as to load the pre-trained model.
- TensorFlow 2.4.1 is used for computations as well as for connecting the retrained model to graphs.

5.4 Training Procedure

- Collect massive datasets from real-world cloud service providers
- Pre-processing dataset
- Scaling and transforming the data by train and test split
- Building the model
- Compile the model
- Train the model

5.5 Implementation Techniques for Dataset Processing

5.5.1 Dataset Description

In this thesis, the datasets obtained from real cloud environments of Microsoft Azure are used to test the performance and verify the method's effectiveness. The following figure illustrates the code snip of reading the Azure trace dataset.

```
#------------------------------------------------------------
# LOAD THE DATASET AND PLOT THE OBSERVATIONS
#------------------------------------------------------------

df = pd.read_csv("C:\\User\\admin\\CPU\\Azure.csv')
df['timestamp'] =  pd.to_datetime(df['timestamp'])
df = df.set_index('timestamp')
df.head()
df.plot(figsize=(16, 8))
plt.show()
```

Simple code 1:Load the Dataset

5.5.2 Preparing the Dataset

The original dataset contained too many features that were unnecessary for predicting cloud resource utilization.

```
def preprocess(url):
  headers=['timestamp','vm id','min cpu','max cpu', 'avg cpu']
  with urllib.request.urlopen(url) as dl_file:
    with gzip.open(dl_file) as f:
        df = pd.read_csv(f, header=None, index_col=False,names=headers,delimiter=',')
        df['timestamp'] = pd.Timestamp(df['timestamp'], unit='s')
        df = df.groupby('timestamp').sum()
        return df
```

Simple code 2: Preparing the Dataset

5.5.3 Scaling and Transforming the Data

It goes through several steps of preprocessing before being placed into the suggested model. To begin, the minimum-maximum scaler is used to scale all dataset's normal values to a range of 0.1 to 1.

```
#------------------------------------------------------------
# SCALING VALUES
#------------------------------------------------------------
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = pd.DataFrame(scaler.fit_transform(train), columns=df.columns)
test_scaled = pd.DataFrame(scaler.transform(test), columns=df.columns)
```

Simple code 3:Scaling and transforming of data

5.5.4 Create the Model

Several features are used to create the sequential model: optimizer, loss, and metrics. Using the Keras Sequential API, the network is built up each layer at a time. The layer of LSTM cells is the network's heart, with dropout to prevent overfitting. The network gains representational power from the completely linked dense layer with Leakyrelu activation.

```
#-------------------------------------------------------------
# USING A Deep-LSTM MODEL FOR PREDICTION ON TIME SERIES
#-------------------------------------------------------------
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(512,input_shape=(X_train.shape[1], X_train.shape[2]),return_sequences=True))
model.add(tf.keras.layers.LSTM(512,return_sequences=False))
model.add(tf.keras.layers.Dense(3))
model.summary()
```

Simple code 4:Create the model

5.6 Model Training

5.6.1 Callback Set (Hyperparameter) and Train

The training parameters specify the regulator, the number of epochs, and the batch size, as well as the number of GPU, dataset, epoch, and batch size configuration;

```
callbacks = [es , lr_red]
history = model.fit(X_train, y_train,
                    epochs=200,
                    validation_split=0.25,
                    batch_size=256,
                    verbose=1,
                    shuffle=False,
                    callbacks = callbacks)

#-------------------------------------------------------------
# I TRAINED MY MODEL ON BATCHSIZES - 64,128,256
#-------------------------------------------------------------
```

Simple code 5:Setting Callback Training

5.7 Implementation of Model Evaluation

The MAE is used to calculate the differences in errors between paired observations expressing the same phenomenon. The MSE computes the average of the error squares, which is the average difference between the expected and real values. The RMSE is used to calculate the error or accuracy of a model's prediction. It also computes the error using standard deviations. The MAPE is calculated to measure the accuracy of the proposed model. The calculation is achieved by taking the difference between the current value and the predicted value and dividing the difference by the actual value.

```
#----------------------------------------------------------------
# LETS CHECK HOW GOOD THE MODEL PERFORMED
#----------------------------------------------------------------
testScore_1 = math.sqrt(mean_squared_error(y_test[:], preds[:]))
print('Test Score: %.2f RMSE' % (testScore_1))

testScore_2 = math.sqrt(mean_absolute_error(y_test[:], preds[:]))
print('Test Score: %f MAE' % (testScore_2))

testScore_3 = np.mean(np.abs(preds - y_test)/np.abs(y_test)*100)
print('Test Score: %f MAPE' % (testScore_3))
```

Simple code 6: Implementation of evaluation model

5.8 Experimental Results

This section discusses some of the implementation process results, such as data preprocessing, normalization, inverse transforming, main training results, and analysis of auto-scaler performance in terms of provisioning and de-provisioning, as well as elasticity speed on the LSTM model on Microsoft Azure cloud workload traces. It also discusses the proposed algorithm's output and collaboration with others, specifically stack LSTM and GRU, using various evaluation metrics such as MAE, RMSE, R^2, and MAPE. Using Microsoft Azure datasets, the proposed LSTM model is evaluated and its accuracy and prediction speed are compared to those of the stack LSTM and GRU.

5.9 Experiment Explanation

We conduct experiments to evaluate the hybrid custom auto scaler, as described at the beginning of this section. In the first experiment, we train and test the prediction models using Microsoft Azure datasets. We divided each dataset into 80% for training and 20% for testing. The stack LSTM and GRU prediction models are compared to the LSTM prediction model described in Chapter 4. All prediction models were built with Python, GPU, and Google Colab's tensor processing unit (TPU). They simulated the environment in the second experiment to see how it performed in terms of resource provision accuracy and elastic speedup. The simulation was created with the Python programming language and runs on the Google Colab environment

5.10 Evaluation Metris

In this section, experiments are carried out to demonstrate the feasibility of the proposed model(Imdoukh et al., 2020). Assessment metrics and other workload prediction models have been introduced in measurement metrics and baseline processes.

