An Optimized Approach for Energy Consumption of Smart Devices in Fog Computing using Computational Intelligence Techniques

A Thesis

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

Doctor of Philosophy

in

Computer Science and Engineering Department

submitted by

Shabnam Bawa

(Reg no: 901803004)

Under the Supervision of

Dr. Prashant Singh Rana

Associate Professor, TIET

Dr. Rajkumar Tekchandani

Assistant Professor,TIET

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Thapar Institute of Engineering and Technology, Patiala, Punjab - 147004, India

October 2024

Certificate

I hereby certify that the work, which is being presented in the thesis, entitled "An Optimized Approach for Energy Consumption of Smart Devices in Fog Computing using Computational Intelligence Techniques" partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy and submitted to the institution is an authentic record of my work carried out under the supervision of Dr. Prashant Singh Rana and Dr. RajKumar Tekchandani. I have cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

The matter presented in this thesis has not been submitted either in-part or full to any other University/Institute for the award of any other degree.

(Shabnam Bawa)
Registration No. 901803004

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

Verified by:

(Dr. Prashant Singh Rana)
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala, Punjab, India.

(Dr. Rajkumar Tekchandani)
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala, Punjab, India.

.....dedicated to all young generation researchers

Abstract

To address the frequent overloading of fog nodes due to the increasing demand for IoT applications, an ensemble approach was employed to classify host load status into underloaded, balanced and overloaded categories. This work introduces an innovative reliability framework that encompasses multiple implementation phases. The process begins with the generation of virtual machines via the command line using various random settings. Various parameters such as CPU utilization, number of CPU cores, RAM, memory allocation, memory availability, disk I/O, and network I/O were analyzed to better understand host workload. Three case studies with varying numbers of virtual machines (VMs) were conducted on two different platforms for load prediction.

A total of ten machine learning models were employed to construct an ensemble model, which ultimately yielded optimal and accurate results for classifying host load. All models are evaluated for their precision, recall, and accuracy. Various pre-processing techniques such as normalization, transformation, principal component analysis (PCA), outlier removal are applied on the generated dataset and various models are compared.

It was observed that applying normalization to a dataset improved the performance of the models. Four models—Random Forest (RF), AdaBoost (AB), Gradient Boost (GB), and Decision Tree (DT)— performed equally well across all three case studies with normalized datasets. However, our proposed ensemble model performed marginally better than these individual models and it achieved nearly 82% accuracy in correctly classifying host load.

As the next major revolution in cloud and fog computing environment, container migration and containerization have emerged as key advancement. Fog computing and mobile edge cloud necessitate the transfer of containers from overloaded hosts to new hosts to ensure adequate resources for executing consumer applications at the network edges. Despite the growing popularity of containers, algorithms to manage the excessive energy consumption of hosts have not been thoroughly investigated. Moreover, optimizing the energy consumption efficiency of hosts remains a critical and challenging task.

In order to address the critical issue of reducing energy consumption in fog computing environment, the study moved beyond traditional virtualization techniques, which have a high computational overhead and are less suitable for fog devices. Containers, known for their efficiency in encapsulating fog services, were used instead. A container selection algorithm was introduced to identify containers for migration when a host becomes

overloaded.

Further, an energy-efficient container migration strategy was implemented using a dynamic inertia weight-based particle swarm optimization (DIWPSO) algorithm. This strategy aimed to balance the load and reduce energy consumption by migrating containers from overloaded hosts. Experimental results demonstrated that the DIWPSO algorithm significantly reduced energy consumption by 10.89% and achieved load balancing at a lower migration cost compared to traditional meta-heuristic solutions such as PSO, ABC, and E-ABC.

Additionally, The study developed a multivariate time series ensemble model for load prediction on hosts, utilizing anomaly detection techniques to forecast CPU utilization in the near future. Based on these predictions, resource utilization for container management was forecasted, determining the number of hosts needed to support the load of running containers.

Anomaly detection techniques were employed to reduce redundancy in generated data and address inconsistencies in load prediction due to the large volume of data. A predictive model with variable load patterns can better estimate future resource needs, which is crucial for capacity planning, meeting service-level goals, and achieving energy efficiency.

Various time series-based models were used for workload prediction, and the top three models were selected based on their TOPSIS scores to develop the ensemble model. To ensure the efficiency of the proposed model, Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and accuracy were evaluated and compared with other existing state-of-the-art models. The results demonstrated that the proposed ensemble model exhibited higher accuracy in workload prediction compared to current state-of-the-art models, achieving the lowest Mean Absolute Percentage Error (MAPE) and providing an accuracy of approximately 88%.

In conclusion, by integrating advanced machine-learning models for load prediction with an optimized container migration strategy, the study effectively enhanced resource utilization and energy efficiency in fog computing environment. This comprehensive approach successfully addressed the dual challenges of load balancing and energy consumption, providing a robust solution for managing the increasing demands of IoT applications.

Keywords: *IoT, Virtual Machine, Containers, Fog Computing, Machine Learning, Load Prediction, Energy consumption, Container Placement, PSO, Load Balance, Container Migration, Anomaly Detection*

Acknowledgement

Reaching this long-awaited day feels like a dream come true, made possible by the unwavering support of my family, friends, and my little one, whose patience was a constant source of strength. My deepest gratitude goes to my phenomenal research supervisors, Dr. Prashant Singh Rana and Dr. RajKumar Tekchandani. Their invaluable guidance, patience, encouragement, and insightful critiques were essential throughout this journey. They created an environment where my intellectual curiosity could thrive and provided the perfect mentorship to navigate my PhD studies. Their trust in my abilities made this achievement all the more rewarding.

Sincere thanks must also go to Dr. Shalini Batra, Head, CSED, TIET, for their continuous support by always being willing to help when needed. Their motivation and enthusiasm are contagious. I want to thank the members of my doctoral committee: Dr. Inderveer Channa, Dr. Anju Bala and Dr. Soumendu Jana. They generously gave their time to provide insightful comments on how to improve my work.

To my friend Jasleen, my deepest thanks for the stimulating discussions and invaluable suggestions that pushed my research forward. I especially want to thank my beloved friend, Akash, for her guidance. But even more importantly, thank you for the camaraderie and shared laughter that made the long hours fly by, and the faculty's encouraging words, always offered with a smile in passing, brightened my days. To all of you, my sincerest thanks.

This journey wouldn't have been possible without the unwavering love and support of my incredible family. I want to thank my wonderful brother Rajdeep for always keeping me going and being there for me when things got tough. My husband Gursewak's unwavering love, patience, and understanding were a constant source of strength that carried me through. To my amazing parents, my deepest gratitude for always believing in me and instilling the value of perseverance. Your encouragement was a constant fuel for my determination, even during the most challenging periods. Gurfurmaan, your pure joy and laughter served as a continual source of inspiration for me, reminding me of the beauty in life. You may not understand the significance of this day yet, but you should know that this achievement is a testament to our journey together.

Finally, a whisper of gratitude to the divine for the unwavering light that illuminated my path.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACS	Ant Colony System
ADT	Anomaly Detection Techniques
ANN	Artificial Neural Networks
ANNs	Artificial Neural Networks
ARDR	ARD Regression
Bi-LSTM	Bidirectional long short-term memory
BRR	Bayesian Ridge Regression
C3S	Concurrent Container Clusters Scheduling
CaaS	Container as a Service
CLA	Cellular Learning Automata
CM	Container Migration
CNNs	Convolutional Neural Networks
DL	Deep Learning
DLM	Dynamic Linear Models
DNN	Deep Neural Network
DVFS	Dynamic Voltage and Frequency Scaling
EN	ElasticNet
GA	Genetic Algorithm
GRU	Gated Recurrent Unit
HPC	High-Performance Computing
IF	Isolation Forest
KF	Kalmen Filtering
KNN	K-Nearest Neighbour
LightGBM	Light Gradient Boosting Machine
LR	Linear Regression
LSTM	Long Short Term Memory
LSTM-ED	Long Short-Term Memory Encoder–Decoder
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MCFP	Minimum Cost Flow Problem
MSE	Mean Square Error
MSE	Mean Square Error
MWh	Megawatts Hour
NN	Neural Network

OSVM	Open Support Vector Machine
PMs	Physical Machines
PSO	Particle Swarm Optimization
QoS	Quality of Service
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SDWF	Self Directed Workload Forecasting Method
SLA	Service Level Agreement
STL	Seasonal Decomposition of Time Series
SVR	Support Vector Regressor
TCN	Temporal Convolutional Network
VMP	Virtual Machine Placement
VMs	Virtual Machines

Chapter 1

Introduction

In a fog computing environment, optimizing performance and efficiency through effective energy consumption management and load balancing is crucial. Load balancing involves detecting overloaded, underloaded, and balanced hosts to distribute the load effectively, preventing any single host from becoming a bottleneck and ensuring optimal resource utilization. Host load prediction, through statistical analysis and machine learning, is used to dynamically allocate resources by forecasting computational and network demands. Energy consumption strategies, including energy-aware load balancing and virtualization, aim to minimize environmental impact, operational costs, and power consumption by enhancing the energy efficiency of hosts. Collectively, these practices ensure that a fog infrastructure is energy-efficient, scalable and reliable, capable of meeting a wide range of application requirements.

1.1 Background

Extensive research has been devoted to the prediction of workload on cloud hosts, employing a variety of techniques to improve model precision. Presently, workload prediction techniques encompass regression, statistical methods, machine learning algorithms, and recurrent neural networks (RNNs). Within the realm of cloud computing, machine learning approaches such as parametric, non-parametric, neural network-based, and deep learning models are employed for workload prediction. Traditionally, host load forecasting predominantly relies on univariate time series prediction models, utilizing historical data solely pertinent to the target variable, notably CPU utilization, for prediction. However, owing to inter-variable correlations, multivariate time series models demonstrate superior predictive capabilities compared to their univariate counterparts. Adding historical data, which is similar to univariate time series prediction, along with time series information from other host load factors has a significant effect on the results of host load prediction. These interdependencies underscore the importance of employing multivariate time series prediction techniques, which facilitate the extraction of additional attributes to augment prediction accuracy.

According to recent studies, container migration modifies the mapping between hosts

and containers to enable load balancing. Although moving containers from one host to another incurs less overhead than moving virtual machines, the cost should still be considered. Regularly moving containers between hosts can be quite costly. Therefore, to supply Container as a Service (CaaS), there is a tradeoff between load balance and migration cost. Furthermore, it is often assumed that resource requirements are known ahead of time, which is not the reality in most situations. To achieve host load balancing and minimize energy consumption while satisfying container resource needs for CaaS, a more sensible container management technique must be developed.

1.1.1 Load Prediction on Hosts in Fog Environment

Fog computing, a form of cloud computing, relocates computational resources to the network's periphery. This proximity is crucial for applications that necessitate immediate data processing, as it reduces latency and improves real-time processing capabilities. However, the efficient management of resources in fog environment presents distinct challenges. Host load prediction is one of the most critical components of resource management in fog computing.

1.1.1.1 Importance of Host Load Prediction

Host load prediction forecasts the computational and network demands on fog nodes. There are numerous reasons why precise load predictions are essential:

1. **Resource Optimization:** Guarantees that fog nodes are not overloaded or underutilized by ensuring that resources are allocated efficiently.
2. **Performance Enhancement:** It aids in maintaining the quality of service (QoS) by ensuring sufficient resources are available for applications running on fog nodes.
3. **Energy Efficiency:** This contributes to the reduction of energy consumption by dynamically adjusting resource utilization in accordance with the anticipated load.

1.1.1.2 Techniques for Host Load Prediction

In fog environment, a variety of techniques can be implemented to predict host load, such as:

1. **Statistical Methods**
    - **Time Series Analysis:** Utilizes historical data to forecast future loads on the host. Autoregressive Integrated Moving Average (ARIMA) and exponential filtering are prevalent methodologies.

- **Regression Models:** Historical data can be utilized to forecast future demand using linear and nonlinear regression models.

2. **Machine Learning Algorithms**
   - **Supervised Learning:** Algorithms such as Support Vector Machines (SVM), Decision Trees, and Random Forests are trained on historical data to predict future demand.
   - **Neural Networks:** Long short-term memory (LSTM) networks, recurrent neural networks (RNNs), and feed forward neural networks are capable of detecting intricate patterns in load data to make precise predictions.
   - **Deep Learning:** Deep neural networks (DNNs) and convolutional neural networks (CNNs) can model intricate relationships in the data.
3. **Hybrid Methods**
   The accuracy of predictions can be enhanced by combining multiple methods to capitalise on the strengths of each. For instance, the integration of neural networks and time series analysis.

1.1.1.3 Challenges of Host Load Prediction

In fog environment, host load prediction faces several challenges:

1. **Data Variability:** Accurate prediction is difficult in fog environment due to the high variability and dependence on a variety of load data factors.
2. **Real-time Requirements:** Real-time predictions require efficient algorithms and computational resources for effectiveness.
3. **Resource Constraints:** In comparison to cloud servers, fog nodes frequently possess restricted computational and storage capabilities, which necessitate the use of lightweight prediction models.
4. **Heterogeneous environment:** The heterogeneity of fog environment, characterised by nodes with varying capabilities and responsibilities, frequently complicates prediction efforts.

1.1.1.4 Applications of Host Load Prediction

In a fog environment, accurate host load prediction can benefit a variety of applications.

1. **Load Balancing:** The process of distributing workload across fog nodes in a manner that prevents any single node from becoming a bottleneck.
2. **Energy Management:** The dynamic adjustment of the power states of fog nodes in accordance with the anticipated demand to reduce energy consumption.
3. **QoS Maintenance:** Guaranteeing that applications are provided with the requisite resources to satisfy their performance specifications.
4. **Scalability:** It facilitates the growth of fog infrastructure by foreseeing the need for extra resources.

Numerous studies have investigated workload prediction for cloud hosts, employing a variety of techniques to enhance model accuracy. Current methods primarily utilize regression, statistical approaches, machine learning, and recurrent neural networks (RNNs). In cloud computing, machine learning techniques for workload prediction encompass deep learning, neural network-based, parametric, and non-parametric methods. Parametric models include methods such as the Kalman filter and the Automatic Regression Moving Average Model (ARIMA) with its various extensions. Non-parametric approaches, such as support vector regression (SVR) and neural network (NN)-based methodologies, are also widely used.
In a fog environment, host load prediction is crucial for effective resource management. Employing statistical methods, machine learning algorithms, and hybrid approaches allows for accurate forecasting of computational and network demands. Despite challenges like data variability, real-time requirements, resource constraints, and heterogeneous environment, effective host load prediction in fog computing systems optimizes resource usage, enhances performance, improves energy efficiency, and maintains service quality.

1.1.2 Container Migration Between Hosts

Container migration is the process of transferring containers from one host to another within a computing environment. This procedure is indispensable for guaranteeing optimal resource utilization and efficacy across a network of hosts. Applications and their dependencies can be dynamically relocated to various hosts based on the current workload and resource availability using containers, which package them into isolated environment.

1.1.2.1 Advantages of Container Migration:

- **Load Balancing:** Migration prevents any single host from becoming a performance bottleneck by redistributing containers, thereby balancing the load across multiple

hosts. This guarantees the efficient utilization of resources and the preservation of a consistent level of service.

- **Fault Tolerance:** The transfer of load from underloaded or failing hosts to migrated containers can improve overall system reliability by enhancing system resilience.
- **Resource Optimization:** It allows for better allocation of resources, as containers can be moved to hosts with available capacity, thereby optimizing the use of computational, memory, and storage resources.

1.1.2.2 Challenges faced during container migration

- **Overhead:** Moving containers involves data transfer and reconfiguration, which can introduce latency and consume network bandwidth. The overhead associated with migration needs to be carefully managed to avoid negating the benefits of load balancing.
- **Consistency:** It is essential to ensure that container states and data are consistently replicated across hosts in order to prevent data loss or disruptions during migration.
- **Cost:** It is crucial to maintain a balance between the frequency of migrations and the associated expenditures, as frequent migrations can result in increased operational costs.

1.1.3 Energy Consumption Minimization

Minimizing energy consumption in computing environment is essential for minimizing environmental impact and operational costs. Strategies that are effective include the following:

1. **Dynamic Voltage and Frequency Scaling (DVFS):** DVFS optimizes processor power consumption in accordance with the current workload. By reducing the voltage and frequency during periods of low demand, DVFS reduces energy consumption without significantly affecting performance.
2. **Workload Consolidation:** The consolidation of numerous workload onto a small number of hosts can result in other hosts being placed in low-power or idle states. By utilising fewer, more efficiently utilized resources, this method reduces overall energy consumption.

3. **Energy-Efficient Hardware:** The utilization of hardware that is specifically engineered to be energy-efficient, such as low-power processors and energy-efficient cooling systems, contributes to the reduction of the overall energy footprint.

The challenges faced in minimizing energy consumption are as follows:

1. **Balancing Performance and Efficiency:** It can be difficult to minimize energy consumption while maintaining performance, particularly during peak loads or when resource demands are unpredictable.
2. **Monitoring and Management:** The continuous monitoring and management of energy usage is a complex and resource-intensive process that is necessary for the implementation of effective energy-saving strategies.

1.1.4 Host Load Balancing

Load balancing is the process of distributing responsibilities across multiple hosts to prevent any single host from becoming overloaded. This is essential for the preservation of system reliability and performance. The following are methods for effective load balancing:

1. **Round-Robin Distribution:** Tasks are distributed sequentially among the available hosts. Although it may not always account for variations in individual host capacity, this straightforward approach guarantees that each host receives an equitable share of the load.
2. **Least Connections:** The system assigns new assignments to the host that has the fewest active connections. This approach mitigates the risk of overloaded hosts that are already managing a substantial volume of connections.
3. **Dynamic Load Balancing:** Dynamic Load Balancing utilizes real-time data and algorithms to make decisions based on the current load conditions, resource availability, and application requirements. Although this method may be more intricate, it offers a more adaptive and responsive approach to load distribution.

Container migration, energy consumption minimization, and load balancing are interconnected strategies that play a crucial role in optimizing performance and efficiency in modern computing environment. While container migration facilitates load balancing and enhances fault tolerance, it must be carefully managed to minimize overhead and costs. Energy consumption minimization strategies, such as DVFS and workload consolidation, help reduce operational expenses and environmental impacts. Effective load balancing ensures that resources are utilized efficiently, maintaining system performance

and reliability. Balancing these strategies requires careful planning and management to achieve an optimal balance between performance, cost, and energy efficiency.

1.2 Research Gaps

1. Studies in the literature compare the performance of linear and nonlinear methods. For instance, Calheiros et al. shows that linear and statistical models work better than artificial neural networks (ANNs) [1]. On the other hand, Zhang et al. shows that ANNs work best when there are a lot of correlations between the variables that aren't related to each other [2]. Safi et al. have demonstrated that hybrid models outperform simple models in time series forecasting [3]. Combining the strengths of individual methods in a hybrid model reduces the risk of errors from using an unsuitable method, leading to more accurate outcomes. This underscores the importance of ensemble models that integrate the best methods to capture both linear and nonlinear dependencies.

2. Most current methods for forecasting host load use univariate time series models, relying solely on historical data for a single variable, typically CPU usage. However, multivariate time series models, which consider correlations between multiple variables, offer greater predictive accuracy. For instance, if CPU usage increases linearly from 0% to 20%, a univariate model might predict a similar increase to 40% and 60%. However, if memory usage concurrently rises from 10% to 80%, it is likely that memory depletion will lead to disk paging, reducing future CPU usage. Therefore, multivariate models, which incorporate historical data and related variables, improve prediction accuracy by capturing these dependencies [4].

3. Implementing load balancing in a fog environment optimizes resource utilization on fog nodes. Real-time migration of virtual machines (VMs) is useful for load balancing, allowing a running VM to migrate between physical systems without disconnecting applications. However, the slow initialization speed of VMs and increased VM quantity can reduce efficiency and hinder quality of service (QoS) for time-sensitive applications. Containers, a lightweight virtualization technique, address these VM limitations by using OS-level virtualization, sharing the host's kernel, and facilitating migration. Therefore, a well-planned container migration strategy is crucial for achieving load balancing and minimizing energy consumption in fog environment [5].

4. Existing solutions for container migration timing often rely on fixed thresholds or predictive methodologies such as machine learning and linear regression. Due

to the NP-hard nature of the container migration (CM) problem, heuristic algorithms have been proposed to approximate optimal solutions, but these can fall prey to local optimization [6]. Meta-heuristic algorithms inspired by biology, like the Ant Colony System (ACS), Genetic Algorithm, and Particle Swarm Optimization (PSO), can avoid local optima. A multi-objective Ant Colony algorithm was introduced to efficiently arrange virtual machines (VMs) in data centers by finding pareto-optimal solutions for resource waste and power consumption. However, most current approaches focus on container redeployment and do not take into account the migration costs and delays associated with fog computing [7].

5. The fog computing system is composed of numerous distributed nodes that are dispersed geographically. Concentration prevents excessive heat production, thereby requiring an additional cooling system. Despite the potential reduction in power consumption, the large number of nodes in the fog computing system will likely lead to a higher energy consumption compared to cloud computing. Consequently, a significant amount of effort should be devoted to the development and optimization of novel energy-saving protocols and architectures in the fog paradigm [8].

6. It is beneficial to predict the future processing demands of an IoT application. It is required to develop an efficient technique of machine learning in the EoT/fog-IoT infrastructure where the machine learning techniques can help to predict the future processing demands of an IoT application based upon the previously encountered behavior of the application [9].