The MAE, RMSE, $R^{2,}$ and MAPE were computed as performance indices to compare the results of the developed resource prediction models. Table 4 shows that the LSTM model achieves lower prediction error values on R^2, RMSE, MAE, and MAPE metrics when compared to the Stack LSTM model predictions, as well as lower prediction error values on MAE metrics when compared to the GRU model prediction. This means that the LSTM outperforms the stack LSTM and GRU models in terms of prediction accuracy. Furthermore, the LSTM model's correlation values in both predictions are higher than those of the stack LSTM and GRU models, indicating that the LSTM can fit the data better than the stack LSTM and GRU models. Finally, when compared to the Stack LSTM and GRU, LSTM not only improves on all metrics but also has a faster prediction speed. Table 5 shows the LSTM prediction speed and Elasticity speedup, LSTM's highest prediction speed and elasticity speedup than GRU and stack LSTM prediction model.

Table 4: LSTM, stack LSTM, and GRU prediction

Model Type	**LSTM**	**Stack LSTM**	**GRU**
RMSE	28232.94	382759.04	**27488.19**
MAE	**129.709539**	570.196129	27488.19
MAPE	129.709539	0.217560	**0.010602**
R^2	**0. 999999**	0.999998	0.99998

Bold values indicate the best results

In the second experiment, the auto-scaler was evaluated in the simulation environment by analyzing its performance in terms of provisioning and elasticity speed. The under-provisioning metric defines the number of missing CPU utilization requirements to meet service-level agreements (SLAs) normalized by measured time for resource provisioning accuracy. Over-provisioning metrics identify the amount of excess CPU utilization provided by the auto-scaler. Finally, the elastic speedup is the autoscaling gain compared to the no autoscaling case, which is the case when using autoscaling.

A comparison of auto-scalers for all cases analyzed is summarized in Table 5. In the comparison, the prediction speed of LSTM and GRU is relatively fast, as shown in Table 5. Although LSTM predicts incoming workload faster than GRU and stack LSTM.

Table 5:LSTM, GRU, and stack LSTM Prediction speed and elasticity speedup

Model	**LSTM**	**GRU**	**stack LSTM**
Speed(ms)	**0.2**	0.3	0.3
Elastic speedup	**1.529**	1.17	1.182

Bold values indicate the best results

Shown the results of the experiments in Tables 4 and 5 by hosting a web application is a cloud-based service using the proposed framework approach. Container orchestration frameworks enable the deployment and management of a multi-tiered distributed application as a collection of containers on a cluster of nodes. Figure 7 depicts the high-level architecture of a simplified containerized system deployed on a cluster of nodes. Docker containers have seen widespread industry adoption in recent years due to their simple approach to packaging web applications and all of their dependencies into a single standardized unit for software deployment. In this architecture, there are two nodes: the first is the manager node, and the second is the worker node.

The manager node serves as the system's server, while the worker nodes serve as the system's clients. The manager node checks the nodes or nodes that request to join the cluster regularly to maintain the state of the cluster nodes. Additionally, it manages job scheduling for tasks like replicating containers and deploying, removing, starting and stopping containers. Another important task of the manager node is to host a containerized application and deep LSTM and hybrid auto-scaler which exists in the manager node to increase system elasticity by adjusting the CPU utilization based on predicted future workload.

The second node is a worker node, whose sole purpose is to host a running docker container. In this system, the worker node has a container load balancer, which receives incoming requests, distributes workload, and provides feedback to the user. HAProxy is a free and open-source proxy server for HTTP load balancing. It has evolved into the de facto industry-stand-

ard open-source load balancer and is now included with the majority of popular Linux distributions. It is also frequently used as the default load balancer in cloud computing platforms.

Worker nodes 2-4 host containerized application replicas. The load balancer sends the request to the containerized application, which response by performing some processing that needs resources like CPU and memory.

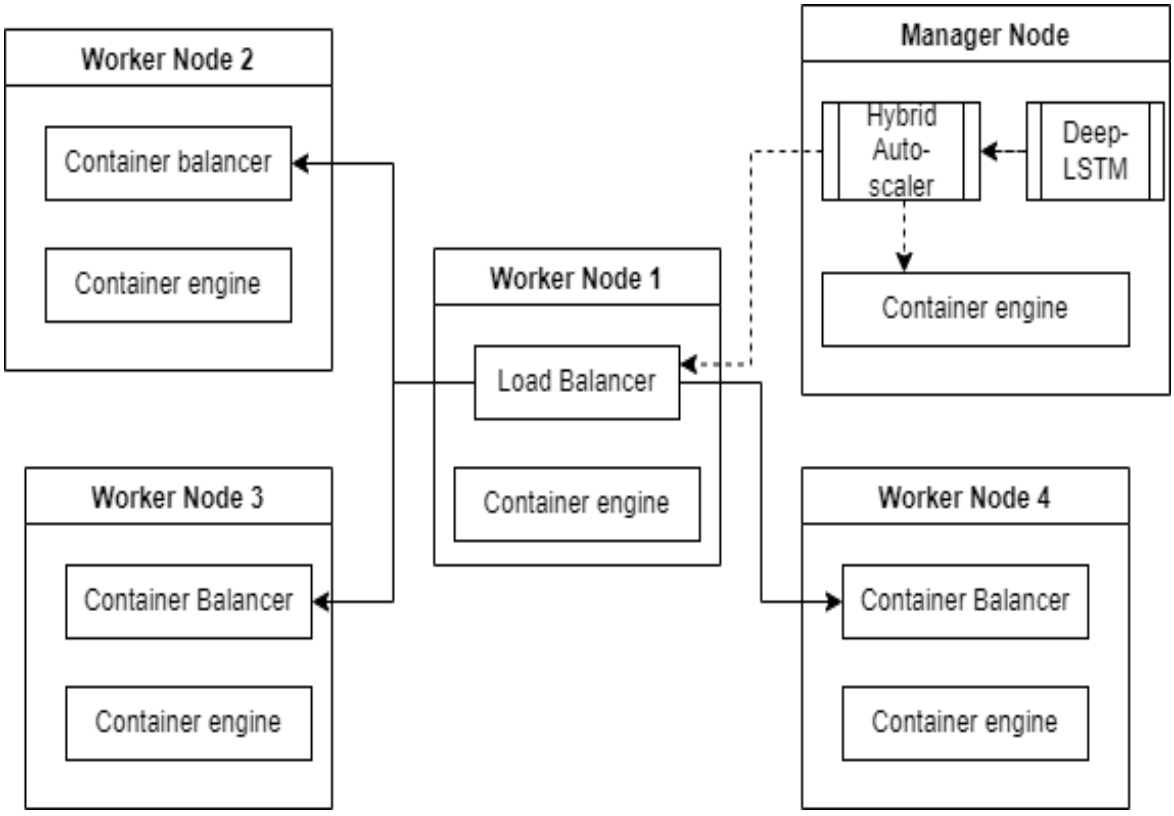


Figure 7:Hosting web application

The graph shown below depicts the Prediction of the LSTM model min CPU, Max CPU, and Avg CPU resource utilization of container-based virtualization application.

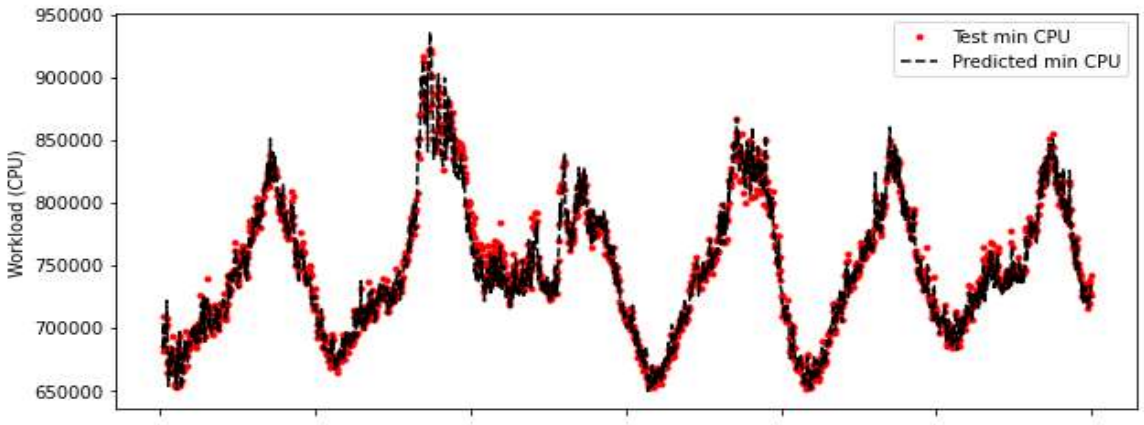


Figure 8: LSTM prediction using Microsoft Azure minimum CPU dataset

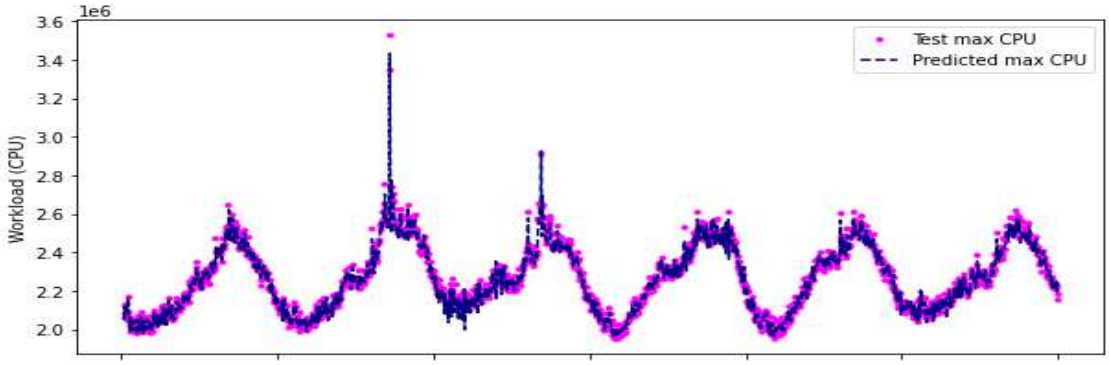


Figure 9: LSTM prediction using Microsoft Azure maximum CPU dataset

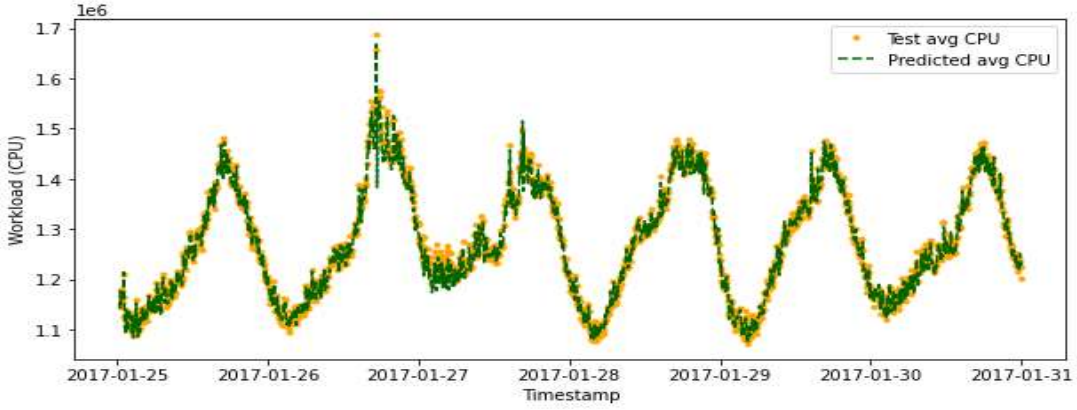


Figure 10:LSTM prediction using Microsoft Azure average CPU dataset

The graph shown below depicts the Prediction of stack LSTM model min CPU, Max CPU, and Avg CPU resource utilization of container-based virtualization application.