7. The energy consumption of smart devices in the fog computing environment using the container as a service has not been fully investigated. It is required to improve the container placement algorithms and develop a model that decreases the energy consumption of smart devices in fog computing [10].

8. The focusing on multi-core CPU architectures, as well as consideration of multiple system resources, such as memory and network interface is required. These resources significantly contribute to the overall energy consumption [9].

1.3 Research Objectives

After analysis of existing research gaps, the following objectives are proposed:

1. To study and analyze the existing literature on the energy consumption of smart devices in the fog computing environment.

2. To design and develop a model for classification of hosts as overloaded, underloaded

and balanced using computational intelligence techniques.

3. To design and develop an efficient migration approach for optimization of energy consumption of smart devices in a fog environment.
4. To verify and validate the proposed approach.

1.4 Research Contribution

1. A multilevel ensemble model has been developed to classify hosts as overloaded, underloaded and balanced.
2. The dataset is generated by executing a different number of VMs in parallel on two platforms with different configurations. Various parameters like CPU usage, number of cores, RAM, memory allocated, disk I/O, memory available, and network I/O are extracted to generate the dataset.
3. To categorize hosts as overloaded underloaded, or balanced, we implemented a variety of supervised machine learning models. These models encompassed Random Forest, Light Gradient Boosting Machine, K-Neighbors Classifier, Ridge Classifier, Linear Discriminant Analysis, Decision Tree, Gradient Boost Classifier, Naive Bayes Classifier, Ada Boost Classifier and Extra Tree Classifier.
4. A multivariate time series ensemble technique has been developed to forecast the load of VMs.
5. Anomaly detection techniques like like iforest, KNN, and LOF are used for preprocessing of data along with various time-series based models like ARIMA, XgBoost, DLM, RNN LSTM, Prophet and ConvLSTM models for prediction of load on hosts.
6. A dynamic inertia weight-based PSO facilitates a container selection algorithm and migration mechanism for energy-aware container consolidation on hosts. This approach aims to balance loads, reduce energy consumption, and minimize the adverse effects of container migration.

1.5 Thesis Organization

The Thesis is organized as follows.

Chapter 1: Introduction
The chapter begins by providing an overview of the background behind the prediction of workload on hosts in a fog environment, as well as the optimization of

energy consumption and load balancing. This includes the significance of host load prediction, the various techniques and challenges associated with it, the process of container migration between hosts, the advantages and challenges encountered during this migration, the optimization of energy consumption and host load balancing, as well as the identification of research gaps and objectives. The chapter finally comes up with a thesis contribution and organization.

Chapter 2: Literature Survey

This chapter presents a detailed literature survey of load prediction on hosts in cloud and fog environment. The chapter also outlines the use of ARIMA, Prophet and B-LSTM models for time series data to predict workload. Load balancing and optimization of energy consumption, which includes various optimization techniques as well as container migration. We also discuss the literature on anomaly detection techniques for predicting host load. Machine learning techniques such as decision tree classifiers, random forest classifiers, gradient boosting, extra tree classifiers, etc., and time series-based workload prediction methods such as ARIMA, Prophet, RNN LSTM, Convolutional LSTM, XgBoost, and DLM are discussed.

Chapter 3: Research Methodology

This chapter provides an extensive review of a variety of software, algorithms, methods and requirements. The chapter defines the comprehensive methodology that was implemented to resolve the research problem.

Chapter 4: Multilevel Ensemble Model for Load Prediction on Hosts in Fog Computing Environment

In this chapter, an overview of load prediction and fog computing is provided. Various materials and methods used for dataset generation, including modeling and simulation, as well as data transformation, are discussed. The process of load prediction on physical machines (PM), service level agreement metrics and various machine learning methods used to predict the load on multiple hosts are explored. The chapter presents the methodology of the proposed multilevel ensemble model and includes various case studies. The chapter concludes with an analysis, comparison and discussion of the results, summarizing the key findings.

Chapter 5: Migration of Containers on the Basis of Load Prediction with Dynamic Inertia Weight based PSO Algorithm

This chapter provides an overview of container migration in a fog environment, focusing on optimizing energy consumption and load balancing on hosts. It describes the system model, which includes a load balancing model, container migration costs, and the problem definition for the proposed work. The chapter details

the dataset used and the various methods employed in the proposed container migration strategies. The chapter also includes the experimental setup, results, and discussion. Finally, the chapter concludes with a summary of the paper.

Chapter 6: Multivariate Time Series Ensemble Model for Load Prediction on Hosts using Anomaly Detection Techniques

This chapter provides an overview of time series-based workload prediction methods for hosts. The proposed work is presented, which includes an overview of performance modelling for usage-aware forecasting, dataset generation through modelling, various workload prediction methods, and anomaly detection techniques. The next section discusses the system model and its mathematical modeling. The proposed methodology is introduced next. The experimental analysis follows, encompassing the experimental setup, evaluation criteria, performance evaluations and TOPSIS score evaluations. Next, the results and case studies undergo analysis, comparison and discussion. Finally, the paper concludes with a summary.

Chapter 7: Conclusion and Future Directions

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

Chapter 2

Literature Survey

In the dynamic and resource-intensive environment of cloud and fog computing, efficient management of computational resources is essential for ensuring high performance, reliability and sustainability. This has driven extensive research into four critical areas: load prediction on hosts, container migration, load balancing and optimizing energy consumption. Each of these areas addresses specific challenges associated with managing distributed computing resources and aims to enhance overall system efficiency and effectiveness.

Accurate load prediction is a cornerstone of effective resource management in cloud and fog computing. Load prediction involves forecasting future resource demands, such as CPU, memory and network usage, to ensure that hosts are adequately prepared to handle upcoming workload. Traditional methods for load prediction have relied on statistical techniques and univariate time series models, which use historical data to predict future load. However, these methods often fall short in capturing complex patterns and inter-dependencies among various resource metrics. To address these limitations, recent research has focused on leveraging advanced machine learning algorithms, including regression models, support vector machines, neural networks and deep learning approaches.

These techniques offer improved accuracy by considering multivariate time series data, which capture correlation between different resource metrics. Additionally, hybrid models that combine statistical and machine learning methods have been proposed to further enhance prediction precision. The literature emphasizes the importance of real-time load prediction to enable proactive resource allocation and avoid performance bottleneck.

Container migration is a critical mechanism for maintaining optimal resource utilization and application performance in a distributed computing environment. Containers, which encapsulate applications and their dependencies, can be dynamically moved between hosts to balance the load and respond to changing resource demands. Migration techniques are categorized into several types, including live migration, cold migration, pre-copy migration, and post-copy migration. Research in this area focusses on optimizing migration techniques to reduce overhead, minimize downtime, and maintain data consistency, with

a particular emphasis on developing algorithms that balance the trade-offs between performance and migration costs.The literature explores various algorithms and frameworks designed to minimize migration costs while maintaining service continuity.

Effective load balancing ensures that workload are evenly distributed across multiple hosts, preventing any single host from becoming a performance bottleneck. Load balancing strategies are essential for optimizing resource utilization, enhancing system reliability, and providing a seamless user experience. Traditional load balancing methods include round-robin distribution and least connections, which are relatively simple but may not account for variations in host capacities and dynamic workload changes. Advanced load balancing techniques employ real-time data and sophisticated algorithms to adapt to changing conditions. These methods include dynamic load balancing algorithms that use predictive analytics to anticipate load fluctuations and redistribute tasks proactively. Additionally, machine learning and artificial intelligence are increasingly being used to develop more adaptive and intelligent load balancing solutions. The literature explores various heuristic and algorithmic approaches to improve the efficiency and responsiveness of load balancing mechanisms.

With the growing emphasis on sustainability, optimizing energy consumption in cloud and fog computing environment has become a critical area of research. Data centers and distributed computing systems consume significant amounts of energy, leading to high operational costs and environmental impact. Techniques for optimizing energy consumption include dynamic voltage and frequency scaling (DVFS), workload consolidation, and the use of energy-efficient hardware. The literature also explores the impact of energy optimization on operational costs and environmental sustainability.

In summary, this literature survey aims to provide a comprehensive overview of current research on load prediction, container migration, load balancing, and optimizing energy consumption in cloud and fog computing environment. By examining the latest advancements, methodologies and challenges in these areas, the survey seeks to identify emerging trends and highlight opportunities for future research. Effective management of these interrelated aspects is crucial for achieving optimal performance, reliability and sustainability in distributed computing systems.

2.1 Prediction of Load on Hosts

Several studies have examined the prediction of load for cloud hosts and implemented various methods to address the problem of workload prediction, thereby enhancing the accuracy of models. Present approaches for workload prediction primarily improve model

accuracy by employing techniques such as regression, statistics, machine learning and RNN-based methods. In cloud computing, workload prediction based machine learning techniques include parametric, non parametric, neural network and deep learning approaches. Parametric models use learning methods like the kalman filter [11] and the automatic regression moving average model (ARIMA) [1]. Support vector regression (SVR) and neural network (NN)-based methodologies are non parametric approaches.

For prediction of load on host, different time series-based approaches are suggested, such as ARIMA [12], linear regression [13], exponential averaging [14] and others.

Table 2.1 provides a summary of the comparative analysis of various approaches for load prediction in diverse computing paradigms. As per Table 2.1, load prediction is evaluated on cloud computing platform whereas our approach performs load prediction on nodes in fog computing environment. As cloud environment generate enormous amount of data volumes, so conventional computing methodologies, like distributed computing and cloud computing are unable to handle challenges such as reducing latency, response time and consumption of energy. Calheiros et al. [1] proposed an ARIMA-based load prediction model for cloud computing in order to accomplish predictive scaling for virtual machine instances. Tan et al. [15], proposed an aggregation method for prediction of traffic flow that incorporates ARIMA [1], moving average, exponential smoothing, and neural networks. The utilization of time series prediction methods is sub optimal in non linear scenarios due to the non linearity and linearity of the time series produced by varying resource loads. A large number of machine learning techniques, including recurrent neural networks (RNN), support vector machines (SVM) and artificial neural networks (ANN), are also employed for load prediction.

Janardhanan et al. [16] used Long Short Term Memory (LSTM) to forecast CPU utilization. LSTM exhibits superior performance to time series, according to some results.

Fan et al. [17] enhanced the performance of the Temporal Convolutional Network (TCN) across various time series prediction tasks. Kumar et al. [18] proposed a self-directed workload prediction method to capture prediction error trends by calculating deviations. By optimizing solutions for the prediction error feedback window, cluster size, and learning rate, its performance can be enhanced further.

Khan et al. [19] conducted a comprehensive study of numerous machine-learning algorithms, including linear regression, Regression with Ridges, ARD Regression, Elastic Net, along with deep learning techniques like the Gated Recurrent Unit.

Patel et al. [20] presented a workload forecast model utilizing neural networks and a self-adapting evolutionary algorithm.

Table 2.1: Comparative Analysis of Existing Approaches of Load Prediction

Paper	**Method**	**Computing Paradigm**	**Datasets**	**Metrics**	**Work done**
Calheiros et al.[1]	ARIMA Model	Cloud Computing	Real traces of user requests to web servers.	CPU Utilization	ARIMA model was used to predict future workload using actual Wikimedia Foundation web server request traces and evaluate its accuracy.
Gao et al. [21]	Machine Learning	Cloud Computing	Google cluster trace	CPU	A clustering-based workload prediction method is designed.
Kumar and Singh [22]	Artificial neural network	Cloud Computing Environment	NASA HTTP traces and Saskatchewan HTTP traces	SLA Violations	A paradigm for workload forecasting is presented that utilizes an artificial neural network and an algorithm for capable of self-adapt differential evolution.
Ramezani [23]	fuzzy workload prediction Algorithm	Cloud Computing Environment	Historical workload data- two datastores	VM CPU Usage	Developed a fuzzy work estimation method that monitors both previous and current VM's CPU consumption and forecasts physical machine utilization.
Yang et al. [24]	linear regression model	Cloud Computing	N/A	CPU Usage, SLA Violation	Utilizing a linear regression approach to forecast the workload, an automated scaling mechanism for scaling virtual assets in service clouds is proposed.
Qiu et al.[25]	Deep Belief Networks	Cloud Computing Environment	PlanetLab	CPU Utilization	A load forecasting method based on deep machine learning is proposed. Improved the predictability of the CPU utilization prediction when compared to current methods.
Jheng et al. [26]	Grey Forecasting model	Cloud Computing	four historical workload of PMs and VMs	Power Consumption	In data centers, the grey model is used to develop a method for workload prediction and to reduce the electrical power consumption of PMs.
Yu et al. [27]	multilayer perceptron neural network	Cloud Environment	Google Dataset	CPU Usage	An approach to workload forecast in the cloud that estimates the workload patterns of new tasks based on the statistical properties of a set of tasks.
Patel and Misra [20]	Deep learning Models	Cloud Computing	real workload traces of PlanetLab	CPU Usage	Predicts large-scale, highly data-driven systems for distributed decision making, such as hot spot mitigation, cold spot mitigation, benchmark violation and service level agreements violation.
Eramo et al. [28]	Convolutional Short-Term Memory neural network	Cloud Computing	N/A	QoS degradation cost	Using a straightforward VNFI reconfiguration and placement algorithm, the effectiveness of a LSTM neural network system for resource prediction and allocation is evaluated.
Caron et al. [29]	Pattern Matching	Cloud and Grid Computing	Traces of Animoto, LCG, Nordugrid and SHARCNET	Min prediction error, Max prediction error, Avg prediction error	It identify previous instances resembling the current short-term work history.
Roy et al. [30]	A model-forecasting algorithm	Cloud Computing	World Cup Soccer 1998 Workload	SLA Violation, reconfiguration cost	For resource automated scaling, a model-predictiv e method for workload forecasting has been devised and implemented.
Janardhanan et al. [16]	Time series forecasting	Cloud Computing	Google's cluster dataset	CPU Usage	The utilization of LSTM Networks to predict the CPU utilization of machines in data centres has demonstrated a substantial advancement in comparison to the ARIMA models.
Kumar et al. [18]	self directed workload forecasting method (SDWF)	Cloud Computing	six different real world data traces	quality of experience (QoE)	A self-directed workload forecasting method is proposed to enhance prediction quality and enhance cloud system service quality.
Proposed Work	Ensemble Model	Fog Computing Environment	Generated Real workload data	CPU Usage, Power Consumption	Developed a multilevel ensemble model to classify hosts based on workload prediction.

2.2 Time Series Based workload Prediction on Hosts

Gao et al. [21] conducted a comparison of the CPU utilization prediction performance of ARIMA, Support Vector Regression (SVR), Bayesian Ridge Regression (BRR) and LSTM. The authors examine the accuracy of BRR in forecasting the CPU utilization of Google cluster hosts in both one-time and three-time steps prior to the present. The authors arrive at the subsequent deductions: ARIMA performs poorly with pathological data, SVR struggles with large datasets, LSTM effectively models nonlinear patterns within the data, and BRR provided the greatest performance on the large, highly fluctuating Google cluster trace. Google cluster trace investigations show that their clustering-based workload prediction methods surpass other comparative methods and enhance CPU and memory prediction accuracy by 90%.

Table 2.2 summarizes the comparative analysis of different approaches for future load prediction on hosts. Chen et al. [31] proposed an ARIMA-based hybrid model for prediction of utilisation of hosts over the intermediate term. In order to enhance performance, the authors partition the non-stationary host utilization trace into constituents of the intrinsic mode function that are comparatively stable, along with a residual component that serves as an input to the ARIMA model. Their method is compared to the ARIMA model and EEMD-ARIMA method for time, cost, efficacy, and error. They found that their method is more cost-effective and suitable for short-term cloud host utilization prediction.

Janardhanan et al. [16] exposed the inadequacies of the conventional ARIMA model in representing the nonlinear dynamics of significantly varying cloud workload. To address this, they suggested an LSTM model that can predict the CPU utilization of Google cluster machines. LSTM provides more accurate predictions than ARIMA, which is better adapted for preserving short-term correlations within time series due to the memory cells that store temporal dependencies over extended time periods. It shows that the LSTM model had 17-23% predicting errors, compared to 37-42% for the ARIMA model.

Gupta et al. [32] proposed Bidirectional LSTM (B-LSTM) to predict CPU usage in Google Trace using multiple features. By learning input patterns in both the forward and backward directions using the reverse copy of the original input signals, the duplicated recurrent layer enables the LSTM to discover additional patterns.

Multivariate B-LSTM demonstrates superior learning of long-range patterns compared to fractional-differencencing-based PRESS, AGILE, ARIMA, and NARNN prediction methods for multi-step advance forecasts, as observed by the authors.

Table 2.2: Comparative Analysis of Existing State-of-the-Art Approaches for Future Load Prediction on Hosts

Paper	**Method**	**Computing Paradigm**	**Datasets**	**Metrics**	**Multi/ Uni-variate**	**Novelty**
Calheiros et al. [1]	ARIMA Model	Cloud Computing	Real traces of requests to web servers.	CPU	Univariate	Development of a SaaS provider-oriented cloud workload prediction module utilizing the ARIMA model.
Gao et al. [21]	Machine Learning	Cloud Computing	Google cluster trace	CPU	Univariate	A clustering based workload prediction method to improve prediction accuracy.
Kumar et al. [22]	Artificial neural network	Cloud Computing	NASA and Saskatchewan HTTP traces	SLA Violations	Univariate	A workload prediction model utilizing a neural network and a self-adaptive differential evolution algorithm.
Ramezani et al. [23]	Fuzzy workload prediction algorithm	Cloud Computing	Historical workload data-two datastores	VM CPU Usage	Univariate	Developed a fuzzy workload prediction method that monitors both historical and current VM CPU utilization.
Yang et al. [24]	linear regression model	Cloud Computing	N/A	CPU Usage, SLA Violation	Univariate	A unique service Utilizing a linear regression model to forecast the workload, cloud architecture is described.
Qiu. et al. [25]	Deep Belief Networks	Cloud Computing	PlanetLab	CPU	Univariate	A deep learning prediction model is developed incorporating a regression layer and a deep belief network comprised of multiple-layered restricted Boltzmann machines.
Jheng et al. [26]	Grey Forecasting model	Cloud Computing	Four historical workload of PMs and VMs	Power Consumption	Univariate	Grey Forecasting model is used to design a workload prediction method and it also reduces the power consumption of the PMs in data centers.
Yu et al. [27]	multilayer perceptron neural network	Cloud Computing	Google Dataset	CPU Usage	Univariate	Clustering and learning-based approach used to predict the workload by utilizing a trained neural network within its cluster.
Patel et al. [20]	Deep learning Models	Cloud Computing	workload traces of PlanetLab	CPU Usage	Univariate	Diverse approaches of deep learning for workload forecasting, were implemented.
Xu et al. [33]	esDNN Method	Cloud Computing	Alibaba and Google cloud	cost of service	Multivariate	An effective supervised learning-based Deep Neural Network method for cloud workload prediction is proposed.
Dang et al. [34]	Bi-LSTM	Cloud Computing	Real world workload dataset	SLA Violations	Multivariate	A multivariate deep learning model to estimate the cloud computing resource workload using bidirectional long short-term memory as the prediction model.
Ouhame et al. [35]	CNN-LSTM Model	Cloud Computing	Bitbrains dataset	CPU Usage, Network Usage	Multivariate	A convolutional neural network and long short-term memory model to forecast multivariate workload.
Shishira et al. [36]	WNNA+NBMOA	Cloud Computing	NASA, Worldcup98	HTTP Requests	Multivariate	A novel BeeM-NN framework combines Workload Neural Network Algorithm and Novel Bee Mutation Optimization Algorithm for cloud workload prediction.
Singh et al. [37]	EQNN Model	Cloud Computing	Google, PlanetLab	CPU Usage, Memory, Disk	Univariate	Workload prediction using an Evolutionary Quantum Neural Network in cloud.
Proposed Work	Ensemble Model	Fog Computing	Generated Real workload data	CPU Usage	Multivariate	Developed a multivariate multilevel ensemble model for workload prediction.

Huang et al. [38] utilized a Nginx web server and a MySQL database server for their experiments. Kumar et al. [39] employ three distinct data sets—HTTP traces from a NASA server, a Calgary server and a Saskatchewan server—to generate LSTM predictions for HTTP requests and encounter prediction errors. Their empirical findings demonstrate that the suggested approach successfully generated precise predictions, as evidenced by a mean squared error reduction of 3.17 x 10^{-3}.

Tran et al. [40] transform the highly volatile Google Workload Trace into a multivariate fuzzy time series by employing fuzzy techniques. Gao et al. [41] employed stacked BLSTM to forecast task failure with the dual purpose of preventing resource wastage caused by failure recoveries and preserving service availability and dependability. The performance of the model is assessed using Google Trace and exhibits superior accuracy in comparison to Hidden Semi-Markov Models, SVR, RNN, and LSTM. Trace-driven experiments show that their approach outperforms previous prediction methods, predicting task failures with 93% accuracy and job failures with 87%.

Karim et al. [42] presented a hybrid deep learning model that forecasts the multi-step CPU utilization of virtual machines by combining the nonlinear pattern learning capabilities of LSTM, GRU, and BLSTM with patterns extracted by One Dimensional CNN. As a consequence of its ability to discover temporal patterns in both past and future data, B-LSTM outperforms ARIMA, LSTM, GRU and BLSTM.