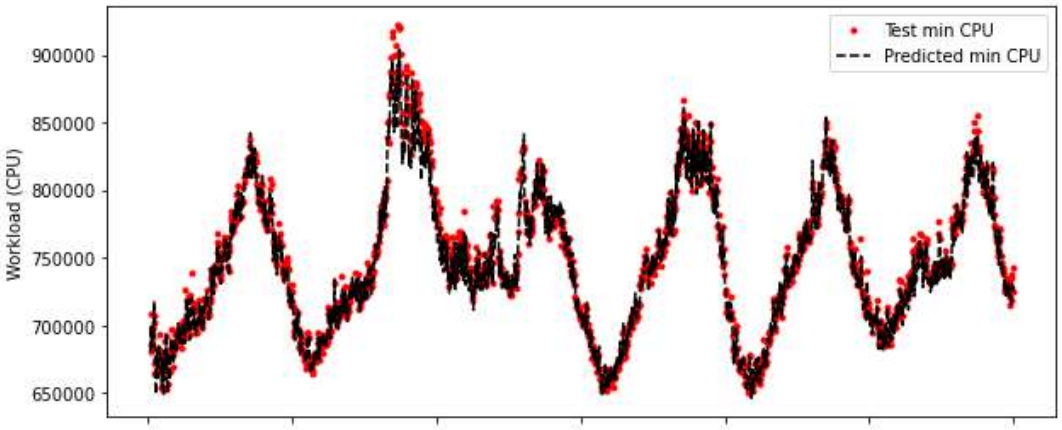


Figure 11: stack LSTM prediction using Microsoft Azure maximum CPU dataset

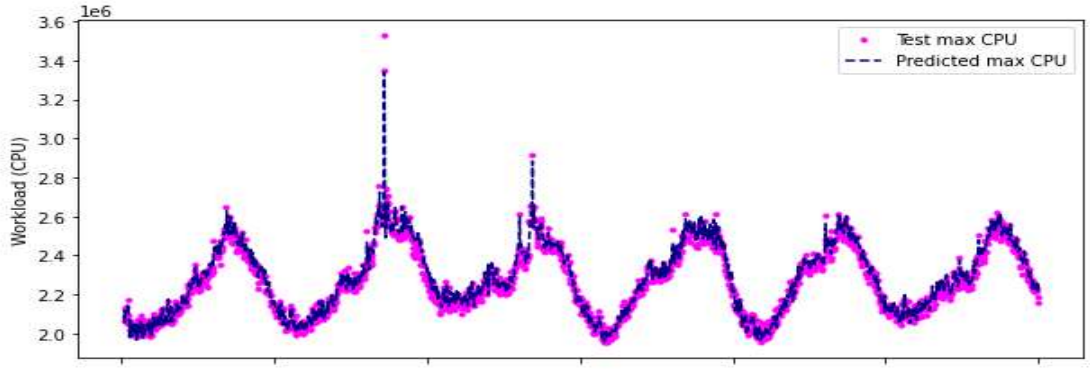


Figure 12:stack LSTM prediction using Microsoft Azure maximum CPU dataset

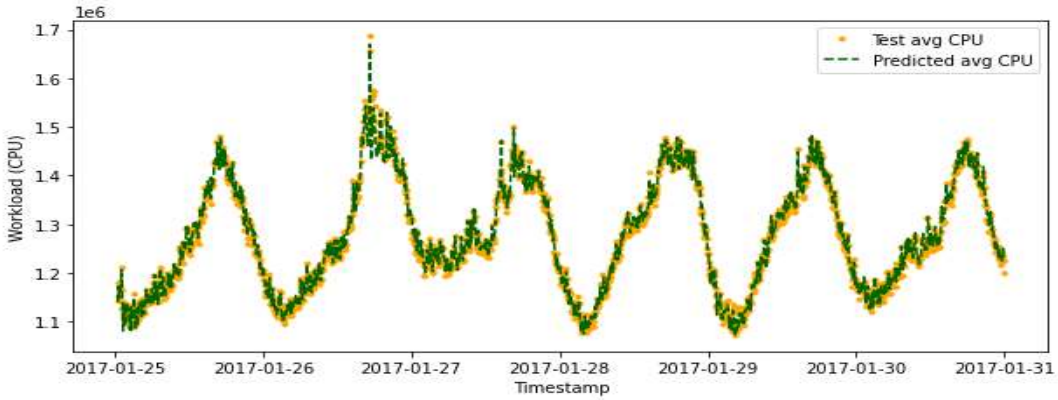


Figure 13:stack LSTM prediction using Microsoft Azure maximum CPU dataset

The graph shown below depicts the Prediction of the GRU model min CPU, max CPU, and Avg CPU resource utilization of container-based virtualization applications.

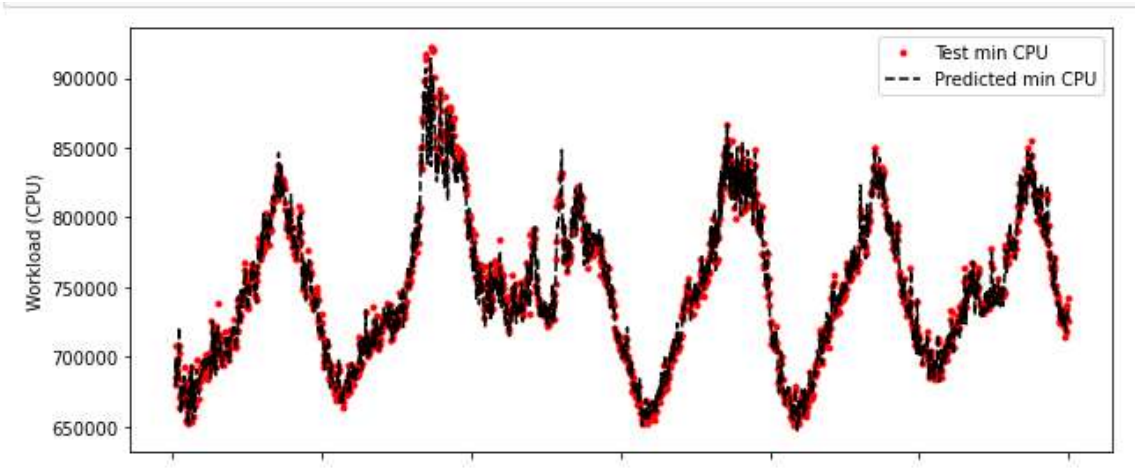


Figure 14:GRU prediction using Microsoft Azure maximum CPU dataset

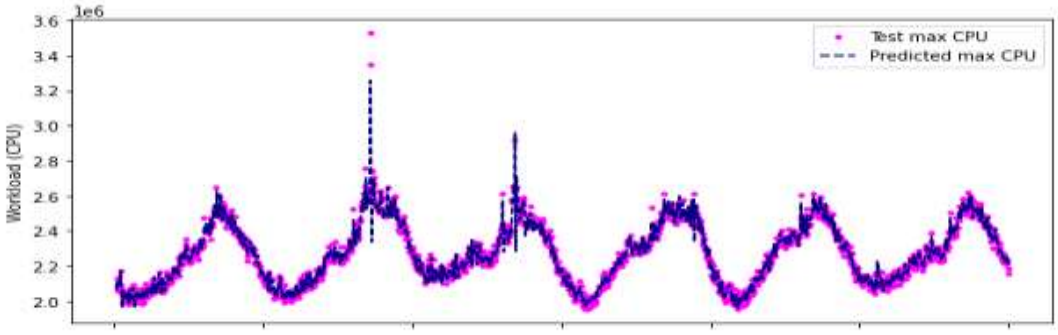


Figure 15:GRU prediction using Microsoft Azure maximum CPU dataset

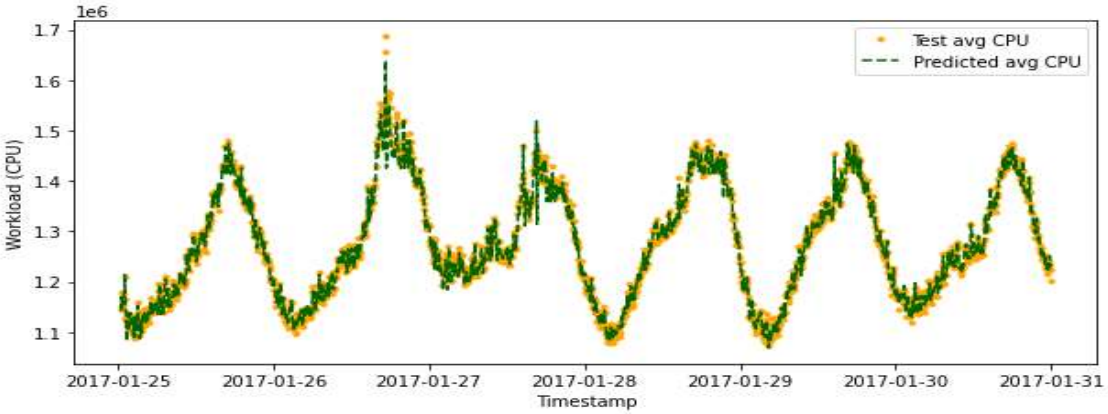


Figure 16:GRU prediction using Microsoft Azure maximum CPU dataset

5.11 Discussions

The experimental results demonstrated that the LSTM is preferable to the stack LSTM and the GRU for estimating the resources in the docker container. In Table 5, the LSTM prediction model was compared to the stack LSTM prediction model. It is demonstrated that the LSTM has a higher prediction accuracy than the stacked LSTM(Imdoukh et al., 2020d).

In addition, in Tables 5 and 6, the LSTM is compared to the GRU. It is demonstrated that the LSTM outperforms the GRU in prediction accuracy. As shown in Table 5, LSTM outperforms GRU in terms of overall prediction accuracy. The elastic speedup results in Table 6 also show that the LSTM auto-scaler has a higher autoscaling gain than the GRU auto-scaler in both step predictions.

This work's proposed architecture is based on existing docker components. This architecture, however, can be implemented in other container-based platforms, such as Docker Swarm for orchestration, and other container engines such as OpenCV and LXC(Bentaleb et al., 2022a). This is because the LSTM predicting service and the adaption manager service can analyze and process monitored data (CPU and memory) to calculate appropriate scaling decisions and are independent not only of container virtualization technology but also of cloud providers. Furthermore, we can modify and update the LSTM to predict more container-based resource metrics such as CPU utilization and memory usage to make more accurate scaling decisions.

Table 6: Comparison between LSTM stack LSTM and GRU

Model Type	**LSTM**	**Stack LSTM**	**GRU**
RMSE	28232.94	382759.04	**27488.19**
MAE	**129.709539**	570.196129	27488.19
MAPE	129.709539	0.217560	**0.010602**
R^2	**0. 999999**	0.999998	0.99998

Bold values indicate the best results

The results, however, clearly show that the LSTM model outperforms other prediction models for future workload patterns (GRU and Stack LSTM).

CONCLUSION AND FUTURE WORK

Conclusion

Containers are the new packaging and deployment trend for cloud applications based on micro-services. Containers' widespread adoption and use as the foundation technology for cloud systems necessitates skilled auto-scaling methods that automatically and timely adjust container provisioning based on workload demands. In this thesis, we proposed a system architecture for Docker containerized applications with an auto-scaler based on a Deep Learning model. The hybrid approach to elasticity uses both horizontal and vertical elasticity principles so, they proposed an algorithm that works based on the CPU usage of the container. Elastic cloud dynamically increases and decreases the number of CPU core and Container replicas. The monitoring mechanisms, time series database, prediction model, and decision mechanism of the auto-scaler were introduced. The prediction model was a simple method based on the LSTM model that learns from historical time series (request statistics) to predict future workload accurately. Experiment results show that the hybrid model significantly improves end-user QoS and performance, lowers customer expenses, and improves resource utilization. This study implements ARIMA, GRU, and stackLSTM as benchmarks. Furthermore, the experimental results show that, when compared to traditional ARIMA models, the proposed method LSTM achieves the best elasticity speed of 1.529, while GRU elasticity speed is 1.17 and stackLSTM is 1.182.

Future Work

In the future, we intend to include more metrics (i.e., adding and removing CPU, CPU time, memory, and disk I/O resources to the same container instance), and/or integrating both horizontal and vertical scaling is another research direction worth exploring further.

***

SPECIAL ACKNOWLEDGMENT

This research was sponsored by Adama Science and Technology University with a grant number is:

ASTU/SM-R/483/22

Adama, Ethiopia

REFERENCES

Abbasi, M., Shahraki, A., Jalil Piran, M., & Taherkordi, A. (2021). Deep Reinforcement Learning for QoS provisioning at the MAC layer: A Survey. In Engineering Applications of Artificial Intelligence (Vol. 102). Elsevier Ltd. https://doi.org/10.1016/j.engappai.2021.104234 Aggarwal, C. C. (n.d.). Neural Networks and Deep Learning. www.dbooks.org

Al-Asaly, M. S., Bencherif, M. A., Alsanad, A., & Hassan, M. M. (2021). A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment. Neural Computing and Applications. https://doi.org/10.1007/s00521-021-06665-5

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges. IEEE Transactions on Services Computing, 11(2), 430–447. https://doi.org/10.1109/TSC.2017.2711009

Al-dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P., Vertical, A., Al-dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2017). Autonomic Vertical Elasticity of Docker Containers with To cite this version : HAL Id : Hal-01522940 Autonomic Vertical Elasticity of Docker Containers with E LASTIC D OCKER.