Calheiros et al. [1] presented cloud workload prediction module based on the ARIMA model can be used by SaaS providers. The simulation showed that their model had an average accuracy of 91%, resulting in effective utilization of resources with minimal QoS impact.

Khan et al. [19] investigated a variety of ML algorithms, including Linear Regression (LR), Ridge Regression (RR), ARD Regression (ARDR), ElasticNet (EN) and Gated Recurrent Unit (GRU). Four different clustering algorithms were proposed for energy state estimation, including semi-supervised affinity propagation based on transfer learning (TSSAP), Cellular Learning Automata (CLA) based on transfer learning (TCLA), K-Means based on transfer learning (TK-Means) and P-teda based on transfer learning (TP-teda). The TSSAP outperformed previous clustering algorithms in VM class identification, attaining 87.48% and 53.80% using conventional criteria like micro-precision for the given workload. Patel et al. [20] presented a workload prediction model that uses a neural network and a self-adaptive differential evolution algorithm.

2.3 Load Balancing and Optimization of Energy Consumption

Load balancing on the hosts can enhance system stability, resource utilization, and service response time and minimize energy consumption in a fog environment. Different studies have proposed a number of algorithms for container positioning and migration to optimize host energy consumption and achieve load balancing. Table 2.3 summarizes the comparative analysis of different approaches for load balancing and energy conservation using containers. Zhang et al. [43] investigated the container placement problem and proposed a genetic algorithm to optimize energy efficiency. However, it ignored the relocation and migration that occurred in real time and only considered the static situation. In their study, Wu et al. [44] introduced a container allocation algorithm with the dual purpose of load balancing and network traffic reduction. The authors implemented load balancing on the host by dividing the containers into distinct sections and simulating the traffic as a Zipf-like distribution.

Alahmad et al. [45] proposed a container deployment strategy for cyber-physical cloud systems. Utilizing the ANN model, resource utilization was predicted, and container placement was determined. The outcome of the simulation demonstrated that the implemented strategy not only conserves but also enhances resource utilization. However, for prediction, more sophisticated machine learning models may be investigated.

Hu et al. [46] introduced a novel container location strategy for heterogeneous clusters. A number of optimization objectives were taken into account, such as ensuring the availability of multiple resources, balancing loads, and recognizing dependencies. As a result, the optimization problem was reformulated as a vector bin packing problem. Hu et al. [47] formulated the container deployment problem as a minimum cost flow problem (MCFP) and also introduced an effective container scheduler as a solution. According to the simulation results, the new scheduler made decisions of higher quality and more quickly than its predecessor.

Asensio et al. [48] proposed a strategy for container deployment in a heterogeneous environment to decrease the number of rejected applications and computation nodes. In addition to resource and quality of service requirements, the strategy also incorporated isolation as a security measure. Hussein et al. [49] developed a framework to optimize the resource utilization of both virtual and real machines through container positioning. Despite this, none of the studies examined the utilization of container migration as a means to dynamically optimize the resources.

Table 2.3: Comparative Analysis of Existing State-of-The-Art Approaches of Load Balancing and Energy Conservation using containers

Paper	**Method**	**Computing Paradigm**	**Metrics**	**Migration**	**Novelty**
Zhang et al. [50]	Two Stage Framework	Cloud Computing	CPU Utilization	Containers Migration	Balance Aware Container Placement and Adaptive Threshold Container Migration suggested as solutions to energy consumption optimization issues.
Asensio et al. [48]	Concurrent Container Clusters Scheduling (C3S)	Cloud Computing	Applications Deployed Nodes Utilization	NA	the C3S problem, which aims to optimize the placement of containers in clusters of heterogeneous nodes while adhering to a specified set of quality of service and resource constraints.
Hu et al. [47]	Enhanced Container Scheduler (ECSched)	Cloud Computing	Container performance, Resource Efficiency	NA	Enhanced Container Scheduler (ECSched) is proposed to schedule the concurrent container requests on heterogeneous clusters with multi-resource constraints in an efficient manner.
Shi et al. [51]	Two-stage Multi-type Particle Swarm Optimization	Cloud Computing	CPU, Memory	NA	The proposed TMPSO algorithm for energy-efficient container consolidation in cloud data centres saves more energy than existing methods.
Kumar et al. [52]	Renewable energy-aware multi-indexed scheme	Cloud Computing	Power Consumption	Container Migration	A host selection and container consolidation strategy based on renewable energy is developed.
Tan et al. [53]	Novel GA with dual chromosomal	Cloud Computing	Energy, Computation Time	NA	A novel genetic algorithm (GA) with dual chromosomal representation is proposed to address the problem and GA outperforms state-of-the-art algorithms in energy efficiency across various test cases.
Chen et al. [54]	Container Selection Algorithm	Cloud Computing	average migration rate, VM creation, energy consumption and SLA violations.	Container Migartion	It compared and evaluated a container selection algorithm during consolidation, evaluated its performance based on average container migration rate, virtual machine creation, energy consumption and service level agreement violations. .
Chhikara et al. [55]	Best Fit Container Placement Technique	Fog Computing	CPU Usage, Power, temperature	Container Migration	A novel method for finding the appropriate destination host for container placement to overcome host overload or underload problems.
Li et al. [56]	Container Migration based Decision Mechanism	Edge Computing	Migration cost	Container Migration	Container migration optimized by adjusting the ant colony algorithm, local pheromone volatilization model, and global pheromone updating model.
Kim et al. [57]	Optical Migration System	Edge Computing	Response time and Network load	Live Migration	Three zero-downtime migration strategies are introduced using state duplication and reproduction during migration.
Ma et al. [7]	CMDM Mechanism	Edge Computing	Migration cost	container migration	A modified Ant Colony System (MACS) uses the LBJC model to guide migration activities.
Singh et al. [58]	Probability-based migration technique	Cloud Computing	Data Transfer Time	Container Migration	Pre-copy container live migration to improve performance in various applications and overcome the memory transmission limitations.
Singh et al. [59]	Random fit and FPO algorithm	Cloud Computing	Power Consumption	NA	Extended Flower Pollination Optimization (EFPO) algorithm is recommended to determine the most feasible allocations of resources in order to balance the data center's power usage.
Mangalampalli et al. [60]	DRLBTSA Approach	Cloud Computing	Makespan, Energy consumption	NA	Deep Q-Learning network model, a machine learning technique based on reinforcement learning, to create an algorithm that makes decisions dynamically depending on both ongoing and impending activities.
Proposed Work	DIWPSO algorithm	Fog Computing	CPU, Power temperature and Frequency	Containers Migration	An energy-efficient migration strategy based on a dynamic inertia weight-based PSO technique for containers to meet the resource needs of the host is proposed.

2.3.1 Container Migration

The concept of container migration remains relatively nascent, resulting in limited literature exploring this particular domain. To our understanding, a comprehensive comparative examination of the available container migration methods has yet to be documented within the existing body of literature.

Pickartz et al. [61] explored migration mechanisms for high-performance computing (HPC) and discussed their prospects as well as the challenges. They found that migration mechanisms have to resolve residual dependencies in terms of dynamically changing topologies on the source server. Piraghaj et al. [10] focused on determining the optimal size of a VM for container hosting, aiming to minimize resource wastage at the VM level while executing the total workload. Historical data guided the grouping of application tasks within containers. These tasks were subsequently assigned to containers, which were then deployed on their corresponding VMs.

Sharma et al. [62] conducted a comparison between two virtualization technologies within a large-scale data centre environment: containers and VMs. The evaluation encompassed aspects of performance, manageability, and software development.

In contrast, Piraghaj et al. [63] used performance metrics such as SLA adherence, average VM creation count, and average container migration rate to measure how well their system worked. These metrics were leveraged to assess and compare various algorithms addressing the challenge of container consolidation. Ramalho and Neto [64] analyzed two distinct virtualization methodologies utilized for deploying and managing applications in the cloud, focusing on their relevance to IoT and network edge contexts. A comparative assessment was performed between container-based and hypervisor-based approaches, both operational on network edge devices. Containers were implemented using Docker, embodying lightweight virtualization, while VMs were instantiated using the kernel-based VM for hypervisor-based virtualization at network edge locations.

Singh and Gupta [65] did a thorough analysis of research problems by comparing well-known VM consolidation methods in the context of cloud computing. This helped researchers better understand how VM migration works.

Carver et al. [66] highlighted a phenomenon where memory reclamation occurs upon booting new containers, leaving inactive containers with residual unused memory from extended periods. They introduced a strategy that quantifies container activity based on recent memory demands, constituting the initial phase of their container consolidation approach. Mavridis and Karatza [67] conducted experiments to scrutinise the performance implications of introducing a virtualization layer on top of VMs. Docker was

employed for container deployment and operation, while the KVM hypervisor facilitated VM management.

Chen et al. [54] proposed an algorithm for selecting containers during consolidation, favouring those with minimal migration time. Migration time was evaluated by dividing the RAM usage of containers by the available network bandwidth on the host. Shi et al. [51] developed a two-phase multiobjective particle swarm optimization (TMPSO) technique specifically designed for the purpose of container consolidation in data centres. Kumar et al. [52] developed a novel methodology for host selection and container consolidation that is based on renewable energy considerations. Their approach involved directing duties from a variety of devices to data centres that possessed sufficient reserves of renewable energy. Tan et al. [53] present a genetic algorithm-based approach that makes use of a dual chromosome representation to address the intricate two-level architecture inherent in container-based cloud consolidation.

2.3.2 Optimization Techniques to reduce the Energy Consumption of Hosts

Several optimization techniques have been proposed to reduce the energy consumption of hosts. A few recent works are discussed as follows: Singh et al. [59] introduced the metaheuristic Virtual Machine Placement (VMP) framework to improve power efficiency in sustainable cloud settings. The Extended Flower Pollination Optimization Technique (EFPOA) combines the random fit technique to find optimal allocations for balanced power usage in data centres. Singh et al. [68] created the WOA with Re-initialization and Decomposition (R&D-WOA) approach to optimize VM movement within the cloud data center and reduce physical host machine energy usage.

Çavdar et al. [69] introduced the Utilization-Based Genetic Algorithm (UGA), which uses genetic algorithms to assign virtual machines (VMs) to physical servers in a data centre. This approach considers machine utilization and node distances to reduce resource waste, network congestion, and energy consumption. Nabavi et al. [70] proposed Tractor, a multi-objective system for VMP that treats virtual machines (VMs) as fog jobs for Edge ECDC. This approach uses ABC Optimization Algorithm (ABCOA) to efficiently assign VMs to PMs while considering power and network requirements. The goal of this effort was to reduce network traffic and power consumption in the data centre switches and PM.

Khan et al. [71] presented a hybrid optimization technique for managing VM migration in cloud environment. This hybrid optimization system combines the Hybrid Cuckoo Search

(CSA) and Particle Swarm Optimization (HCPSOA) algorithms. The key goals were to reduce energy use, time computation, and migration expenses. Cloud users are increasing due to benefits such as interoperability, adaptability, and mobility, creating a demand for more computing resources. Saxena et al. [72] proposed a new secure and multi-objective framework for VM placement (SM-VMP) that includes efficient VM migration. The VMP method was implemented using the Whale Optimisation GA (WOGA), which was inspired by whale evolutionary optimisation and GA's non-dominated sorting principles.

Goyal et al. [73] developed optimization techniques such as PSO, Cat Swarm Optimization (CSO), CSA, and WOA to optimize load distribution, energy efficiency, and resource allocation for a well-organized cloud.

2.4 Anomaly Detection Techniques

Anomaly detection techniques are crucial for time series-based load prediction on hosts, as they help identify unusual patterns or outliers that could indicate potential issues or inefficiencies in resource management. People commonly employ techniques like Statistical Process Control (SPC) methods, which use statistical thresholds to detect deviations from expected load patterns, to identify anomalies in time series data. Machine learning-based approaches, including isolation forests and one-class SVMs, offer more sophisticated methods by learning from historical data to recognize patterns that deviate from normal behavior. Additionally, deep learning models like autoencoders and long short-term memory (LSTM) networks can be used for anomaly detection by learning the typical load patterns and flagging deviations as anomalies. These techniques are valuable for detecting unexpected spikes or drops in load, ensuring timely responses to potential issues, optimizing resource allocation and maintaining system stability. By incorporating anomaly detection into time series forecasting, organizations can proactively address anomalies, minimize downtime and enhance overall operational efficiency.

Yue and Meng[74] developed a comprehensive anomaly-based intrusion detection solution by leveraging both point anomaly detection and sub-sequence anomaly detection, which complement each other effectively. This integrated solution is proficient in identifying abnormal patterns while minimizing false alarm rates. It considers the patterns of system load profiles and can accurately detect point anomalies, contextual anomalies, and collective anomalies resulting from either erroneous measurements or cyberattacks. The potential applications of this integrated solution are evident in the planning phase for identifying problematic data. When combined with the automated implementation of

common mitigation strategies, it can facilitate the development and online utilization of cyber-secure short-term forecasting algorithms for load and renewable generation.

Lenka Benova and Ladislav Hudec [75] introduced a method to discover and classify anomalies in HTTP logs without requiring substantial expert analysis or data labelling, which require domain and system knowledge. Numerous studies claim effective prediction and classification, but they often apply to a single dataset and are not universally applicable. They demonstrated the model's applicability by predicting and categorizing anomalies without a tagged dataset while collecting a large dataset from noisy network traffic. The suggested LSTM-based intrusion detection system proved effective in real-world settings, identifying multiple intrusions during use. They categorized anomalies into four categories using 5-minute windowing over predicted and real update curves: unexpected traffic (high load without expected update), unusually large update (high load after update), missed update (low load with expected update), and degraded performance (lower load after update). They demonstrated the system's usability by achieving accurate update curve prediction on noisy web traffic without automatic updates and real-time incident response, crucial for addressing threats to the system and infrastructure.

Gupta, Shaifu et al.[76] analyze the resource utilisation patterns to identify failures that are caused by resource contention in the cloud. The normal and anomalous working behaviours are modelled by analysing the resource utilisation and performance metrics of the working system at regular time intervals. In the initial stage, a hybrid of long short-term memory (LSTM) and bidirectional long short-term memory (BLSTM) is employed to forecast future resource consumption and performance metric values. This framework is divided into two stages. The hybrid model is employed in the second stage to categorize the anticipated state as either normal or aberrant. The proposed anomaly detection model was assessed in a virtual environment that has been established using Docker containers.

Tan et al.[77] have introduced the ALERT system, which offers an adaptive runtime anomaly prediction system for large-scale hosting infrastructures. ALERT utilizes a self-evolving learning algorithm and a tunable anomaly prediction model to accommodate dynamic hosting infrastructures. Their research represents the initial endeavour to accomplish context-aware anomaly prediction for dynamic distributed systems. The ALERT system has been deployed on numerous production hosting infrastructures.

By identifying and addressing unusual patterns or outliers that could disrupt accurate forecasting and efficient resource management, anomaly detection techniques play a pivotal role in enhancing load prediction on hosts. These methods aim to identify deviations from typical load patterns, utilizing statistical process control (SPC) methods, machine

learning algorithms like isolation forests and one-class SVMs, and deep learning models like autoencoders and LSTMs. These methods help people step in at the right time to stop problems like system overloads, inefficiencies, or failures before they happen by finding oddities like sudden increases or decreases in load. The integration of anomaly detection into load prediction models guarantees that predictions are dependable and robust, even in the presence of atypical data points. This proactive approach not only ensures the adequate scaling of computational resources to meet actual demand but also optimizes resource allocation, thereby maintaining system stability and performance. Consequently, anomaly detection is indispensable for maintaining efficient operations, reducing outages, and improving the overall efficacy of load prediction strategies on hosts.

2.5 Machine Learning Techniques

Machine learning techniques are crucial for forecasting host loads and optimizing energy consumption. They provide sophisticated methods for analyzing historical data and predicting future demands with high accuracy. Support Vector Machines (SVM), Random Forests, and Gradient Boosting Machines (GBM) like XGBoost are capable of analyzing large datasets and capturing intricate, non-linear relationships between a variety of factors that influence energy consumption and load. Deep learning models, like recurrent neural networks (RNN) and long short-term memory (LSTM) networks, are excellent at processing sequential data, which makes them perfect for time series forecasting in environment that change quickly. These models are capable of making accurate predictions of future energy and load requirements by learning from past patterns and trends. Machine learning techniques ensure that computational resources are not underutilized or overextended by accurately forecasting demand, thereby facilitating optimal resource allocation. In addition, they facilitate proactive energy management strategies, including the dynamic adjustment of power consumption to comply with anticipated loads, the reduction of energy waste, and the improvement of overall system efficiency. Consequently, the incorporation of machine learning into energy optimization and load prediction results in more cost-effective and sustainable operations.

2.5.1 Decision Tree

A decision tree classifier is a powerful and intuitive tool for the prediction of load and optimization of energy consumption of hosts in computing environment. By analyzing various performance metrics such as CPU usage, memory usage, disk I/O, and network I/O, the classifier can effectively model the complex relationships between these factors

and the target outcomes. Decision trees split the data into branches based on feature values, creating a tree-like structure that makes predictions simple to interpret. For load prediction, the classifier can identify patterns and thresholds that lead to high or low system loads, enabling proactive management of resources. In terms of energy consumption, the decision tree can pinpoint which host configurations or usage patterns lead to higher energy usage, providing insights for optimizing energy efficiency. The straightforward nature of decision trees, combined with their ability to handle both numerical and categorical data, makes them particularly suited for these types of predictive tasks in dynamic and resource-intensive environment [78].

2.5.2 Random Forest Classifier

A random forest classifier is an advanced ensemble learning technique that excels at predicting host load and energy consumption in computing environment. Unlike a single decision tree, a random forest builds multiple decision trees during training and merges their outputs to improve predictive accuracy and control overfitting. Each tree in the forest is trained on a random subset of the data and features, ensuring a diverse set of models that collectively provide robust predictions. For load prediction, the random forest can capture intricate patterns in host performance metrics like CPU usage, memory usage, disk I/O, and network I/O, offering reliable insights into potential system loads. Regarding energy consumption, the classifier can identify key factors and interactions that contribute to higher energy usage, guiding efforts to optimize resource allocation and reduce energy costs. The inherent randomness and ensemble nature of random forests make them highly resilient to noise and capable of delivering superior predictive performance in complex and dynamic computing environment [78].

2.5.3 Gradient Boosting Classifier

The gradient-boosting classifier builds models sequentially, with each new model correcting the previous ones' errors by focusing on the residuals. This iterative approach results in a highly accurate and robust model capable of capturing complex patterns in host performance metrics, such as CPU usage, memory usage, disk I/O, and network I/O. It is particularly effective in identifying subtle relationships and trends that impact system load and energy consumption, enabling more precise predictions and efficient resource management [79].

2.5.4 Extra Tree Classifier

The extra trees classifier, or extremely randomized trees, builds multiple decision trees by selecting random splits for each feature rather than the most informative splits. This randomness helps to reduce overfitting and increase model diversity, leading to improved generalization on unseen data. By analyzing host performance metrics, extra-tree classifiers can provide reliable predictions of load and energy consumption, highlighting critical factors and interactions. Both classifiers offer distinct advantages: gradient boosting excels at accuracy and handling complex data relationships, while extra trees provide robustness and simplicity, making them valuable tools for optimizing host performance and energy efficiency in dynamic computing environment [79].

2.5.5 AdaBoost Classifier

AdaBoost (adaptive boosting) is a powerful ensemble learning technique that can significantly improve host load prediction and energy consumption forecasting. By combining the outputs of multiple weak classifiers, AdaBoost constructs a strong predictive model that iteratively focuses on the most challenging instances. This adaptive nature allows the algorithm to fine-tune its predictions, making it particularly effective for complex and dynamic environment where load and energy consumption patterns may vary over time. In the context of load prediction on hosts, AdaBoost can accurately forecast fluctuations in resource demands, enabling efficient allocation and utilization of computing resources. For energy consumption prediction, the algorithm can identify subtle trends and anomalies in historical data, leading to more precise and actionable insights. By improving the accuracy of predictions, AdaBoost helps in optimizing operational efficiency, reducing energy waste, and enhancing the overall performance of systems that rely on accurate load and energy consumption forecasts [80].

2.5.6 k-Nearest Neighbours Classifier

The k-Nearest Neighbours (k-NN) classifier is an intuitive and effective non-parametric algorithm for load prediction on hosts and optimization of energy consumption. By identifying the k closest data points in the feature space to a given input, k-NN classifies the input based on the majority class among its neighbors. For load prediction on hosts, k-NN uses historical data to discern patterns and predict future resource demands, ensuring efficient allocation and minimizing over-provisioning. Regarding energy consumption, k-NN identifies past instances with similar characteristics to the current context, enabling accurate predictions of future energy needs. Its simplicity and ability to handle nonlinear

relationships make k-NN a robust choice for applications requiring responsiveness to new data and interpret-ability [80].

2.5.7 Linear Discriminant Analysis Classifier

Linear Discriminant Analysis (LDA) is a powerful statistical method used for classification tasks, including load prediction on hosts and optimization of energy consumption. LDA seeks to find a linear combination of features that best separates different classes, making it particularly effective when the data exhibits linear separability. In load prediction on hosts, LDA identifies linear relationships between factors affecting resource usage, facilitating precise and efficient forecasts. For optimizing energy consumption, LDA analyzes historical data to uncover trends and patterns, enabling reliable predictions and proactive management. Its simplicity, computational efficiency, and strong theoretical foundation make LDA an excellent tool for scenarios where linear assumptions hold true and rapid decision-making is required [81].