Al-Dhuraibi, Y., Zalila, F., Djarallah, N., & Merle, P. (2018). Coordinating vertical elasticity of both containers and virtual machines. CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science, 2018-Janua, 322–329. https://doi.org/10.5220/0006652403220329

Bankole, A. A., & Ajila, S. A. (2013). Cloud client prediction models for cloud resource provisioning in a multitier web application environment. In Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013 (Issue April). https://doi.org/10.1109/SOSE.2013.40

Bentaleb, O., Belloum, A. S. Z., Sebaa, A., & El-Maouhab, A. (2021). Containerization technologies: taxonomies, applications, and challenges. In Journal of Supercomputing (Issue 0123456789). Springer US. https://doi.org/10.1007/s11227-021-03914-1

Bentaleb, O., Belloum, A. S. Z., Sebaa, A., & El-Maouhab, A. (2022a). Containerization technologies: taxonomies, applications, and challenges. Journal of Supercomputing, 78(1), 1144–1181. https://doi.org/10.1007/s11227-021-03914-1

Bentaleb, O., Belloum, A. S. Z., Sebaa, A., & El-Maouhab, A. (2022b). Containerization technologies: taxonomies, applications, and challenges. Journal of Supercomputing, 78(1), 1144–1181. https://doi.org/10.1007/s11227-021-03914-1

Bentaleb, O., Belloum, A. S. Z., Sebaa, A., & El-Maouhab, A. (2022c). Containerization technologies: taxonomies, applications, and challenges. Journal of Supercomputing, 78(1), 1144–1181. https://doi.org/10.1007/s11227-021-03914-1

Bhardwaj, A., & Krishna, C. R. (2021). Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey. Arabian Journal for Science and Engineering, 46(9), 8585–8601. https://doi.org/10.1007/s13369-021-05553-3

Buyya, R., & Vazhkudai, S. (2001). Compute Power Market: Towards a market-oriented grid. Proceedings - 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid 2001, 574–581. https://doi.org/10.1109/CCGRID.2001.923245

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6), 599–616. https://doi.org/10.1016/j.future.2008.12.001

Duc, T. le. (2019). Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey. https://doi.org/0000001.0000001

Fenila Naomi, J., & Roobini, S. (2019). A Hybrid Auto Scaling for Cloud Applications Using an Efficient Predictive Technique in Private Cloud model. Indian Journal of Science and Technology, 12(8), 974–6846. https://doi.org/10.17485/ijst/2019/v12i8/141807

Gao, J., Wang, H., & Shen, H. (n.d.). Machine Learning Based Workload Prediction in Cloud Computing.

Goli, A., Mahmoudi, N., Khazaei, H., & Ardakanian, O. (2021). A Holistic Machine Learning-based Autoscaling Approach for Microservice Applications. 190–198. https://doi.org/10.5220/0010407701900198

Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2020a). Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 32(13), 9745–9760. https://doi.org/10.1007/s00521-019-04507-z

Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2020b). Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 32(13), 9745–9760. https://doi.org/10.1007/s00521-019-04507-z

Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2020c). Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 32(13), 9745–9760. https://doi.org/10.1007/s00521-019-04507-z

Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2020d). Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 32(13), 9745–9760. https://doi.org/10.1007/s00521-019-04507-z

Irufaan, A. (2021). Microservice dynamic resource provision for small and medium-sized enterprises. 5(1).

Jafarnejad Ghomi, E., Rahmani, A. M., & Qader, N. N. (2019). Applying queue theory for modeling of cloud computing: A systematic review. Concurrency and Computation: Practice and Experience, 31(17). https://doi.org/10.1002/cpe.5186

Janardhanan, D., & Barrett, E. (2018a). CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models. 2017 12th International Conference for Internet Technology and Secured Transactions, ICITST 2017, 55–60. https://doi.org/10.23919/ICITST.2017.8356346

Janardhanan, D., & Barrett, E. (2018b). CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models. 2017 12th International Conference for Internet Technology and Secured Transactions, ICITST 2017, 55–60. https://doi.org/10.23919/ICITST.2017.8356346

Maenhaut, P. J., Volckaert, B., Ongenae, V., & de Turck, F. (2020). Resource Management in a Containerized Cloud: Status and Challenges. Journal of Network and Systems Management, 28(2), 197–246. https://doi.org/10.1007/s10922-019-09504-0

Mostofi, M. (2021). Auto-Scaling Containerized Microservice Applications. http://hdl.handle.net/1880/113888

Ndamlabin Mboula, J. E., Kamla, V. C., & Tayou Djamégni, C. (2021). Dynamic provisioning with a structure-inspired selection and limitation of VMs based cost-time efficient workflow scheduling in the cloud. Cluster Computing, 24(3), 2697–2721. https://doi.org/10.1007/s10586-021-03289-1

Nikravesh, A. Y., Ajila, S. A., & Lung, C. H. (2017). An autonomic prediction suite for cloud resource provisioning. Journal of Cloud Computing, 6(1). https://doi.org/10.1186/s13677-017-0073-4

Rossi, F., Nardelli, M., & Cardellini, V. (2019). Horizontal and vertical scaling of container-based applications using reinforcement learning. IEEE International Conference on Cloud Computing, CLOUD, 2019-July, 329–338. https://doi.org/10.1109/CLOUD.2019.00061

Salmanian, Z., Izadkhah, H., & Isazadeh, A. (2020). Auto-Scale Resource Provisioning in IaaS Clouds. The Computer Journal, 00(0). https://doi.org/10.1093/comjnl/bxaa030

Shahidinejad, A., Ghobaei-Arani, M., & Masdari, M. (2021). Resource provisioning using workload clustering in cloud computing environment: a hybrid approach. Cluster Computing, 24(1), 319–342. https://doi.org/10.1007/s10586-020-03107-0

Sharma, V. (n.d.-a). Dynamic Resource Management Schemes for Containerized Deep Learning Applications.

Sharma, V. (n.d.-b). Dynamic Resource Management Schemes for Containerized Deep Learning Applications.

Struhár, V., Behnam, M., Ashjaei, M., & Papadopoulos, A. v. (2020). Real-time containers: A survey. OpenAccess Series in Informatics, 80. https://doi.org/10.4230/OASIcs.Fog-IoT.2020.7

Taherizadeh, S., & Stankovski, V. (2019). Dynamic multi-level auto-scaling rules for containerized applications. Computer Journal, 62(2), 174–197. https://doi.org/10.1093/comjnl/bxy043

Tang, X., Liu, Q., Dong, Y., Han, J., & Zhang, Z. (2018). Fisher: An Efficient Container Load Prediction Model with Deep Neural Network in Clouds. 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing& Communications (ISPA/IUCC/BDCloud/ Social Com/ Sustain Com),199–206. https://doi.org/10.1109/BDCloud.2018.00041

Tao, Y., Wang, X., & Xu, X. (2018a). Containerized resource provisioning framework for multimedia big data application. Multimedia Tools and Applications, 77(9), 11439–11457. https://doi.org/10.1007/s11042-017-5366-6

Tao, Y., Wang, X., & Xu, X. (2018b). Containerized resource provisioning framework for multimedia big data application. Multimedia Tools and Applications, 77(9), 11439–11457. https://doi.org/10.1007/s11042-017-5366-6

Vinothiyalakshmi, P., & Anitha, R. (2021). Efficient dynamic resource provisioning based on credibility in cloud computing. Wireless Networks, 27(3), 2217–2229. https://doi.org/10.1007/s11276-021-02558-6

Wang, T. (2021). Predictive vertical CPU autoscaling in Kubernetes based on time-series forecasting with Holt-Winters exponential smoothing and long short-term memory.

Xu, M., Song, C., Wu, H., Gill, S. S., Ye, K., & Xu, C. (2022). esDNN: Deep Neural Network based Multivariate Workload Prediction in Cloud Computing Environments. ACM Transactions on Internet Technology. https://doi.org/10.1145/3524114

Yadav, M. P., Akarte, H. A., & Yadav, D. K. (2020). Container Elasticity: Based on Response Time using Docker. Recent Advances in Computer Science and Communications, 15(5), 773–785. https://doi.org/10.2174/2666255813999201012192010

Yadav, M. P., Pal, N., & Yadav, D. K. (2021a). Resource provisioning for containerized applications. Cluster Computing, 24(4), 2819–2840. https://doi.org/10.1007/s10586-021-03293-5

Yadav, M. P., Pal, N., & Yadav, D. K. (2021b). Resource provisioning for containerized applications. Cluster Computing, 24(4), 2819–2840. https://doi.org/10.1007/s10586-021-03293-5

Yadav, M. P., Pal, N., & Yadav, D. K. (2021c). Resource provisioning for containerized applications. Cluster Computing, 24(4), 2819–2840. https://doi.org/10.1007/s10586-021-03293-5

Yadav, M. P., Rohit, & Yadav, D. K. (2021a). Maintaining container sustainability through machine learning. Cluster Computing, 24(4), 3725–3750. https://doi.org/10.1007/s10586-021-03359-4

Yadav, M. P., Rohit, & Yadav, D. K. (2021b). Resource Provisioning Through Machine Learning in Cloud Services. Arabian Journal for Science and Engineering. https://doi.org/10.1007/s13369-021-05864-5

Yadav, M. P., Rohit, & Yadav, D. K. (2021c). Maintaining container sustainability through machine learning. Cluster Computing, 24(4), 3725–3750. https://doi.org/10.1007/s10586-021-03359-4

Yadav, M. P., Rohit, & Yadav, D. K. (2022). Resource Provisioning Through Machine Learning in Cloud Services. Arabian Journal for Science and Engineering, 47(2), 1483–1505. https://doi.org/10.1007/s13369-021-05864-5

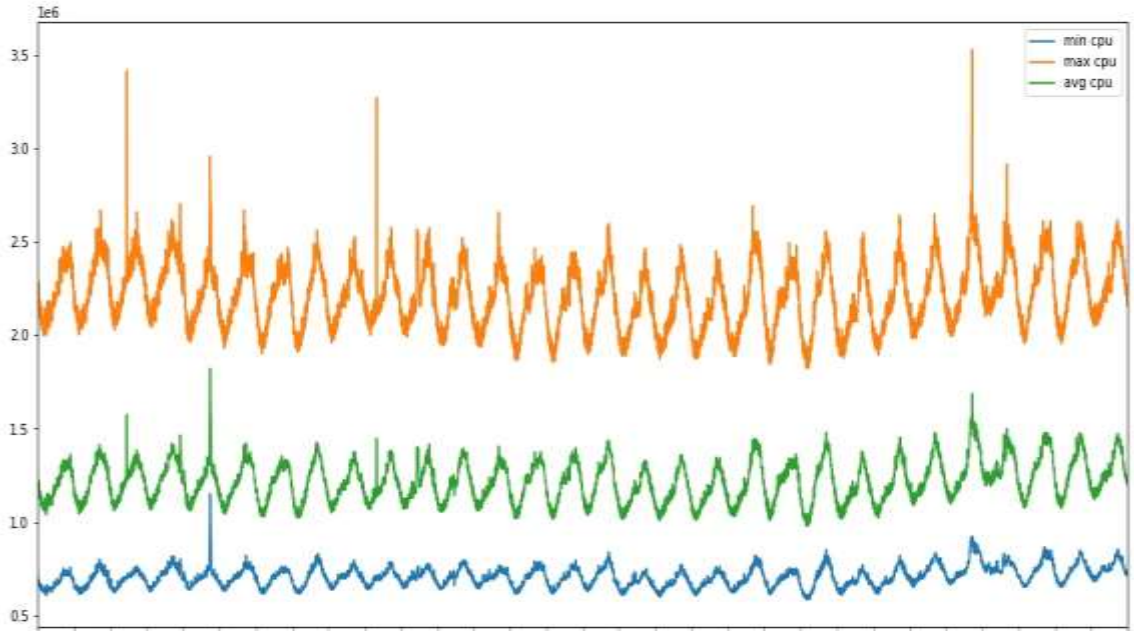
Zhong, Z., Xu, M., Rodriguez, M. A., Xu, C., & Buyya, R. (2022). Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions. ACM Computing Surveys. https://doi.org/10.1145/3510415