2.5.8 Light Gradient Boosting Machine Classifier

LightGBM (Light Gradient Boosting Machine) is an advanced gradient boosting framework designed for fast and accurate prediction tasks, including load prediction on hosts and optimization of energy consumption. LightGBM constructs decision trees in a leaf-wise manner, focusing on growing the most significant branches, which results in better accuracy and faster training times compared to traditional boosting methods. For load prediction on hosts, LightGBM can handle large datasets and complex feature interactions, providing detailed and accurate forecasts that are crucial for dynamic resource management. In terms of optimizing energy consumption, LightGBM excels at capturing intricate patterns and temporal dependencies in the data, leading to precise and actionable insights. Its scalability, speed, and ability to handle high-dimensional data make LightGBM an excellent choice for real-time prediction and optimization in energy and resource management systems [82].

2.5.9 Ridge Regression Classifier

Ridge Regression is a powerful linear model that addresses multi-collinearity issues in datasets by adding a regularization term to the loss function, penalizing large coefficients. This approach is particularly useful for load prediction on hosts, where multiple features may exhibit high inter-correlation. By incorporating regularization, Ridge Regression ensures stable and reliable predictions, facilitating effective resource allocation

and load balancing. In the context of energy consumption optimization, ridge Regression can analyze historical consumption patterns while controlling for potential over-fitting, leading to accurate and robust forecasts. This accuracy helps in designing energy-efficient strategies, reducing wastage and optimizing overall energy usage in dynamic environment [83].

2.5.10 Naive Bayes Classifier

Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming the independence of features. Despite its simplicity, it performs well in various applications, including load prediction on hosts and energy consumption optimization. For load prediction, Naive Bayes can quickly and efficiently analyze the likelihood of different load levels based on historical data, providing prompt and reliable forecasts for resource management. In energy consumption optimization, Naive Bayes leverages historical energy usage data to predict future consumption patterns by calculating the probability of different energy usage levels. Its simplicity, speed, and efficiency make Naive Bayes a suitable choice for real-time prediction and optimization tasks, particularly in environment where computational resources are limited [84].

2.6 Time Series Prediction Techniques

Time series-based prediction techniques are essential for accurate load prediction on hosts, as they effectively capture and model temporal patterns and dependencies in data. Techniques such as Autoregressive Integrated Moving Average (ARIMA) provide robust forecasting by combining autoregression, differencing, and moving averages to handle stationary time series data. Seasonal Decomposition of Time Series (STL) excels at managing complex seasonal variations and trends by decomposing the time series into seasonal, trend, and residual components. Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, are adept at capturing long-term dependencies and complex patterns in load data, making them suitable for dynamic environment. Exponential Smoothing State Space Models (ETS), including methods like Holt-Winters, are valuable for their ability to adapt to changes in trends and seasonality. Prophet, developed by Facebook, offers a flexible approach to forecasting that handles seasonal effects, holidays, and trend changes, even in the presence of missing data and outliers. These techniques collectively enhance the accuracy of load predictions, enabling better resource management and optimization in computational systems.

2.6.1 ARIMA

The ARIMA model is one of the most important tools for the prediction of time series. It combines autoregressive (AR) processes, differencing (I) to achieve stationarity, and moving averages (MA) to pick up on temporal dependencies and noise. ARIMA is particularly effective for predicting future values in a time series because it models the relationship between past observations and future forecasts. The autoregressive component leverages the dependency between an observation and a number of lagged observations, while the moving average component models the relationship between an observation and a residual error from a moving average model. Differencing is applied to make the time series stationary, meaning its statistical properties remain constant over time. ARIMA is widely used for load prediction on hosts due to its ability to handle various patterns in historical data, such as trends and cycles, providing accurate forecasts that assist in resource management and optimization. Its flexibility in modelling different types of time series data makes it a valuable tool for anticipating future load demands and ensuring efficient allocation of computational resources.

Zhang et al. proposed an efficient deep learning model named Sibyl to enhance the accuracy and efficiency of load prediction [85]. Sibyl consists of two components: a metrics selection module and a neural network training module. Initially, Sibyl filters out irrelevant metrics to select the most relevant ones. Subsequently, it utilizes a robust neural network model built with bidirectional long short-term memory (BiLSTM) to predict the actual load one step ahead. Experimental results demonstrated that Sibyl can reduce the number of training metrics while maintaining high prediction accuracy. Moreover, Sibyl significantly outperforms other state-of-the-art methods, including those based on autoregressive integrated moving average (ARIMA) and long short-term memory (LSTM), in terms of prediction accuracy.

Calheiros et al. have presented the development of a cloud workload prediction module for SaaS providers using the autoregressive integrated moving average (ARIMA) model [1]. This module uses ARIMA for future workload prediction, and it evaluates its accuracy using real traces of web server requests. Additionally, they assess the impact of the model's accuracy on resource utilization efficiency and quality of service (QoS). Simulation results indicate that our model achieves an average accuracy of up to 91 percent, resulting in efficient resource utilization with minimal impact on QoS.

2.6.2 Prophet

Prophet, a more recent addition to the forecasting toolset, was developed by Facebook's Data Science team. Prophet is expressly engineered for time series data that exhibits robust seasonal patterns, which are frequently encountered in business applications. It provides a number of benefits over conventional models, such as the capacity to manage missing data, the automatic detection of changepoints, and intuitive parameters for modelling seasonality and vacations.

Daraghmeh et al. [86] employed the Facebook Prophet forecasting framework on Microsoft Azure VM workload to predict future resource utilization for running tasks. Their research showed that using data preprocessing and transformation on real virtual machine traces, along with an automatic model hyperparameter tuning process, can greatly improve the accuracy of forecasts, with an average gain of over 85%. Moreover, cloud providers can leverage their data center workload and utilize various forecasting models to achieve significant improvements in cost-efficient resource management.

Chen et al. [87] have introduced Prophet, a method that accurately forecasts the performance degradation of latency-sensitive applications on accelerators as a result of application co-location.They conducted a real-world system investigation to analyse the performance interference on accelerators and discovered that, in contrast to multicores, where the primary contentious resources are shared caches and main memory bandwidth, the primary contentious resources on accelerators are processing elements, accelerator memory bandwidth, and PCIe bandwidth. In response to this observation, they developed interference models that allow for precise prediction of processing elements, accelerator memory bandwidth, and PCIe bandwidth contention on actual hardware. Prophet can accurately anticipate the performance degradation of latency-sensitive applications on non-preemptive accelerators by employing a novel technique to forecast solo-run execution traces of the co-located applications using interference models. Prophet enables them to identify "safe" co-locations on accelerators, thereby enhancing utilization without violating the QoS target.

2.6.3 RNN LSTM

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are effective tools for time series-based load prediction on hosts, offering enhanced capabilities for capturing temporal dependencies in load data. RNNs are designed to handle sequential data by maintaining hidden states that carry information from previous time steps, allowing the model to learn patterns and trends over time. However, traditional RNNs

can struggle with long-term dependencies due to issues like vanishing gradients. LSTMs address these limitations by incorporating gating mechanisms—input, forget, and output gates—that regulate the flow of information and maintain long-term memory. This makes LSTMs particularly well-suited for predicting future load demands, as they can remember and utilize information from distant past data points. By analyzing historical load patterns, LSTMs provide accurate forecasts of future load, enabling efficient resource management and optimization on hosts. Their ability to model complex temporal dynamics ensures that predictions reflect both short-term fluctuations and long-term trends, leading to more effective load balancing and resource allocation strategies.

Nguyen et al. [88] have proposed a prediction method that is based on the Long Short-Term Memory Encoder–Decoder (LSTM-ED) and is capable of predicting both the mean load over consecutive intervals and the actual load multi-step in advance. By constructing an internal representation of time series data, their LSTM-ED-based approach enhances the memory capacity of LSTM, which is employed in the most recent previously published work. They have conducted experiments using a 1-month trace of a Google data centre with over twelve thousand machines to assess our approach. The experimental results indicate that the LSTM-ED-based approach obtains a higher level of accuracy than other previous models, despite the fact that multi-layer LSTM results in overfitting and a decrease in accuracy when compared to single-layer LSTM, which was employed in the previous work.

2.6.4 Convolutional LSTM

Convolutional LSTM (ConvLSTM) networks offer a powerful approach for time series-based load prediction on hosts by integrating convolutional layers with Long Short-Term Memory (LSTM) units. This hybrid architecture effectively addresses the challenges of modeling both spatial and temporal dependencies in load data. The convolutional layers in ConvLSTM extract spatial features from sequential data, such as load patterns across different servers or data center components, while the LSTM layers capture temporal dependencies, learning from past load trends to forecast future demands. This combination allows ConvLSTM to analyze complex interactions between spatial and temporal factors, resulting in more accurate and nuanced load predictions. By leveraging both spatial correlations and temporal dynamics, ConvLSTM enhances the ability to optimize resource allocation and manage computational resources efficiently, making it particularly useful for environment where load patterns exhibit intricate spatial relationships and dynamic changes over time [89].

2.6.5 XgBoost

XgBoost, an advanced gradient boosting framework, is increasingly used for time series-based load prediction on hosts due to its ability to handle complex and non-linear relationships in data. Unlike traditional time series models that focus solely on sequential data, XgBoost leverages gradient boosting to combine the predictions of multiple weak models, typically decision trees, to create a robust and accurate forecasting model. For load prediction, XgBoost can effectively process features derived from historical load data, such as time lags, rolling statistics, and external variables, to capture intricate patterns and trends. Its regularization techniques help prevent overfitting, making it well-suited for dynamic environment where load patterns can vary significantly. XgBoost refines its predictions iteratively by fitting a series of trees that focus on the residual errors of previous trees. This results in precise and reliable forecasts. This capability enhances resource management by providing actionable insights into future load demands, leading to improved efficiency and optimization in computational resource allocation [90].

2.6.6 Dynamic Linear Model (DLM)

DLMs are a versatile and effective approach for time series-based load prediction on hosts, offering a probabilistic framework for modelling and forecasting temporal data. DLMs extend traditional linear models by incorporating dynamic components that account for changes in the underlying system over time. This allows them to adapt to evolving load patterns and trends in a data centre or computing environment. By modelling the load as a combination of latent variables and observable data, DLMs can capture both short-term fluctuations and long-term trends, providing a comprehensive view of future load demands. The flexibility of DLMs enables them to incorporate various factors, such as seasonality and external influences, into the prediction process. This adaptability enhances their accuracy in forecasting future load requirements, leading to more efficient resource allocation and optimization. Overall, DLMs offer a robust tool for managing dynamic load environment by providing reliable and actionable insights into future computational needs [91].

Chapter 3

Research Methodology

The methodology employed to accomplish the thesis's research objectives is the focus of this chapter. Figure 3.1 illustrates that the methodology is divided into two components: computationally intelligent approaches and requirement specifications. Requirement specifications encompass a comprehensive examination of a variety of Python programming language libraries, from data generation to model construction. In the ML-based approach, a variety of models were constructed independently using ML and DL algorithms and subsequently combined to form an ensemble model.

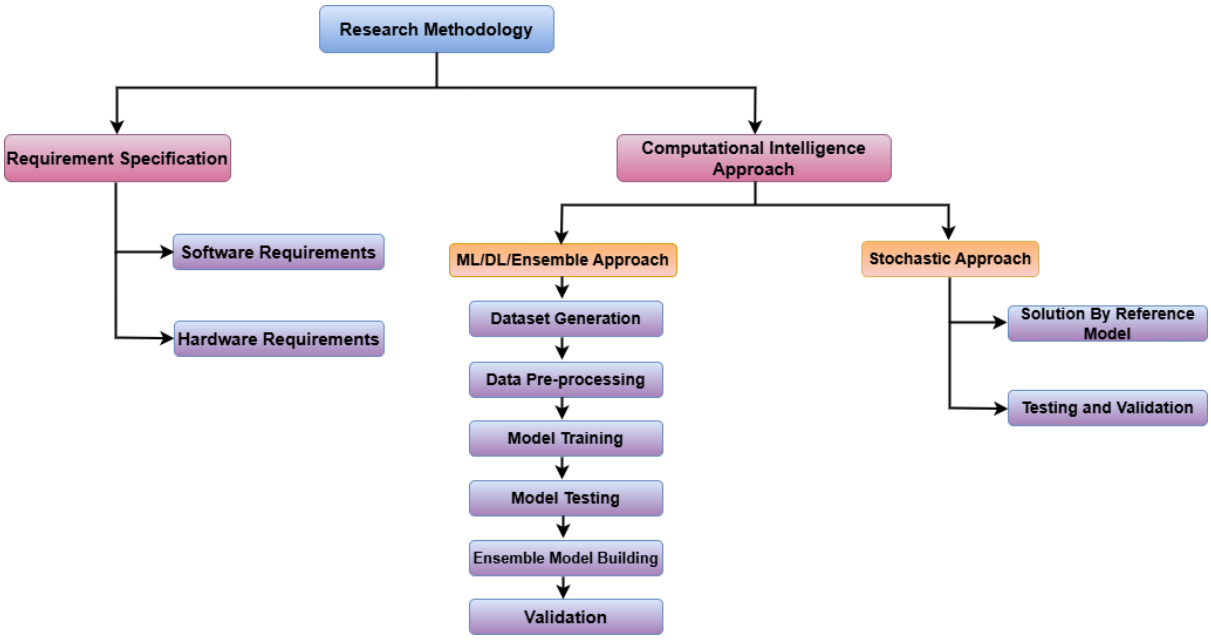


Figure 3.1: Flow of Research Methodology to attain the proposed objectives

3.1 Requirement Specifications

To fulfill the thesis's stated objectives, it is critical to understand the necessary specifications for both the hardware and software components. The specifications serve the same purpose.

3.1.1 Software Requirements

The following tools and libraries available in various programming languages are used:

- Python Libraries: A variety of Python open-source libraries were utilized to train, test and validate the model. They are described as follows:

    1. **Pandas:** Data science, statistics, and machine learning extensively employ Pandas, an open-source Python-based data analysis and manipulation library. It facilitates user-friendly manipulation of labelled and relational data by offering flexible and efficient data structures, particularly data frames and eries. Pandas enables users to effortlessly execute a diverse array of data operations, including merging, reshaping, selecting, and cleansing datasets. It enhances its capabilities for data analysis and visualisation by seamlessly integrating with other Python libraries, such as NumPy, SciPy and Matplotlib.

    2. **NumPy:** In Python, NumPy is a fundamental library for numerical computing. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. Due to its powerful capabilities and ease of use, NumPy finds widespread use in data analysis, machine learning, scientific computing, and engineering. It forms the backbone of many other scientific libraries in Python, such as SciPy, Pandas, and TensorFlow.

    3. **Matplotlib:** It is utilized for the purpose of data visualisation, which is of utmost importance in comprehending data distributions, trends, and models.performance.

    4. **Scikit-learn:** This library provides a wide range of methods for machine learning, making it adaptable and flexible. It include methods for validating models,as well as algorithms for classification, regression, and clustering.

    5. **TensorFlow:** Google developed TensorFlow, an open-source library for deep learning and machine learning. TensorFlow stands out for its versatility and ability to handle intricate computations. It is extensively employed in the tasks of natural language processing and speech and image recognition.

    6. **Keras:** An advanced neural network application programming interface(API) was specifically created to facilitate and expedite the process of developing prototypes. It serves as a platform for TensorFlow, simplifying the process.process of creating deep learning models with its intuitive and modular design.

7. **Oracle VM virtualBox:** The Oracle Corporation has developed VirtualBox, a hypervisor for x86 virtualization that is both free and open-source. By establishing and administering virtual machines (VMs), it enables users to operate multiple operating systems concurrently on a single physical machine. Each virtual machine (VM) has the ability to operate a distinct operating system, including Windows, Linux, macOS, and others, without affecting the host operating system.

8. **OpenCV:** For real-time computer vision and machine learning applications, OpenCV is an open-source software library. It was initially developed by Intel and is currently supported by Willow Garage and Itseez. The library includes a comprehensive array of tools for object detection, feature extraction, machine learning, and image and video processing.

- Containers are running on VMs/Hosts. Range of Various parameters used for VM creation is described as:
  - Number of CPU cores: 4-10
  - RAM used: 1GB-6GB
  - Disk Write Throughput(file size): 1GB-5GB
  - Disk Size: 80GB
  - Memory: 2048MB-8192MB

3.1.2 Hardware Requirements

Platforms used for dataset generation and implementation: The configurations of the two platforms that are used for the generation of dataset and for the implementation of the research work is described in the following Table 3.1.

Table 3.1: Platforms used in data set creation and experimentation

Platform 1	**Platform 2**
Processor-11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz	INTEL(R) XEON(R) CPU E5-2650 V2 @ 2.40GHZ
RAM 12GB	RAM 16GB

3.2 Computational Intelligent Approaches

Computational intelligence (CI) methodologies are a collection of techniques that imitate natural processes and systems in order to address intricate problems that are challenging for conventional computational methods.

Artificial intelligence, which is the term commonly used to refer to the intelligence of computers, is the field of study in which computer machines are trained to learn and behave in human-like ways. The learning experience is the focal point of the entire procedure. The process of constructing a computationally intelligent framework to achieve the objectives is outlined below:

1. **DataSet Generation:** The dataset is generated by deploying a variety of virtual machines in order to gain a deeper understanding of the host load. The dataset consists of 10,000 recordings, each containing ten performance parameter values. RAM, CPU cores, disk read/write throughput, disk size, and memory are all parameters that are considered during the VM creation process. A variety of operational host features are derived, such as CPU utilization, number of cores, RAM, allocated memory, available memory, disk I/O, and network I/O.

2. **Data Pre-processing:** It was verified that all feature values were numeric and that they were within a particular range before the dataset was processed. The labelled encoding method was employed to replace any missing values with the mean of the feature vector that corresponded to them, and non-numeric values were converted to numeric values. Normalization, transformation, PCA, and outlier removal are implemented subsequent to dataset generation. Data pre-processing eliminates superfluous columns, including system, iowait, and memtotal, from the dataset that is generated.

3. **Features Selection:** The term "feature importance" denotes the process of selecting features from a given dataset that make the most significant contribution to the classification of the trained dataset and the prediction of the objective variable. PyCaret is a Python library that specializes in automating machine learning processes and procedures [92]. PyCaret is capable of assessing the importance of a feature. PyCaret's Feature Importance operation identifies the features that have the most significant influence on the classification accuracy in the subsequent sequence. Once we obtain the classification result, we can use the dataset to determine the significance of the features. We must start by defining the optimal model, which we refer to as the proposed ensemble model. The proposed model encompasses a variety of

models, such as random forests. Random Forest generates the mode of classes or the average prediction of individual trees by constructing a substantial number of decision trees during the training process. By concentrating on the most informative characteristics, it is possible to gain insights into the dataset and improve the model by comprehending the value of the dataset's features.

4. **Machine Learning Model Building:** ML is a subfield of artificial intelligence that concentrates on the development of statistical models and algorithms. These models enable computer systems to improve their performance on a specific task by learning from experience or training on data. A variety of machine learning models are employed to classify the workload on hosts as overloaded, underloaded, or balanced.

5. **Deep Learning Model Building:** Deep learning models' ability to capture intricate patterns and long-term data dependencies has led to their increasing popularity for workload prediction on hosts. The capacity of Long Short-Term Memory (LSTM) networks to manage time series data sets them apart. Compared to conventional methods, these deep learning models provide substantial enhancements in prediction accuracy, thereby facilitating improvements in computational system optimization and resource management. Deep learning models ensure that predictions incorporate both short-term fluctuations and long-term trends by utilizing their capacity to model complex temporal dynamics. This enables more effective resource allocation and load balancing.

6. **Ensemble Model Building:** Ensemble learning is a machine learning technique that enhances the accuracy and resilience of the overall prediction by integrating numerous models. Ensemble learning can frequently outperform individual models and decrease the probability of overfitting by combining the predictions of multiple models. This study uses an ensemble learning model to predict host load. This model combines the results of several deep learning and machine learning models, such as ARIMA, ConvLSTM, and DLM. This approach enhances the overall precision and consistency of host load prediction by leveraging the combined strengths of numerous diverse models.

7. **Testing and Validation:** Machine learning models, including those used to predict host load, undergo critical phases of testing and validation. Assessment of the model's performance on a distinct dataset that was not employed during the training phase is the essence of testing and validation. This facilitates the assessment of the model's ability to generalise and reliably predict the class labels of novel, unobserved data. During the testing and validation phases, it is critical to ensure

that the model does not excessively conform to the training data and can accurately predict the class labels of novel, unobserved data. It is helpful in identifying any deficiencies in the model and provides insight into areas that may require improvement. The testing and validation process evaluates the model's performance on a unique dataset, thereby guaranteeing its effectiveness in generalising to new data and its robustness. The dataset is frequently divided into two subsets: a training set and a testing set, as part of the process of load prediction on hosts. The testing set evaluates the model's performance, while the training set instructs the model. Accuracy, precision, recall, F1-score, and the area under the curve (AUC) of the receiver operating characteristic (ROC) curve are all utilized to evaluate the model's performance.

3.3 Summary

In order to optimize energy consumption, this chapter addressed the hardware, software, tools, techniques, and approaches employed in the building and validation of models for load prediction on hosts.