APPENDIX

Appendix A: Sample Code for Loading Dataset and Plot the Observation

```
#------------------------------------------------------------
# LOAD THE DATASET AND PLOT THE OBSERVATIONS
#------------------------------------------------------------

df = pd.read_csv("C:\\Users\\Administrator\\CPU\\azure.csv")
df['timestamp'] =  pd.to_datetime(df['timestamp'])
df = df.set_index('timestamp')
df.head()
df.plot(figsize=(16, 8))
plt.show()
```



Appendix B: Sample Code Train-Test Split, Scaling and LSTM Model

```
#------------------------------------------------------------
# CREATE TRAIN-TEST SPLIT (80:20)
#------------------------------------------------------------
TRAIN_LENGTH = round(len(df)*0.8)
TEST_LENGTH = len(df) - TRAIN_LENGTH
train = df.iloc[0:TRAIN_LENGTH]
test = df[TRAIN_LENGTH : ]
```

```
#------------------------------------------------------------
# SCALING VALUES
#------------------------------------------------------------
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = pd.DataFrame(scaler.fit_transform(train), columns=df.columns)
test_scaled = pd.DataFrame(scaler.transform(test), columns=df.columns)
```

```
#------------------------------------------------------------
# GENERATOR TO DATA TO FEED INTO MODEL
#------------------------------------------------------------
def train_generator(dataset, n_lags=1):
    dataX, dataY = [], []
    for i in range(len(dataset)- n_lags -1):
        a = dataset.iloc[i:(i+n_lags)].to_numpy()
        dataX.append(a)
        dataY.append(dataset.iloc[i + n_lags].to_numpy())
    return (np.array(dataX), np.array(dataY))
```

```
#------------------------------------------------------------
# DEFINING INPUTS AND EXPECTED OBSERVATIONS
#------------------------------------------------------------
TIME_STEPS = 5
X_train, y_train = train_generator(train_scaled, n_lags = TIME_STEPS)
X_test_scaled, y_test_scaled = train_generator(test_scaled, n_lags = TIME_STEPS)
X_test, y_test = train_generator(test, n_lags = TIME_STEPS)
```

```
model = keras.models.Sequential()
model.add(keras.layers.LSTM(128, input_shape=(trainX.shape[1], trainX.shape[2])))
model.add(keras.layers.Dense(3))
adamOpt = keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, decay=0.0, amsgrad=False)
model.compile(loss='mean_squared_error', optimizer=adamOpt, metrics=['mae'])
history = model.fit(trainX, trainY, validation_split=0.25,epochs=200, batch_size=4, verbose=2)
model.summary()
```

Appendix C: Sample Code for Defining Callbacks and Prediction Workload

```
#------------------------------------------------------------
# DEFINING CALLBACKS
#------------------------------------------------------------
es = tf.keras.callbacks.EarlyStopping( monitor='val_loss', patience=8, verbose=1, restore_best_weights=True)
lr_red = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4, verbose=1, min_lr=0.0000001,)
```

```
callbacks = [es , lr_red]
history = model.fit(X_train, y_train,
                    epochs=200,
                    validation_split=0.25,
                    batch_size=4,
                    verbose=1,
                    shuffle=False,
                    callbacks = callbacks)

#------------------------------------------------------------
# I TRAINED MY MODEL ON BATCHSIZES - 64,128,256
#------------------------------------------------------------
```

```
index=df.index
TestY= pd.DataFrame(testY,columns=['min_cpu','max_cpu','avg_cpu'])
PredY=pd.DataFrame(testPredict,columns=['min_cpu','max_cpu','avg_cpu'])

x=index[-1722:]
fig, axs = plt.subplots(3,figsize=(10,15))

axs[0].plot(x,TestY.min_cpu,'.',label='Test min CPU',color='red')
axs[0].plot(x,PredY.min_cpu,'--',label='Predicted min CPU',color='black')
axs[0].legend()
axs[1].plot(x,TestY.max_cpu,'.',label='Test max CPU',color='magenta')
axs[1].plot(x,PredY.max_cpu,'--',label='Predicted max CPU',color='navy')
axs[1].legend()
axs[2].plot(x,TestY.avg_cpu,'.',label='Test avg CPU',color='orange')
axs[2].plot(x,PredY.avg_cpu,'--',label='Predicted avg CPU',color='darkgreen')
axs[2].legend()
for ax in axs.flat:
    ax.set(xlabel='Timestamp', ylabel='Workload (CPU)',autoscale_on=True)
for ax in axs:
    ax.label_outer()
#fig.suptitle('Prediction of Workload on Azure cloud at a particular timestamp',fontsize=20)
plt.savefig("C:\\Users\\Administrator\\PDRP\\LSTM output.png", dpi = 300)
plt.show()
```

```
trainScore = math.sqrt(mean_squared_error(trainY[:], trainPredict[:]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[:], testPredict[:]))
print('Test Score: %.2f RMSE' % (testScore))

from sklearn.metrics import mean_absolute_error
trainScore = (mean_absolute_error(trainY[:], trainPredict[:]))
print('Train Score: %f MAE' % (trainScore))
testScore = math.sqrt(mean_absolute_error(testY[:], testPredict[:]))
print('Test Score: %f MAE' % (testScore))

trainScore2 = np.mean(np.abs(trainPredict - trainY)/np.abs(trainY))
print('Train Score: %f MAPE' % (trainScore2))
testScore2 = np.mean(np.abs(testPredict - testY)/np.abs(testY))
print('Test Score: %f MAPE' % (testScore2))

trainScore3 = np.corrcoef(trainPredict, trainY)[0,1]
print('Train Score: %f R2' % (trainScore3))
testScore2 = np.corrcoef(testPredict, testY)[0,1]
print('Test Score: %f R2' % (testScore2))
```