Chapter 4

Multilevel Ensemble Model for Load Prediction on Hosts in Fog Computing Environment

The purpose of this study is to address the issue of load balancing in fog computing, which is crucial for maximizing resource utilization in response to the growing demand for various IoT applications. Fog nodes often become overloaded, necessitating effective load balancing to achieve workload consolidation. This study aims to determine the load on hosts using various parameters, including CPU utilization, the number of CPU cores, RAM, memory allocated, memory available, disk I/O, and network I/O, to better understand host workload.The proposed work involves detecting the load status of hosts using an ensemble approach, categorizing them into three states: under-loaded, balanced and overloaded. The study considers three case studies, executing varying numbers of virtual machines (VMs) with different parameter combinations. Each case study executes a different number of VMs in parallel on two different platforms. Advanced machine-learning models are employed to predict the load on multiple hosts. Models with higher accuracy, based on performance evaluation criteria, are selected to construct an ensemble model. The ensemble method is applied to address the worst-case scenario in model prediction. For the selected case studies, the Random Forest model, Ada Boost Classifier, Gradient Boost, and Decision Tree models perform better than other models. The proposed ensemble model outperforms these state-of-the-art predictive models, achieving an improved accuracy of nearly 82% by correctly classifying hosts as overloaded, under-loaded and balanced.

4.1 Overview

Mobile Internet, Cyber Physical Systems, along with the Internet of Things (IoT) have empowered a multitude of entities, including individuals, machinery and objects, to establish continuous connections with the digital realm, irrespective of time or location. Data is being generated in unprecedented quantities and variety [93, 94].

Cisco predicts that by 2023, the number of networked gadgets per user will increase from 2.4 in 2018 to 3.6. This surge will drive the total count of networked devices from 18.4 billion in 2018 to an estimated 29.3 billion by 2023. This proliferation in connected devices corresponds to a remarkable surge in data generation rates. In a recent study including a healthcare related IoT application, it has been found that thirty million people produce 25,000 data tuples per second collectively [95].

Given this tremendous data volume, the current processing and storage capacities fall short of the demands. Conventional computing methods, including distributed computing and cloud computing, struggle to cope with this data deluge. As a more pragmatic solution, fog computing has emerged, bridging the gap between network edge devices and centralized cloud centers to efficiently address these limitations [8].

Fog Computing is a middle man between the Internet of Things and the cloud that brings information closer to the sensors [96]. Nevertheless, considering the increasing demand for a variety of IoT applications, fog nodes frequently become overloaded, degrading the response times of IoT apps with latency constraints and as a result, compromising users' quality of experience [97].

Understanding workload characteristics and underlying data centers are critical for both data center operators and fog service providers to sustain the adoption of fog data centers and increase data center operators' ability to tune existing and build new resource management approaches. The architecture of fog computing is shown in Figure 4.1.

Google Data Centers reportedly utilized an estimated 260 million Watts of electrical power in 2013, accounting for one percent of global energy consumption [98]. A 2020 New York Times article states that data centers consumed around 1% of global electrical output 2018. The inefficient use of hardware resources is the cause of high energy consumption. Servers inactive can use up to 60% of their peak power [99, 100].

Virtualization capabilities, which were initially introduced by IBM in the early 1960s as a transparent method to facilitate continuous and direct access to powerful computers by numerous users, represent a valuable method for addressing energy inefficiencies [101].

This virtualization technique abstracts the physical resources that underlie a system, thereby generating an integrated view of a resource pooling within fog data centers, where a single server can accommodate multiple virtual machines. Virtualization provides several advantages for distributed systems, including increased resource utilization, enhanced isolation, improved manageability, and heightened reliability [102].

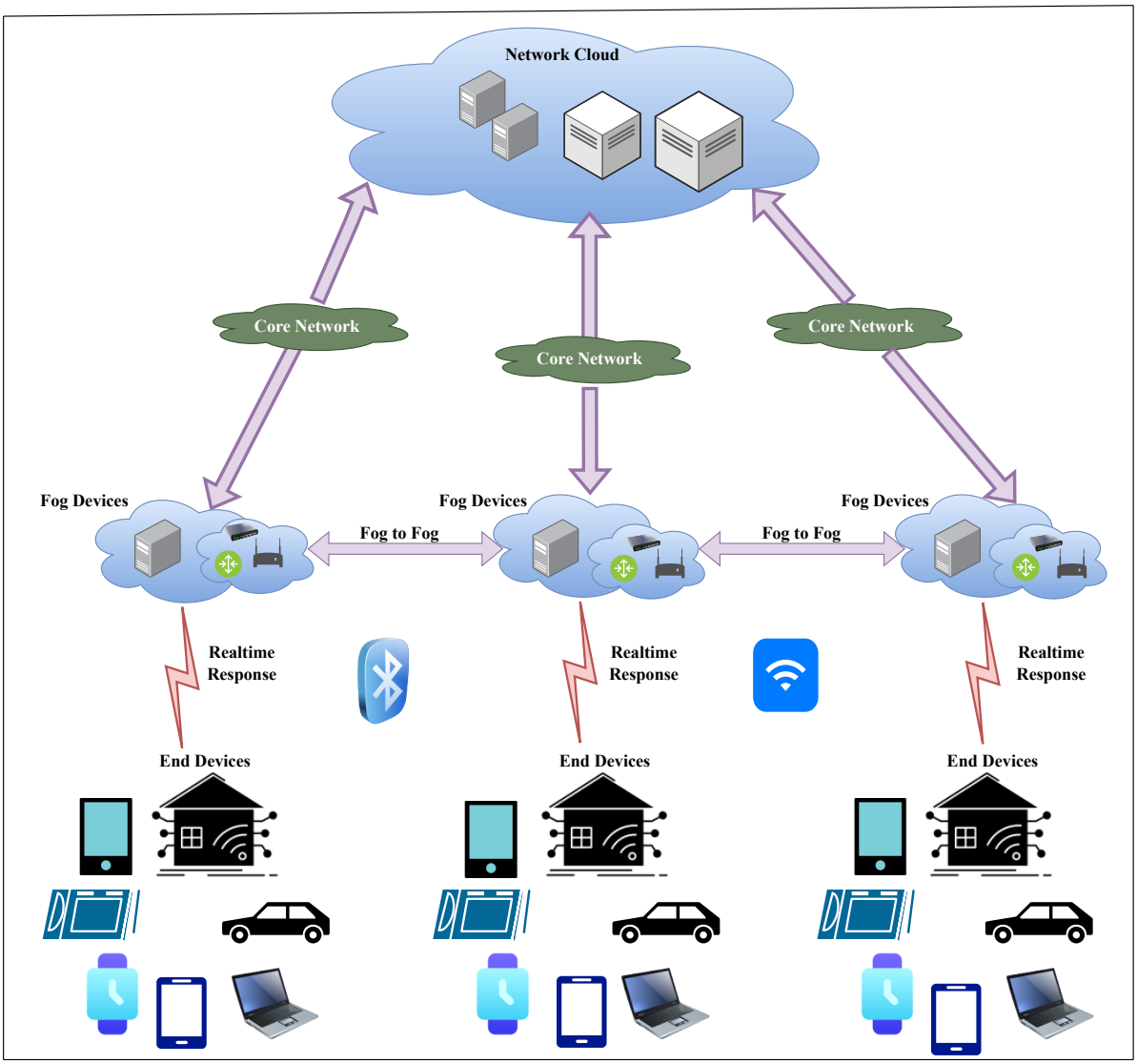


Figure 4.1: The Architecture of Fog Computing

It becomes crucial to assess the host's workload to achieve effective workload consolidation. This is useful when migrating containers from a particular virtual machine to another in order to reduce the number of service level agreement (SLA) violations and balancing of loads violations.

A novel framework has been developed to classify hosts into overloaded, underloaded and balanced hosts. Then the process of workload consolidation in a fog environment is discussed by detecting overloaded hosts. Containers represent a groundbreaking innovation in the era of cloud computing due to their lightweight nature, simplified configuration and administration, and substantial reduction in startup times [103].

Consequently, a diverse range of applications can be seamlessly executed within these containers. To achieve this, we've employed Podman to run containerized applications on VMs. We can run heavy applications through containers as these are lightweight. The two platforms we have used are shown in Table 4.5.

We can efficiently run heavy applications through containers, such as applications related to heavily loaded images due to high resolution. Containers are executed on the virtual machines, created using the command line with various random parameters, including the CPU, number of cores, RAM, memory allocated, memory available, disk Input/Output and network I/O in order to understand the host workload better. Figure 4.2 shows the container-based virtual machines.

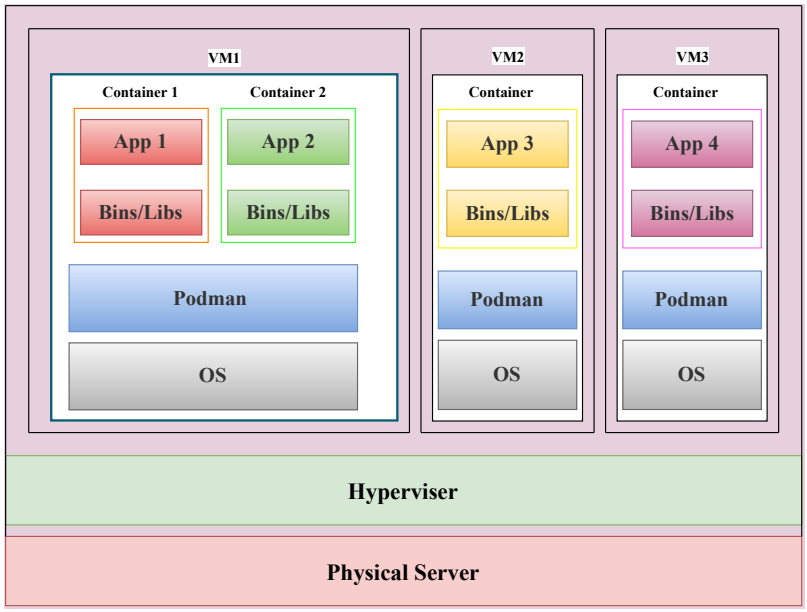


Figure 4.2: Container-based Virtual Machines (VMs)

In this work, own dataset is generated using various types of VMs. The entire dataset is merged, after which machine learning models are applied to detect loads on multiple hosts.

Several neural network-based methods for improving prediction accuracy have been proposed. However, accuracy of these methods is insufficient and considers specific workload.

As shown in Figure 4.1 fog computing effectively bridges the gap between network gadgets at the network's periphery and centrally located cloud computing facilities, emerges as a simpler approach to manage these challenges. In our proposed work, a multilevel ensemble model for classifying hosts as overloaded, underloaded, or balanced is presented. Multiple features are extracted in the proposed work. All of the features utilized by the

proposed model—including CPU utilization, number of cores, RAM, allocated memory, disk I/O, available memory and network I/O are extracted from running hosts.The experimental outcomes demonstrate that the performance of the proposed multilevel ensemble model for prediction of load gives more accuracy as compared to existing state-of-the-art approaches.

4.2 Materials and Methods

4.2.1 Parameters used for DataSet Generation

The dataset contains ten thousand records with eleven performance parameter values. For predicting whether a host is overloaded, underloaded, or balanced, the total CPU usage parameter is retained as the target variable, while the remaining variables in the dataset serve as input for machine learning models. Each row of the dataset represents a performance metric observation. Each row is formatted as follows:

1. CPU cores: The quantity of number of virtual CPU cores allocated.
2. user: The percentage of CPU consumption that happened while the user was executing (application).
3. nice: Percentage of CPU utilization during user-level execution with pleasant priority.
4. Total Usage of CPU: in terms of percentage.
5. Ideal CPU: The percentage of time the CPU or CPUs were idle when no disk I/O requests were pending.
6. Memory Total: the memory given to the system in terms of KB.
7. Memory free: the memory that is actively not used in terms of KB.
8. Memory available: An estimate of the amount of memory available to execute new applications without swapping in additional KB.
9. Disk read/write throughput: in terms of KB/s.
10. Network received throughput: Received network throughput in terms of MBITS/s.
11. Network transmitted throughput: Transmitted network throughput in terms of MBITS/s.

Table 4.1: Basic Statistics of various Features

	user	**nice**	**idle**	**Total CPU Usage**	**Network transmitted Throughput**	**Network recieved Throughput**	**cpu cores**	**MemTotal**	**MemFree**	**MemAvailable**	**Disk write Throughput**
CoV	60.84	126.38	23.39	63.81	226.72	223.17	44.96	48.03	103.84	86.15	457.92
mean	12.58	0.36	73.18	26.82	131.18	137.06	6.01	3690608.00	264915.80	1318698.00	0.18
std	7.65	0.45	17.11	17.11	297.42	305.87	2.70	1772700.00	275097.20	1136045.00	0.81
min	1.01	0.03	27.96	2.35	0.00	0.00	1.00	1026028.00	53468.00	21752.00	0.00
25%	6.43	0.12	62.16	14.09	14.10	14.00	4.00	1537580.00	101126.00	446484.00	0.00
50%	11.59	0.20	81.07	18.93	19.10	18.80	7.00	4100348.00	139480.00	1127272.00	0.02
75%	17.13	0.29	85.91	37.85	23.45	23.20	9.00	4666636.00	298198.00	1616788.00	0.09

Table 4.2: Sample DataSet

user	**nice**	**system**	**iowait**	**idle**	**CPU Usage**	**Network transmitted Throughput**	**Network recieved Throughput**	**CPU cores**	**Mem Free**	**Memory Available**	**Disk write throughput**
16.05	0.1	21.49	13.78	48.59	51.41	17.0 MBITS/SEC	15.3 MBITS/SEC	9	54756	37168	1.06206131
16.11	0.1	21.85	13.92	48.02	51.98	20.4 MBITS/SEC	16.9 MBITS/SEC	9	61428	47468	0.985242367
15.96	0.1	22.47	14.21	47.27	52.73	12.7 MBITS/SEC	10.3 MBITS/SEC	9	54820	21752	0.130439281
15.82	0.1	23.05	14.45	46.58	53.42	11.3 MBITS/SEC	11.2 MBITS/SEC	9	55792	41660	0.442959309
15.58	0.09	24	14.65	45.68	54.32	16.3 MBITS/SEC	16.8 MBITS/SEC	9	53468	26824	10.23108006
19.2	0.32	15.92	1.76	62.79	37.21	25.1 MBITS/SEC	20.0 MBITS/SEC	4	514724	1415904	0.375194073
20.16	0.32	16.4	2.38	60.73	39.27	23.1 MBITS/SEC	24.7 MBITS/SEC	4	619868	1403036	0.002605438
22.34	0.3	18.25	2.87	56.24	43.76	23.4 MBITS/SEC	19.8 MBITS/SEC	4	188060	970992	0.495999813
22.6	0.3	18.16	4.62	54.32	45.68	18.8 MBITS/SEC	21.4 MBITS/SEC	4	223156	1319912	0.70169425
29.94	0.23	25.14	6.64	38.04	61.96	22.7 MBITS/SEC	11.9 MBITS/SEC	4	708996	1570532	0.004311085
30.44	0.23	25.59	6.5	37.24	62.76	13.7 MBITS/SEC	19.4 MBITS/SEC	4	891736	1558736	0.625935555

Table 4.3: Range of various parameters used for VM Creation

SN	Parameter	Range
1	CPU cores	4-10
2	RAM	(1-6GB)
3	Disk Write Throughput(file size)	(1-5GB)
4	Disk Size	80GB
5	Memory	(2048-8192MB)

Basic features statistics are calculated for the statistical characterization, including the min, the standard deviation (SDev), the mean and the unitless coefficient of variation (CoV). Table 4.1 shows basic statistics of various features used in this work. Table 4.2 demonstrates the sample of the dataset used in the course of this work.

4.2.2 Dataset Generation using Modeling and Simulation

To better understand the workload of hosts, virtual machines are generated through the command line through different random parameters. The proposed model targets a Container as a Service (CaaS) environment where an application is executed on containers. Various applications are running on the containers and containers are executed on these virtual machines. A whole dataset is merged, a new dataset is generated and then a set of machine learning models is applied to them to detect loads on various hosts. The flow chart for creating VMs through the command line is shown in Figure 4.3. Table 4.3 below describes performance parameters and their range used for the construction of VMs through the command line for this work.

4.2.2.1 Transformation of Data

Static thresholds T_{ol} and T_{ul} are used to decide if a host's load is overloaded, under-loaded and balanced as shown in equation 6.18 respectively .

$$HostStatus = \begin{cases} Overloaded & \text{if } U_{(i,t)} > T_{ol} \\ Underloaded & \text{if } U_{(i,t)} < T_{ul} \\ Balanced & \text{if } T_{ul} < U_{(i,t)} < T_{ol} \end{cases} \tag{4.1}$$

where, T_{ol} and T_{ul} represents the overloaded and underloaded thresholds and $U_{(i,t)}$ is the utilization of CPU of host i at a particular time t. The power consumption of the CPU is considered to be a critical factor that exhibits the greatest variability in power usage based on its utilization rate. CPU Utilization is considered a target variable, and the CPU Utilization feature binning takes place. The transformation of the dataset is done

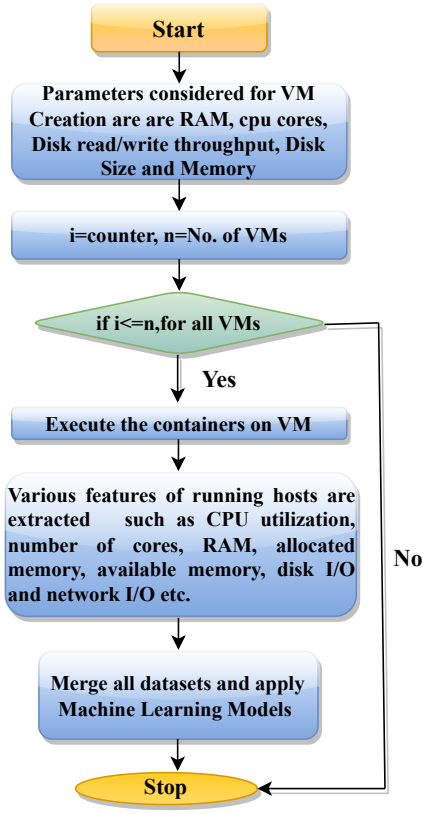


Figure 4.3: Flow Chart of Data Generation

to label the data and classifies them into three classes: overloaded (O), underloaded (U) and balanced (B). The sample of dataset transformation is shown in Table 4.4.

Table 4.4: Sample of DataSet Transformation

user	nice	idle	Total CPU Usage	Network transmitted Throughput	Network recieved Throughput	cpu cores	MemFree	Mem Available	Disk write throughput	Label
19.63	1.22	74.76	25.24	9.07	4.77	1	1430276	5860808	0.668997049	U
57.72	18.16	0.05	99.95	31.4	27.6	2	1048228	2578964	0.04445672	O
56.19	20.9	0.04	99.96	28.3	27.8	2	676580	2209468	0.374528646	O
25.18	0.97	68.5	31.5	43.5	46.4	3	546796	3116432	0.005527258	B
19.93	1.2	74.36	25.64	5.23	4.93	4	1406644	5900648	0.769234896	U
24.93	0.97	68.79	31.21	50.6	52.7	3	636200	3210656	1.203319788	B
24.85	0.97	68.89	31.11	45	49.7	3	674404	3250004	1.265292645	B
57.29	19.26	0.05	99.95	27.2	27.4	2	906184	2437896	0.002002001	O

4.2.2.2 Load prediction on Physical Machine

A virtual machine (VM) behaves like a physical machine (PM). So each VM has its CPU, memory and bandwidth.The key component for estimating the energy consumption of hosts is the CPU's power consumption. CPU utilization presents the most significant variance in power consumption concerning its utilization rate. CPU utilization is regarded as a target variable. Load on the i^{th} VM can be calculated using equation 6.21 as

follows:

$$VM^{i}_{load} = VM^{i}_{CPU} \tag{4.2}$$

Determine the weight on the PM Since multiple VMs operate on each host. Thus, as the equation 6.20 illustrates, the overall load on the PM equals the sum of all the VM loads.

$$\mathrm{PM}^{m}_{load} = \sum_{i=1}^{n} \frac{\mathrm{VM}^{i}_{load}}{n} \tag{4.3}$$

Where n represents the number of virtual machines on the mth host.

4.2.2.3 Service Level Agreement Metric

This effectiveness metric represents the mean percentage of service level agreement (SLA) violations. It becomes relevant when virtual machines fail to obtain the requested resources or when the shared host's average computing capacity isn't allocated to the requested virtual machines, as discussed in [104]. This metric directly influences the Quality of Service (QoS) level. Equation 6.19 outlines the calculation for the average SLA violation as follows:

$$A_{SLA} = \frac{\sum_{j=1}^{q} \frac{RM_j - AMj}{RM_j}}{q} \tag{4.4}$$

In this context, A_{SLA} signifies the average unallocated VM Million Instructions Per Second (MIPS), which results in a decline in performance. RM represents the requested MIPS by a VM, AM represents the MIPS actually allocated to the VM during its operation, and q signifies the total number of VMs.

4.2.3 Feature Importance

The term feature importance refers to the procedure of feature selection from a given dataset that makes the greatest contribution to the classification of the trained dataset and the prediction of the objective variable. PyCaret is a Python library [92] designed for automating machine learning processes and workflows. PyCaret has the capability to determine the significance of a feature. The Feature Importance operation of PyCaret finds out the features that have the greatest impact on the accuracy of classification in the following sequence.

- Generation of Dataset
- Multi-classification
- Selection of Relevant Features

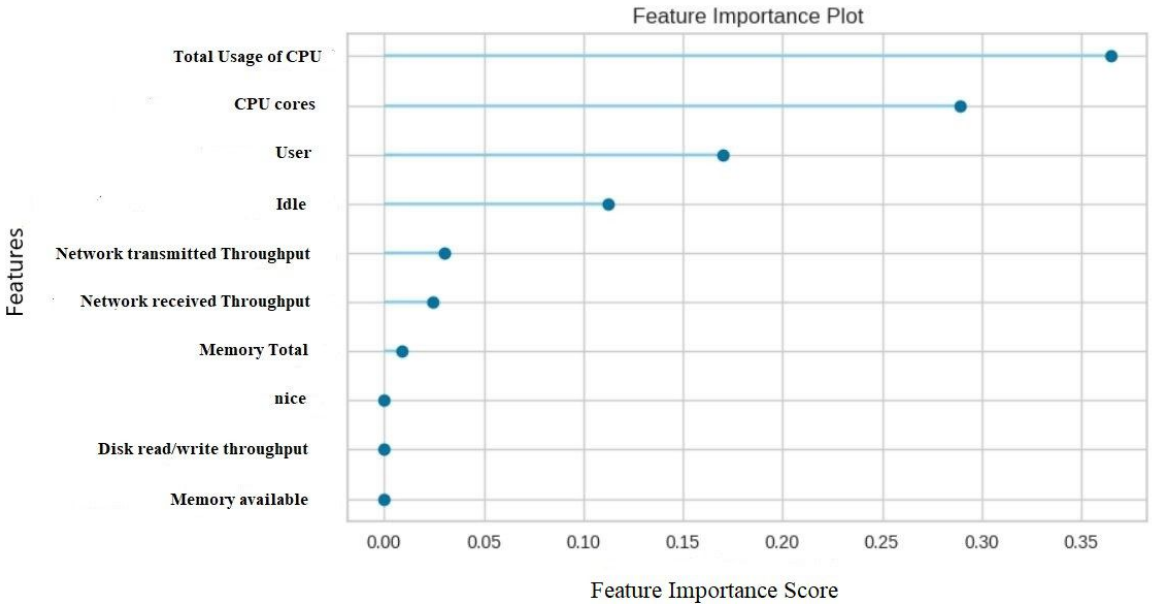


Figure 4.4: Feature importance for the proposed work

Following the generation of the dataset, preprocessing is performed, including normalization, transformation, PCA and outlier removal.

During data pre-processing, unnecessary columns such as system, iowait and memtotal are removed as shown in Table 4.1. Further, the data is classified and as a result, we have determined that the proposed ensemble model performs best with an average accuracy of 82%. The feature importance can be derived from the dataset once the classification result has been obtained. We must begin by defining the optimal model, which we refer to as the proposed ensemble model. The proposed model includes various models including random forest. Random Forest builds a large number of decision trees during training to generate the mode of classes or the average prediction of individual trees. Understanding the value of features of dataset helps to gain insights into the dataset and enhance the model by focusing on the most informative characteristics.

It could improve classification problem feature engineering, model understanding and feature selection decisions. Moreover, the model is plotted according to the importance of the features on the basis of information gain theory. From the feature importance plot as shown in Figure 4.4, we can conclude that with respect to the generated dataset, total usage of CPU is the most important feature, followed by CPU cores and user.

4.2.4 Machine Learning Methods

Various machine learning models are used to classify load on hosts as overloaded, underloaded and balanced. The details of these models are given below:

1. A Random Forest Classifier: It is a method of supervised machine learning that employs decision trees for various applications including classification, regression, and others. Using a randomly chosen training dataset, the RF classifier generates a set of trees of decision. Essentially, it is a collection of decision trees (DT) generated by choose a random portion of the training set.

2. Gradient Boosting Classifier: It is a sequential learning ensemble approach in which the model's efficacy increases over time. Using this method, the model is constructed successively. With the addition of each feeble learner, a new model is developed, resulting in a more accurate estimation of the response of the variable.

3. Decision Tree Classifier: A decision tree is a graphical representation resembling a tree that models decision-making processes, including prospective outcomes, expenses for inputs, and utility factors. This algorithm is relevant to both continuous and categorical variables as outputs and lies under the category of trained learning methods.

4. Extra Tree Classifier: This class implements a meta-estimator that uses averaging to improve predicted accuracy and prevent overfitting by fitting multiple randomized decision trees to different sub-samples of the dataset.

5. Ada Boost Classifier: It integrates multiple classifiers to increase classifier precision. AdaBoost generates a robust classifier by combining several underperforming classifiers, resulting in an excessively accurate classifier. Its fundamental principle is to establish the weights of classifiers and teach the data set in each iteration so that accurate predictions can be made for out-of-the-ordinary observations.

6. K Neighbors Classifier: The KN Classifier is one of the most fundamental algorithms for classification in the field of machine-learning. Among the applications found in the domain of supervised learning are pattern recognition, data extraction, and intrusion detection. It makes no assumptions regarding data distribution.

7. Linear Discriminant Analysis: LDA is a learning with supervision algorithm used for solving classification issues in learning under supervision. It is a method for determining the optimal method for classifying a dataset using a linear combination of features. LDA functions by projecting the data onto a lower-dimensional space that maximizes the separation of various classes. It determines the orientations in the feature space that most effectively separate the various data classes.

8. Light Gradient Boosting Machine: Gradient boosting and tree-based learning are utilized by Light GBM. Unique algorithm Light GBM possesses multiple parame-

ters. The dataset is swiftly expanding. Traditional data science methods battle to supply correct results. Light GBM manages large data sets with minimal memory. Based on Ridge regression, the resulting Ridge Classification converts label data to the interval [-1, 1] and employs regression to solve the problem. Using multiclass data and multioutput regression, it selects the target class with the best value of prediction.

9. Ridge Classifier: Derived from the Ridge regression approach, the Ridge Classifier transforms label data into the range [-1, 1] and employs regression techniques to address the problem. When dealing with multiclass data, It employs multioutput regression and chooses the target class with the maximum predictive value.

10. Naive Bayes Classifier: The naive bayes model is constructed using the principle of Bayes, creating a compilation of classification algorithms. It takes that each pair of classified features has no relationship with the others, a foundational concept shared by every algorithm in this family.

4.3 Methodology

The workflow of the present work is shown in Figure 4.5. The dataset is generated using different numbers of VMs. The extracted feature vector is fed into the various machine-learning models. Further, an ensemble model is proposed and the performance is computed accordingly by predicting the load on the host.

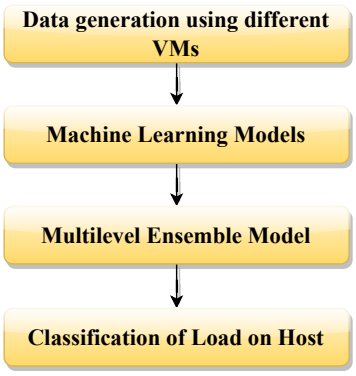


Figure 4.5: Flowchart of Proposed Scheme

4.3.1 Proposed Multilevel Model

The ensemble method is used to manage the model prediction's worst-case scenario. The current study concentrates on the model's false and accurate predictions and a multilevel ensemble model is used to deal with false and accurate predictions. As depicted in

Figure 4.6, a total of seven models, including the RF Classifier, DT Classifier, Linear Discriminant Analysis Classifier, NB, ET Classifier, KN Classifier, and Ridge Classifier, are integrated to enhance accuracy.

Seventy percent (70%) of the dataset is used for training, while the rest of the thirty percent 30% is reserved for testing. The proposed approach comprises three distinct phases, each of which is explained below:

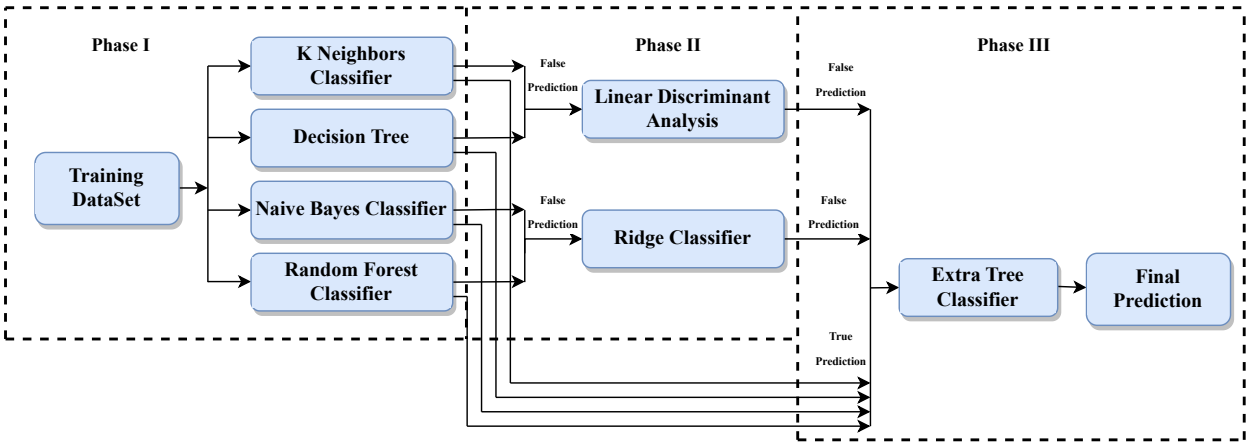


Figure 4.6: Multilevel Ensemble Model

Phase I: The RF Classifier, DT Classifier, KN Classifier, and NB learned from 70% of the data set and generate predictions from 30% of the data.

Phase II: The Linear Discriminant Analysis model is trained using the inaccurate predictions generated by two models, namely the Decision Tree (DT) and K Nearest Neighbor Classifier in Phase I. Ridge Classifier model is trained using the inaccurate predictions generated by two models, namely the RF Classifier and NB, in Phase I.

Phase III: The false predictions from Phase II are combined with the accurate predictions from Phase I. This new combined dataset is used to train the Extra tree classifier model, which makes conclusive predictions. This method refines accurate and inaccurate predictions to produce a model proposal that is precise. The purpose of incorporating accurate predictions into other models is to combat false-positive results. The data is passed through seven models because, ideally, these models will use the data to produce accurate and reliable results.

4.3.2 Case Study

Three types of cases are considered for evaluation: In different cases, there are different numbers of virtual machines running in parallel and two types of platforms are used to run these VMs. Oracle VM VirtualBox has been used to run these VMs. Each VM's

dataset has 11 parameters: user, nice, system, iowait, CPU Usage, idle, write throughput, memory-free and memory available etc. with CPU usage being the target variable. Table 4.5 provides an overview of three cases and the platforms used.

Table 4.5: Different Cases Considered for Evaluation

Case Study	**Number of VMs running in parallel**	**Platform 1**	**Platform 2**	**Software**
Case 1	4 VMs (2 on Platform 1 and other 2 on 2nd Platform)	Processor-11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz	INTEL(R) XEON(R) CPU E5-2650 V2 @ 2.40GHZ	Oracle VM VirtualBox, Python(OpenCV)
Case 2	6 VMs (3 on Platform 1 and other 3 on 2nd Platform)	RAM 12GB	RAM 16GB	
Case 3	10 VMs (5 on Platform 1 and other 5 on 2nd Platform)			

- **Case 1**: In the first case, four VMs run in parallel, two on the first platform and two on the second. Table 8 shows the detailed comparison of different supervised learning models with ensemble model proposed. The models have been compared in various ways like without pre-processing and applying different pre-processing methods on the generated dataset, like normalization, outlier removal, transformation, PCA and feature selection. When outliers are removed and the dataset is normalized simultaneously, the RF, GBC classifier, AB and DT models perform equally well.

- **Case 2**: In the second case, six VMs run in parallel, three on the first platform and three others on the second. Table 9 shows the detailed comparison of different supervised models with the ensemble model proposed. It has been observed that normalizing the dataset enhances the accuracy of the model. Conversely, when PCA is applied to the dataset, the accuracy of the model decreases.

- **Case 3**: In the third case, ten VMs are running in parallel, five on the first platform and five others on the second platform. Table 10 provides a comprehensive comparison of various artificial intelligence models and the suggested ensemble model. The performance of our proposed model is marginally superior to that of all other models, with a maximum accuracy of around 82% after the normalization of the dataset.

  As previously stated, we have presented three cases. We have observed our proposed model performance through different cases by increasing the number of VMs in each case. As per our results, as shown in Tables 8, 9 and 10, it has been seen that increasing the amount of VMs makes our proposed model work better.

4.4 Model Evaluation

4.4.1 Evaluation Parameters

Various metrics, including feature importance, recall, precision, the F1 score, Kappa, and accuracy, are employed to assess the model's performance. This study comprehensively evaluates all these parameters. To investigate the robustness of the proposed model, repeated cross-validation with k folds is utilized. Below, we provide explanations for these metrics:

4.4.1.1 Precision

It assesses the model's ability to correctly predict true classes. Precision is determined by dividing the count of true positive predictions by the total of true positive predictions and fake positives. This precision calculation is based on a specific formula as shown in 4.5.

$$Precision = \frac{TP}{TP + FP} \tag{4.5}$$

4.4.1.2 Recall

Recall quantifies the fraction of predicted positive classes. It is determined by dividing the total number of true positive and deceptive negative events by the number of true events that were positive. The formula for calculating the recall is shown in Equation 4.6.

$$Recall = \frac{TP}{TP + FN} \tag{4.6}$$

4.4.1.3 F1-score

It is a metric that represents the weighted average of recall and precision scores. It is a value that ranges from 0, indicating the poorest performance, to 1, representing the highest performance. The F1 score is a statistical measure that combines both precision and recall to offer a complete assessment.The following Equation 4.7 is used to figure out F1-Score:

$$F1 - score = \frac{2}{(\frac{1}{Precision} + \frac{1}{Recall})} \tag{4.7}$$

4.4.1.4 Cohen's Kappa (Kappa)

It determines how accurate the model's predictions are. The stronger the model, the lower the Kappa value. The kappa statistic developed by Cohen is ideal for addressing multi-class and unbalanced class problems. Cohen's Kappa is calculated using the equation 4.8.

$$Kappa = \frac{p_o - p_e}{1 - p_e} \tag{4.8}$$

where p_o represents the observed agreement and p_e represents the anticipated agreement. It indicates by how much our classifier outperforms a classifier that makes random predictions based on the frequency of each class. Cohen's Kappa is never greater than 1 or equivalent to 1. Values of 0 or less signify an ineffective classifier.

4.4.1.5 Accuracy

Accuracy is a measure of the ratio of correct predictions. It is computed by dividing the sum of true positives (TP) and true negatives (TN) by the total number of events. The calculation of accuracy can be performed using either equation 4.9.

$$Accuracy = \frac{No.\ of\ true\ predictions}{Total\ no.\ of\ predictions} \tag{4.9}$$

4.4.1.6 Matthews correlation coefficient (MCC)

It accounts for true and false positives and negatives in machine learning and is widely recognized as a metric that can be applied even when the class sizes vary. The MCC is a correlation coefficient number between -1 and 1. A flawless forecast has a coefficient of +1, while an average random forecast has a coefficient of 0 and an inverse forecast has a coefficient of -1. Another name for the statistic is the phi coefficient. The MCC can be determined using the following Equation 4.10.

$$MCC = \frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.10}$$

4.4.1.7 Learning Curve

A learning curve is a visual representation of the relationship between a learner's performance and the amount of time necessary to complete an activity. The learning curve for the proposed model is depicted in Figure 4.7.

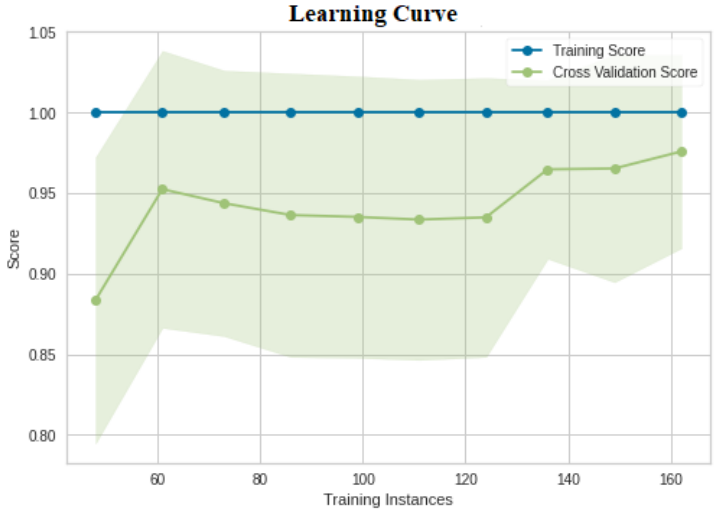


Figure 4.7: Learning Curve of Proposed Model

4.4.1.8 K-fold cross validation

Conducting extensive comparisons is a recommended practice for assessing the performance of a model. One way to achieve this is by performing K-fold cross-validation repeatedely, which involves executing K-fold validation multiple times or increasing the number of comparisons, as discussed by Arlot and Celisse in [105]. Inside it, only K comparisons are generated, with random data allocated for each iteration. This technique is employed to gauge the resilience of machine learning models. In our study, we evaluate the robustness of the best prediction model through ten consecutive executions, commonly referred to as 10-fold cross-validation.

4.4.2 Implementation

4.4.2.1 Algorithm

Algorithm 1 is used to program the proposed approach. The Load Prediction algorithm provides a mathematical explanation for the phases of data processing and prediction. The conventions used in the algorithm are shown in Table 4.6. The Load Prediction Algorithm involved examining three case studies, each with a varying number of virtual machines (VMs). For each VM, various features were extracted from the dataset. Next, the dataset was transformed into three categories: overloaded, underloaded, and balanced. Next, the data was divided, and K-fold cross-validation was conducted. There are two segments in the dataset: the training set and the testing set. Seventy percent (70%) of the data has been allocated for training the models, while the remaining thirty percent (30%) is reserved for testing. The final step involves employing the ensemble model to

obtain the end result.

4.5 Result Analysis, Comparison and Discussion

The performance is evaluated in three different case studies. All the models are evaluated based on three parameters: accuracy, recall and precision.

Table 4.6: Conventions Used

Conventions	**Description**
i	Number of cases
j	The number of feature vectors
X, Y	Input and output space
x, y	Input and target vectors
µ	DataSet after preprocessing
µ1, µ2	Partitions of the data: Training and testing datasets i.e. µ1 and µ2.
M	Model Representation i.e., M: X->Y
M_{EM}	Represents the final multilevel ensemble model created
F_{EM}	Final output of ensemble model
e	Acceptable error
V_m, VM	Virtual Machine
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

In Case 1, it has been observed that after applying normalization to the dataset, the accuracy of models is very similar to the performance of models without preprocessing. However, the accuracy of models decreases after applying PCA to the dataset. NB and KN classifier performance are low compared to all other models, even without preprocessing and with preprocessing of models. However, our proposed ensemble model performance is better than all the applied models.

It provides about 78% accuracy after simultaneously applying outlier removal and normalization to the dataset. Table 4.7 displays the performance of different models.

In Case 2, it has been observed that the models' accuracy increases after the dataset's normalization and the models' accuracy decreases after applying PCA. NB, Linear Discriminant Analysis, Ridge classifier and K neighbors classifier performance are low compared to all other models when PCA is applied to the dataset. During the normalization of the dataset, four models, the RF model, AB, GB and DT model, perform equally well. However, our proposed ensemble model performance is better than these models, even

Algorithm 4.1: Load Prediction Algorithm

BEGIN
for *i=1 to 3, where i is the number of cases (Count of V_m)* **do**
  foreach *V_m = 1 to n, where n is the number of VMs in case i* **do**
    foreach *V_m,$\exists x_j \in V_m$, where x_j is collection of features* **do**
      Transformation of data using following equation:

$$HostStatus = \begin{cases} Overloaded & if\ U_{(i,t)} > T_{ol} \\ Underloaded & if\ U_{(i,t)} < T_{ul} \\ Balanced & if\ T_{ul} < U_{(i,t)} < T_{ol} \end{cases}$$

    end
  end
end
Data Partition or split
K-fold validation (μ_1=random(μ, fraction =0.70));
$\mu_1 = \mu - \mu_1$
μ_1= *Training Set*
$\mu 2$= *Testing Set*
***********Training Models************
M_k, *for* $k \in s$. ($s = M_1, M_2 \ldots . M_n$);
M_k [X-> Y] : $M_k[x_j$ -> $y_j]$, where $(x_j, y_j) \in \mu_{train}$
******** Ensemble and Testing Phase *********
$F_{EM}(x_j) = M_{EM}(y_j')$ *where* $x_j \in \mu_{test}$
*************Error Rate*************
Absolute(y_j-y_j') = e;
Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$;
Precision = $\frac{TP}{TP+FP}$;
END

without preprocessing and with preprocessing of models. Our proposed ensemble model provides maximum accuracy of about 80% when applying outlier removal and normalization simultaneously to the dataset. Table 4.8 displays the performance of different models.

In Case 3, it has been observed that models perform more accurately after applying normalization to the dataset. In this case, the RF, GB, AB and DT models perform more accurately than all others. These models also perform better after applying the transformation to the dataset. NB, Ridge classifier, Linear Discriminant classifier and KN classifier performance decrease after applying PCA and feature selection to the dataset.

In contrast to these other models, the performance of the suggested ensemble approach is relatively preferable, achieving a maximum accuracy of around 82% after normalizing the dataset. The performance of different models is shown in Table 4.9. Figures 4.9, 4.8

and 4.10 illustrate the comparison of various models in terms of their precision, accuracy and recall, respectively.

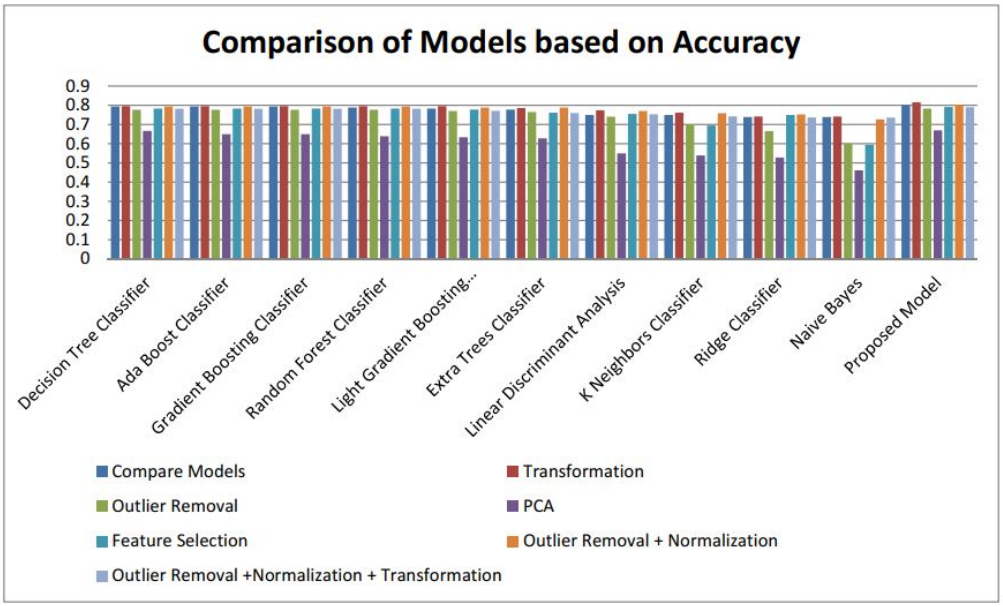


Figure 4.8: Different Models Comparison based on Accuracy

During the training process, it is possible for models to become biased. The SMOTE algorithm can be employed as a means of addressing this particular issue. Another issue that arises in machine learning is the problems of overfitting and underfitting. To mitigate issues related to over-fitting and under-fitting, it is advisable to utilize cross-validation and assess the model's performance on an independent dataset. Consistent high performance indicates that the model is not affected by over-fitting or under-fitting. Overfitting arises when a model captures an excessive amount of information, whereas underfitting occurs when a model lacks sufficient information.

During cross-validation, the model is repeatedly evaluated n times, and its accuracy is measured accordingly. If the recorded accuracy exhibits significant fluctuations, the model may suffer from issues such as overfitting, underfitting, or bias. In this study, repeated K-fold cross-validation is employed to assess the reliability of the correctness of the suggested model, hence demonstrating its resilience against potential issues [105]. The suggested model exhibits a lack of over-fitting, under-fitting, and bias-related concerns. Furthermore, the results of the suggested model demonstrate a significant improvement compared to existing methodologies.

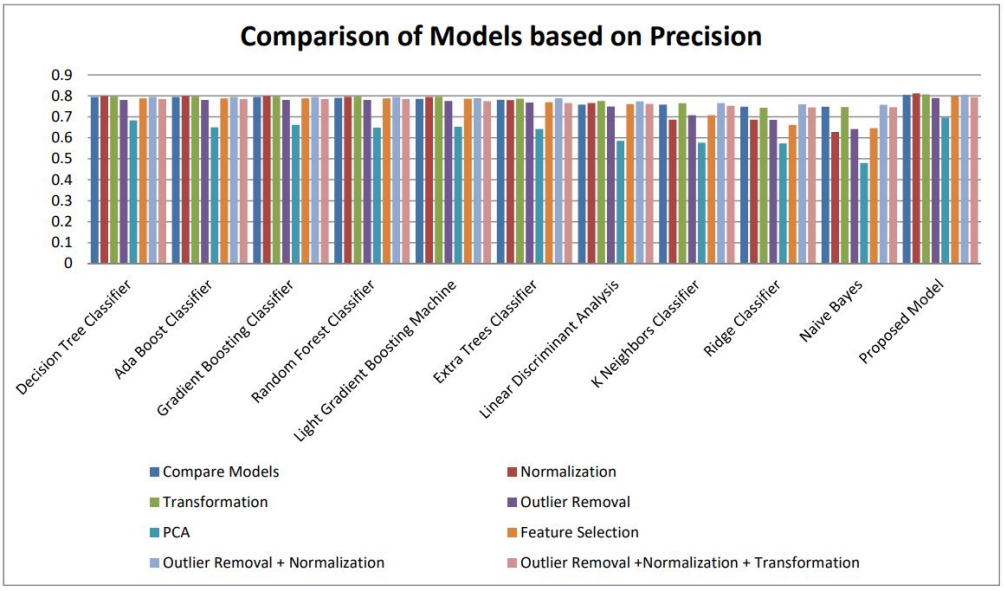


Figure 4.9: Different Models Comparison based on Precision

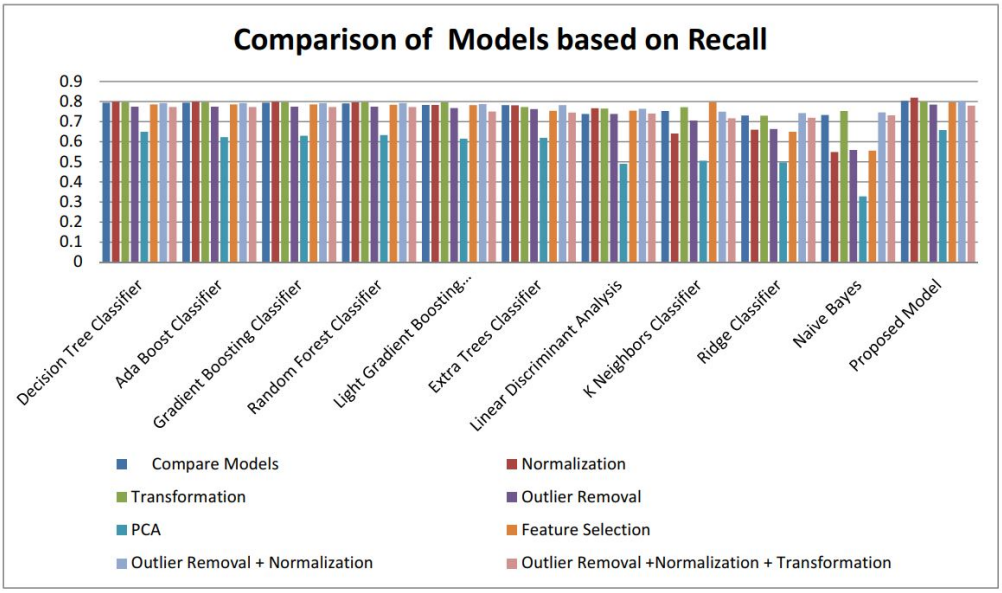


Figure 4.10: Different Models Comparison based on Recall

Table 4.7: Detailed Comparison of Models of Case 1

		Compare Models			Normalization			Transformation			Outlier Removal			PCA			Feature Selection			Outlier Removal + Normalization			Outlier + Normal + Transformation		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.760	0.758	0.734	0.757	0.759	0.758	0.754	0.754	0.756	0.726	0.724	0.731	0.666	0.649	0.623	0.763	0.745	0.768	0.758	0.774	0.758	0.742	0.723	0.735
ada	Ada Boost	0.758	0.758	0.734	0.758	0.758	0.758	0.754	0.754	0.755	0.726	0.725	0.731	0.65	0.63	0.642	0.763	0.745	0.768	0.758	0.763	0.758	0.742	0.723	0.735
gbc	Gradient Boosting	0.758	0.759	0.730	0.757	0.757	0.756	0.754	0.754	0.755	0.726	0.725	0.731	0.639	0.633	0.649	0.763	0.744	0.764	0.758	0.763	0.756	0.742	0.723	0.735
rf	Random Forest	0.758	0.757	0.734	0.758	0.757	0.758	0.753	0.754	0.755	0.725	0.728	0.731	0.65	0.621	0.650	0.763	0.746	0.768	0.758	0.763	0.758	0.739	0.723	0.735
lightgbm	Light Gradient Boosting	0.752	0.753	0.752	0.754	0.741	0.755	0.748	0.743	0.749	0.721	0.718	0.625	0.633	0.615	0.653	0.757	0.742	0.761	0.763	0.758	0.749	0.720	0.740	0.724
et	Extra Trees	0.732	0.734	0.733	0.736	0.728	0.737	0.743	0.747	0.746	0.715	0.713	0.718	0.628	0.619	0.622	0.741	0.714	0.710	0.763	0.752	0.742	0.705	0.736	0.716
lda	Linear Discriminant Analysis (LDA)	0.721	0.663	0.714	0.736	0.685	0.718	0.747	0.756	0.721	0.711	0.708	0.709	0.515	0.489	0.586	0.736	0.715	0.701	0.710	0.724	0.734	0.704	0.710	0.713
ridge	Ridge Classifier	0.648	0.534	0.645	0.728	0.726	0.732	0.727	0.764	0.740	0.641	0.676	0.658	0.544	0.506	0.577	0.724	0.657	0.708	0.748	0.720	0.719	0.656	0.687	0.693
knn	K Neighbor Classifier	0.508	0.485	0.515	0.703	0.683	0.708	0.716	0.739	0.740	0.626	0.614	0.645	0.529	0.497	0.574	0.815	0.611	0.642	0.713	0.723	0.720	0.677	0.689	0.694
nb	Naive Bayes	0.355	0.423	0.415	0.728	0.711	0.715	0.716	0.761	0.722	0.551	0.509	0.601	0.461	0.327	0.481	0.554	0.546	0.646	0.727	0.717	0.717	0.686	0.672	0.676
	Proposed Model	0.760	0.771	0.772	0.767	0.768	0.771	0.763	0.772	0.775	0.745	0.735	0.747	0.677	0.659	0.631	0.750	0.757	0.793	0.780	0.773	0.782	0.754	0.753	0.749

Table 4.8: Detailed Comparison of Models of Case 2

		Compare Models			Normalization			Transformation			Outlier Removal			PCA			Feature Selection			Outlier Removal + Normalization			Outlier + Normal + Transformation		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.799	0.790	0.791	0.780	0.796	0.793	0.763	0.784	0.795	0.776	0.774	0.781	0.666	0.649	0.683	0.783	0.786	0.788	0.794	0.793	0.795	0.782	0.773	0.785
ada	Ada Boost	0.785	0.789	0.788	0.781	0.790	0.793	0.784	0.771	0.785	0.766	0.785	0.780	0.652	0.631	0.682	0.780	0.781	0.786	0.734	0.763	0.795	0.732	0.773	0.785
gbc	Gradient Boosting	0.786	0.780	0.783	0.791	0.793	0.742	0.754	0.734	0.705	0.776	0.742	0.781	0.609	0.613	0.642	0.787	0.783	0.781	0.799	0.773	0.745	0.762	0.783	0.780
rf	Random Forest	0.796	0.796	0.798	0.795	0.798	0.790	0.794	0.795	0.795	0.776	0.774	0.781	0.650	0.623	0.652	0.783	0.786	0.788	0.794	0.793	0.795	0.782	0.773	0.785
lightgbm	Light Gradient Boosting	0.772	0.734	0.792	0.774	0.782	0.724	0.768	0.773	0.749	0.770	0.768	0.676	0.633	0.615	0.653	0.778	0.782	0.787	0.788	0.788	0.789	0.771	0.751	0.775
et	Extra Trees	0.772	0.774	0.773	0.776	0.769	0.777	0.783	0.787	0.786	0.764	0.763	0.769	0.628	0.619	0.642	0.761	0.754	0.770	0.788	0.782	0.799	0.759	0.746	0.766
lda	Linear Discriminant Analysis (LDA)	0.724	0.703	0.734	0.756	0.745	0.767	0.772	0.775	0.780	0.721	0.739	0.788	0.550	0.469	0.586	0.745	0.714	0.761	0.750	0.764	0.775	0.753	0.740	0.762
ridge	Ridge Classifier	0.680	0.588	0.699	0.748	0.726	0.776	0.768	0.789	0.783	0.701	0.716	0.708	0.544	0.506	0.577	0.794	0.697	0.708	0.759	0.749	0.766	0.742	0.717	0.753
knn	K Neighbor Classifier	0.558	0.515	0.565	0.754	0.733	0.759	0.767	0.769	0.771	0.666	0.664	0.685	0.528	0.497	0.573	0.850	0.650	0.662	0.753	0.743	0.760	0.737	0.719	0.745
nb	Naive Bayes	0.405	0.453	0.465	0.748	0.741	0.750	0.755	0.752	0.761	0.601	0.559	0.642	0.461	0.327	0.480	0.594	0.556	0.647	0.747	0.747	0.757	0.736	0.732	0.746
	Proposed Model	0.80	0.811	0.812	0.818	0.818	0.815	0.803	0.812	0.815	0.785	0.786	0.798	0.676	0.658	0.691	0.790	0.797	0.803	0.801	0.813	0.827	0.794	0.785	0.796

Table 4.9: Detailed Comparison of Models of Case 3

		Compare Models			Normalization			Transformation			Outlier Removal			PCA			Feature Selection			Outlier Removal + Normalization			Outlier + Normal + Transformation		
	Model	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.	Accuracy	Recall	Prec.
dt	Decision Tree	0.798	0.790	0.789	0.799	0.813	0.815	0.786	0.767	0.786	0.772	0.770	0.783	0.669	0.640	0.681	0.782	0.785	0.780	0.791	0.773	0.784	0.746	0.713	0.745
ada	Ada Boost	0.789	0.788	0.790	0.812	0.810	0.814	0.769	0.790	0.786	0.705	0.715	0.782	0.650	0.623	0.650	0.788	0.784	0.781	0.748	0.797	0.784	0.781	0.713	0.705
gbc	Gradient Boosting (GBC)	0.792	0.763	0.785	0.800	0.799	0.812	0.776	0.787	0.798	0.756	0.775	0.781	0.650	0.631	0.602	0.783	0.786	0.708	0.794	0.798	0.775	0.783	0.773	0.785
rf	Random Forest	0.789	0.785	0.787	0.790	0.791	0.769	0.790	0.779	0.729	0.767	0.747	0.780	0.659	0.683	0.699	0.753	0.744	0.768	0.774	0.783	0.725	0.702	0.763	0.775
lightgbm	Light Gradient Boosting	0.782	0.780	0.798	0.774	0.781	0.775	0.769	0.764	0.798	0.750	0.784	0.762	0.623	0.625	0.643	0.718	0.788	0.771	0.758	0.727	0.769	0.719	0.720	0.785
et	Extra Trees	0.777	0.742	0.782	0.777	0.781	0.780	0.785	0.773	0.786	0.764	0.765	0.769	0.628	0.619	0.641	0.761	0.754	0.770	0.788	0.782	0.789	0.759	0.746	0.766
lda	Linear Discriminant Analysis (LDA)	0.749	0.719	0.728	0.763	0.764	0.768	0.770	0.726	0.767	0.742	0.731	0.745	0.510	0.489	0.586	0.756	0.755	0.761	0.771	0.765	0.774	0.754	0.740	0.762
knn	K Neighbor Classifier	0.750	0.753	0.758	0.672	0.649	0.685	0.769	0.760	0.762	0.701	0.712	0.718	0.534	0.519	0.550	0.654	0.783	0.728	0.752	0.740	0.763	0.724	0.720	0.754
ridge	Ridge Classifier	0.739	0.735	0.747	0.662	0.642	0.686	0.744	0.732	0.741	0.661	0.662	0.684	0.529	0.496	0.573	0.621	0.650	0.662	0.752	0.743	0.720	0.737	0.759	0.745
nb	Naive Bayes	0.718	0.732	0.749	0.590	0.549	0.626	0.742	0.753	0.749	0.600	0.558	0.641	0.460	0.326	0.480	0.595	0.558	0.647	0.748	0.757	0.759	0.738	0.712	0.776
	Proposed Model	0.802	0.804	0.805	0.821	0.824	0.812	0.816	0.801	0.807	0.784	0.784	0.792	0.672	0.658	0.696	0.791	0.796	0.798	0.804	0.803	0.806	0.790	0.781	0.794

4.6 Summary

This work introduces an innovative reliability framework that encompasses multiple phases of implementation, beginning with the generation of virtual machines via the command line with multiple random settings, followed by the generation of datasets, machine learning techniques, and result analysis. All models are evaluated for their precision, recall, and accuracy. A total of ten machine learning models were employed in order to construct an ensemble model, which ultimately yielded optimal and accurate results for the classification of host loads. It has been observed that applying normalization to a dataset improves the performance of models. Four models, the RF, AB, GB, and DT models, performed equally well in all three case studies during normalization of the dataset. However, compared to these models, our proposed ensemble model performs marginally better, with an accuracy of approximately 82%. The robustness of an ensemble model was then evaluated using the 10-fold cross-validation method [105].

Chapter 5

Migration of Containers on the Basis of Load Prediction with Dynamic Inertia Weight based PSO Algorithm

The purpose of this study is to address the critical challenge of reducing energy consumption in a fog environment with limited resources through effective virtualization techniques. Service providers are increasingly tasked with managing energy usage efficiently, especially due to the high computational overhead of existing virtualization methods, which may not be optimal for minimizing energy consumption in fog devices. With the growing popularity of containers for encapsulating fog services, this study investigates their potential to address energy efficiency issues. The proposed work utilizes an ensemble model for load prediction on hosts, classifying them as overloaded, underloaded, or balanced. A container selection algorithm is employed to identify containers for migration from overloaded hosts. Additionally, an energy-efficient container migration strategy, facilitated by a dynamic inertia weight-based particle swarm optimization (DIWPSO) algorithm, is introduced to meet resource demands. This approach involves migrating containers from overloaded hosts to balance the load and reduce energy consumption at fog nodes. Experimental results demonstrate that this method achieves load balancing with lower migration costs. The proposed DIWPSO algorithm significantly reduces energy consumption by 10.89% through container migration. Furthermore, compared to other meta-heuristic solutions such as PSO (Particle Swarm Optimization), ABC (Artificial Bee Colony), and E-ABC (Enhanced Artificial Bee Colony), the DIWPSO algorithm exhibits superior performance across various evaluation parameters.

5.1 Overview

The Internet of Things (IoT), electronic systems and mobile internet have made it possible for numerous people, devices and objects to be connected to the information space at any time and from any location. Data is being produced in quantities and varieties never before seen [93, 94].

Cisco projects that there will be 3.6 networked devices per person in 2023, up from 2.4 in 2018. There will be 29.3 billion devices with networks in 2023, compared to 18.4 billion in 2018. In tandem with the exponential growth of data volume, the rate of data production is accelerating.

This entire produced data is not useful. Only some of the collected data is useful to consumers. By implementing data processing at the network's interface, only critical and significant information can be transmitted to the cloud server, resulting in a substantial reduction in computational expenses. Both edge computing and fog computing are distinguished by the fact that they relocate data processing in closer proximity to the origin of the generated data. Its primary objective is to decrease the volume of data that is transmitted to the cloud server. It reduces the system's latency, improving its response time. Thus, by bringing data processing closer to the source, industries are enhancing security, as sending all data over the Internet is no longer necessary. Figure 4.1 depicts the layered architecture of fog computing [106].

Fog computing is an intermediate layer between the Internet of Things and cloud computing layers, bringing information closer to the sensors. VM technology bases a significant portion of the current cloud infrastructure and fog computing. Cloud providers deliver these services in the form of VM images, which may contain resources of different volumes and configurations [107].

Recently, VM technology has been indispensable to cloud computing. A novel cloud technology has emerged that operates on containers, providing a highly adaptable and effective method for distributing energy-efficient resources. The acquisition of container-based virtualization has risen by twofold since 2017 due to the rapid development of Docker, Podman, and rkt (a containerisation platform that packages the application and all of its dependencies in the form of a container to ensure that the application operates seamlessly in any environment) [67].

In contrast to alternative paradigms like virtual machines, containers offer several benefits, including rapid deployment, efficient use of resources, and minimal administrative expenses [108]. Additionally, container migration can be employed to optimize resource utilization and service provision, much like virtual machines. The majority of the current study is devoted to VM consolidation [109].

Traditional virtualization technologies may incur quite high management expenses in Fog computing scenarios [96]. One potential remedy to this issue is the Linux Container, which is a lightweight virtualization technology [110]. The container consolidation problem has not been thoroughly investigated as it pertains to a novel technology [111].

Containers can be considered a new revolution in the cloud era since they are lightweight, easier to configure and manage, and can considerably decrease start-up time [112]. The increased popularity of containers is due to their low cost and energy efficiency [53], [113]. Various types of applications are run on these containers. Podman is utilized to run containerized applications on VMs. Heavy applications can be run through containers because of their lightweight nature. Numerous applications have been run through containers, including those related to heavily loaded images, due to their high resolution. The architecture of container-based VMs is shown in Figure 4.2 [114].

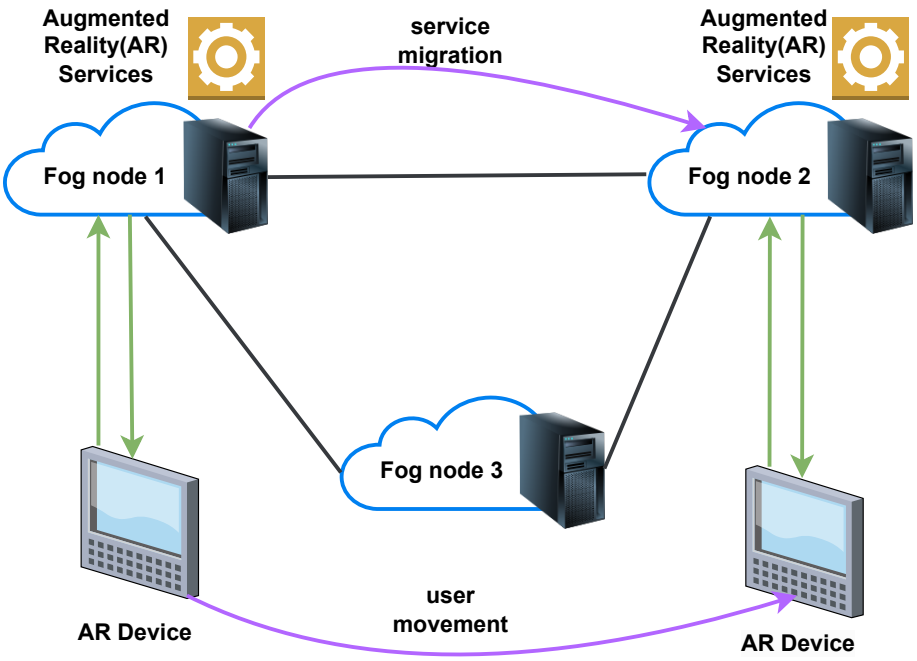


Figure 5.1: Use case of Container Migration in Fog Computing (Mobility Support)

5.1.1 Migration of Containers in a Fog Environment

In contrast to cloud-based environment, the migration of containers is influenced by many factors in a fog environment. Wide area network (WAN) interconnection, total migration time, and heterogeneity of nodes occur when the cloud is extended to the network's periphery. In an industrial setting, personnel employ portable augmented reality (AR) devices, including smart eyewear and tablets, to acquire up-to-date information pertaining to products, instructions, and other pertinent subjects. Augmented reality (AR) devices employ fog computing to augment images while maintaining a constrained low latency, thereby eliminating the need for their offloading to the cloud.

As depicted in Figure 5.1 [115], load-balancing policies that optimize resource utilization

in the fog computing environment and monitor the ongoing fulfilment of application requirements may be enforced via migration between distinct fog nodes or hosts. In this hypothetical scenario, an operator is examining a production facility while using intelligent eyewear that provides instantaneous data regarding the condition of the equipment. The migration of the augmented reality (AR) service is required in this instance to guarantee the accumulation and analysis of images in a timely and uninterrupted manner. To mitigate service degradation, it is imperative to devise an algorithm that effectively manages load distribution on fog infrastructure by discerning the optimal moment to transfer services from a fog node that is excessively encumbered to another.

As container migration within the context of fog computing demonstrates enhanced functionality, its advantages will grow [116]. By migrating fog hosts, the implementation of load-balancing policies enables the continuous optimization of resource utilization in the fog environment. The issue at hand may be mitigated through the implementation of an appropriate migration strategy that transfers the overloaded workload from a fog node to additional hosts. Through the implementation of an adaptive energy-aware computation outsourcing technique, fog and cloud systems can experience a significant reduction in their overall energy consumption.

5.2 System Model

This section demonstrates a system model for container management and load balancing on the hosts. The system model is summarized in Figure 5.2. We proposed a method for managing containers on the hosts to facilitate load balancing and to establish a mapping between the containers and the hosts. Containers are running on various hosts. The running information of hosts is collected, and a dataset is generated using this information, which is shown in Table 6.4.

The proposed ensemble model for load prediction is trained using this generated dataset. We assume that the resource requirements of containers for application requests vary significantly across various time slots. For instance, container 1 is accommodated on PM1, container 2 on PM2 and container 3 on PM2 in Figure 5.2. When the hosts become overloaded, it is necessary to migrate the containers between hosts in order to balance the load on hosts and complete the resource requirements of containers.

The container selection algorithm is used to select the containers with maximum RAM and CPU usage from overloaded hosts. Finally, a decision system based on dynamic inertia weight-based PSO migrates the containers from an overloaded host to another host. For instance, container 3 is migrated from PM2 to PM1 in Figure 5.2 when PM2

becomes overloaded.

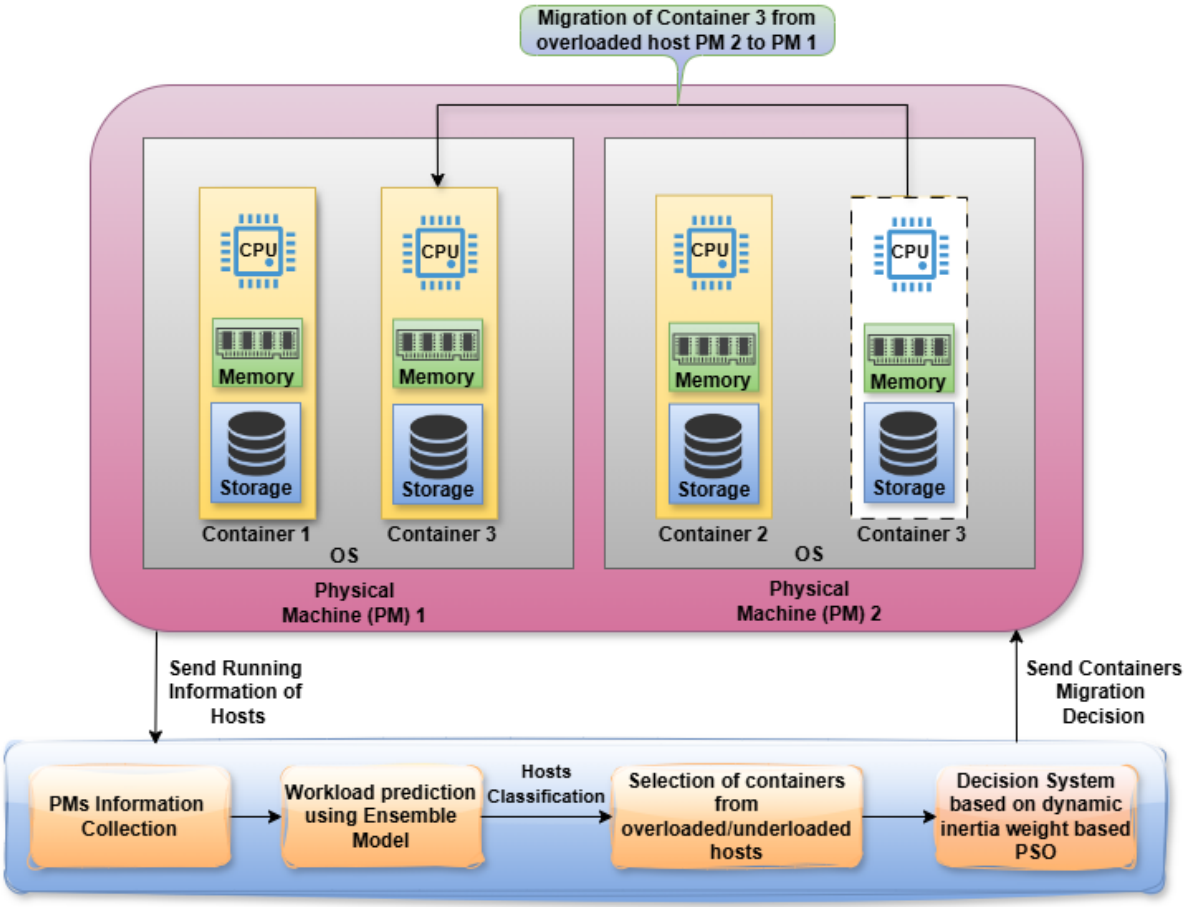


Figure 5.2: System Model

We consider a number of hosts, H = $\{h_1, h_2, h_3, ...h_j\}$, j ∈ N, where the host $h_j \in$ hosts is characterized by the CPU capacity of host h_j^{CPU} and memory capacity h_j^{RAM}. A set of containers is considered as C = $c_1, c_2, c_3, ...c_i$, $c_i \in$ m, where the container $c_i \in$ C is characterized by CPU utilization of container c_i^{CPU}, memory utilization of container c_i^{RAM}, the amount of data transmitted c_i^{data} and image size of container c_i^{img}. To indicate whether a container is placed on a specific host, a set of binary variables is defined as X=$x_{i,j}$, where $x_{i,j} = 1$, which represents that the container c_i is deployed on host h_j; otherwise, $x_{i,j} = 0$.

5.2.1 Load Balancing Model

In order to prevent hosts from becoming overloaded or underloaded, container placement and migration can be implemented across the hosts, considering that the load on each host could be different. An overview of the notations of the variables is presented in Table 5.1. Prior to assessing the load of hosts, the load of containers is evaluated as shown

in Table 5.2. Each host and container consume a unique quantity of system resources. The memory and CPU utilization requirements of a container c_i are c_i^{CPU} and c_i^{RAM}, respectively, where i = 1, 2,...,m and $c_i \in$ C. Let h_j^{CPU} and h_j^{RAM} represent the current CPU and memory load of the host h_j. It is assumed that the load on the host h_j and container $c_i$is determined by the utilization of the CPU and memory, as expressed by the equations 5.1 and 5.2, respectively.

$$L_{h_j} = h_j^{CPU} + h_j^{RAM} \tag{5.1}$$

$$L_{c_i} = c_i^{CPU} + c_i^{RAM}, c_i \in C \tag{5.2}$$

The containers that are deployed on host h_j determine the load of that host L_{h_j}. After placing container c_i, the total load of host h_j can be described using equation 5.3.

$$L_{h_j} = L_{h_j} + \sum_{c_i \in C} L_{c_i}, h_j \in Hosts \tag{5.3}$$

The average load of hosts is calculated using equation 5.4.

$$\bar{Load} = \frac{1}{N} \sum_{h_j \in Hosts} \sum_{c_i \in C} (L_{h_j} + (c_i^{CPU} + c_i^{RAM})) \tag{5.4}$$

The main aim is to minimize the load imbalance. The load balance among the hosts can be given by equation 5.5.

$$L^{bal} = \sum_{j=1}^{N} \frac{(L_{h_j} - \bar{Load})^2}{N} \tag{5.5}$$

where N is the total number of hosts and $\bar{Load}$ is the mean load of all the hosts.

5.2.2 Cost of Container Migration

In order to calculate the cost of container migration, the duration of the container's migration is taken into account; this duration is primarily determined by the volume of data and the transmission speed of the container from one host to another.

$$T_i^{Mig} = \frac{c_i^{img} + c_i^{data}}{NetworkBandwidth(NB)} \tag{5.6}$$

where c_i^{data} represents the information that needs to be transmitted to other host in order to execute the application within the container c_i. Since the container migration can be performed at the same time, the migration time of all hosts N can be determined by the

Table 5.1: Notations of the variables used in proposed algorithms

Symbol	Definitions
c_i^{CPU}	CPU Utilization of the i_{th} container c_i
c_i^{RAM}	Memory Utilization of the i_{th} container c_i
h_j^{CPU}	CPU Utilization of the j_{th} host h_j
h_j^{RAM}	Memory Utilization of the j_{th} host h_j
H	Hosts
N	The number of Hosts
C	Containers
m	The number of Containers
L_{left}	Variable used to maintain the list of hosts
h_{min}	Host with minimum load
$y_{i,j}$	Binary variable represents whether i_{th} container c_i on the j_{th} host should be migrated or not
$\overline{Load}$	Mean load of all the hosts
L^{bal}	Load balance among all the hosts
L_{h_j}	Load on the j_{th} host h_j
L_{c_i}	Load on the i_{th} container c_i

maximum one, i.e.,

$$T^{Mig} = \max_{c_i \in C} \{ z_i^{j,\hat{j}} T_i^{Mig} \} \tag{5.7}$$

where $z_i^{j,\hat{j}}$ represents that the container c_i is migrated from host h_j to host $h_{\hat{j}}$.

5.2.3 Problem Definition

The problem formulation for container management is as specified below. The first step involves conducting an evaluation to determine whether the host is in a balanced or overloaded state. The migration of containers from one host to another becomes necessary if the hosts become overloaded. The primary objective is to reduce the load imbalance and migration costs of containers. The optimization issue may be expressed as follows:

$$\min f = \alpha T^{Mig} + \beta L^{bal} \tag{5.8}$$

$$min(L^{bal}), \tag{5.9}$$

$$s.t. \sum_{c_i \in C} x_{i,j} c_i^{CPU} \leq H_j^{CPU}, \forall N \tag{5.10}$$

$$\sum_{c_i \in C} x_{i,j} c_i^{RAM} \leq H_j^{RAM}, \forall N \tag{5.11}$$

$$\sum_{c_i \in C} x_{i,j}(c_i^{img} + c_i^{data}) \leq H_j^{storage}, \forall N \tag{5.12}$$

$$\sum_{h_j \in Hosts} x_{i,j} = 1, \forall i \tag{5.13}$$

In Equation 5.8, $\alpha + \beta = 1$, where α is the weight of migration time and β is the weight of load imbalance. The capacity constraints of the CPU, memory, and storage are represented by equations 5.10, 5.11, and 5.12, respectively. A container needs to be placed on a host, as indicated by equation 5.13.

5.3 DataSet and Methods

This section provides an overview of the generated data set and describes the methodologies utilized in the proposed work.

5.3.1 DataSet

To develop a deeper comprehension of host workload, the dataset is generated by deploying ten VMs. These machines were configured with varying randomized parameters, including CPU usage, core count, RAM, allocated and available memory, disk I/O and network I/O. The range of parameters utilized to construct the VMs is detailed in Table 4.3. Furthermore, Table 6.1 provides insights into the platform and software used to generate the dataset, as well as for experimentation purposes.
The proposed model has been developed for a container-as-a-service (CaaS) environment where applications run on containers. On VMs, these containers execute various types of applications. Table 6.4 illustrates sample of the generated dataset.

5.3.2 Proposed Approach

Our framework consisted of a different number of phases for addressing the issue of energy-efficient container consolidation:

- To predict the load on the host, an ensemble model is utilized, which classifies the hosts into three categories: overloaded, underloaded and balanced.
- The situation of container migration from one host to another is identified.
- To address the issue, the containers that are needed to be migrated are selected.
- The determination of the destination host for the migration of containers using the proposed dynamic inertia weight-based PSO.

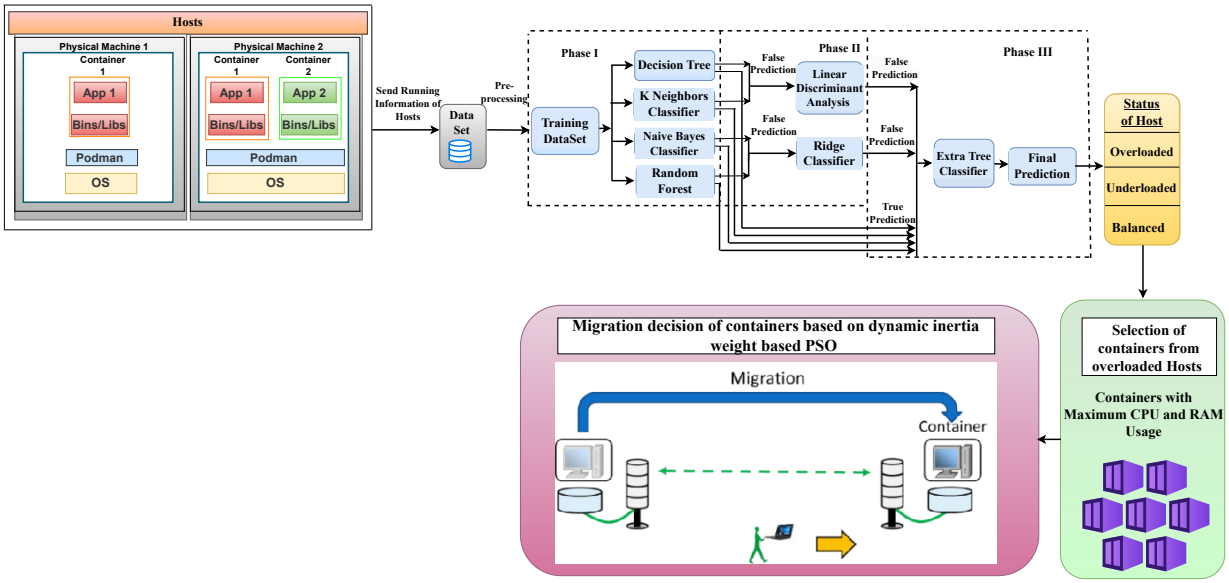


Figure 5.3: Proposed approach for load balancing using container migration

The proposed methodology for load balancing using container migration is illustrated in Figure 5.3. Numerous applications related to IoT devices, such as smart surveillance systems and healthcare monitoring applications, including video processing and object detection, are executed in the form of containers using podman on hosts.

Table 6.4 shows a snapshot of the generated dataset. The generated dataset included ten performance parameters that provide running information about the hosts. After that, different pre-processing methods are applied to the generated dataset, like normalization, outlier removal, transformation, Principal Component Analysis(PCA) and feature selection. An ensemble model is proposed for load prediction, and it is trained using the generated dataset. This ensemble model classifies hosts as either overloaded, underloaded, or balanced. A container selection algorithm is employed to prioritize containers with the highest CPU and RAM utilization for migration from overloaded hosts to another host.

Table 5.2 contains the container running information. Finally, we implemented the proposed dynamic inertia weight-based PSO algorithm to migrate the containers from overloaded hosts to other hosts, thereby achieving load balancing and reducing the host's energy consumption.

5.3.2.1 Detection of load on the host

Various machine learning models are employed to categorize host load as overloaded, under-loaded, or balanced. The ensemble method is employed to address the model pre-

Table 5.2: Running Containers with Memory and CPU Information

ID	NAME	CPU %	Mem_Usage/Limit	MEM%	PIDS	CPU TIME	AVG CPU%
0c09145d6b25	confident_blackburn	26.27%	1.819GB / 5.453GB	33.36%	6	1m49.77232s	20.95%
0f39e17f2b74	recu_chandrasekhar	0.75%	589.8kB/5.453GB	0.01%	1	54.765295s	5.86%
1d6e70a47630	thirsty_wing	0.72%	798.7kB/5.453GB	0.01%	1	9.197809s	1.72%
2af6d885dcf2	cool_galileo	2.47%	2.122MB/5.453GB	0.04%	1	42.553347s	8.66%
6bfc72873c57	funny_blackwell	28.42%	701.3MB/5.453GB	12.86%	6	3m59.594865s	25.46%
86244572dfc6	sweet_khayyam	2.51%	9.286MB/5.453GB	0.17%	1	2m20.444696s	16.01%
95a50bfa472d	bold_dubinsky	0.32%	630.8kB/5.453GB	0.01%	1	31.171776s	3.18%
a70b3d382d10	zen_shaw	21.86%	10.96MB/5.453GB	0.20%	1	5.051081s	0.91%
be1cd1023936	magical_almeida	26.62%	24MB/5.453GB	0.44%	1	1m45.463995s	20.84%
fd645e6f06e8	stoic_almeida	33.64%	1.153GB/5.453GB	21.14%	6	2m8.105746s	23.49%

diction in its worst-case scenario. The development of a multilevel model to account for both accurate and inaccurate predictions. As depicted in Figure 5.3, seven models—RF Classifier, DT Classifier, Linear Discriminant Analysis Classifier, NB, ET Classifier, KN Classifier, and Ridge Classifier are combined to improve accuracy. The training set utilizes 70% of the dataset, while the testing set is allocated 30%. The proposed model consists of three distinct phases, each of which is described in detail below :

Phase I: In this phase, 70% of the dataset is used to train the RF Classifier, DT Classifier, KN Classifier, and NB, which generate predictions using the remaining 30%.

Phase II: It involves training the linear discriminant analysis model using the false predictions of two models (DT and KN Classifier) from Phase I. For training the Ridge Classifier model, the false predictions of two models (RF Classifier and NB) from Phase I are utilized.

Phase III: The inaccurate forecasts generated in Phase II and the precise predictions generated in Phase I are merged in Phase III. The final predictions are generated by the Extra Tree classifier model, which is trained on this merged new dataset. By refining true and false predictions, this method generates a model proposal that is precise. By providing genuine prediction as an input for other models, false-positive results can be mitigated. The data is passed through seven models in an effort to ensure that the results produced by these models are dependable and precise.

Utilizing a supervised ensemble model, the underloaded or overloaded status of a host has been determined. The models have been compared using a variety of approaches, including those that did not involve pre-processing and those that utilized various pre-processing techniques, such as feature selection, normalization, outlier removal, transformation and principal component analysis. Following the implementation of these pre-processing methods, ensemble learning is implemented by feeding the previously mentioned supervised machine learning algorithms on data set. After that, the ensemble model that was proposed is implemented on each host in order to ascertain whether it is overloaded, underloaded or balanced. Figure 5.3 illustrates the methodology utilized for

load prediction on the host along with three phases. It depicts container migration along with load prediction on the host utilizing dynamic inertia weight-based PSO.
When the container selection algorithm detects that a host has become overloaded (OL), it proceeds to identify the specific containers that are responsible for the overload of host. These containers are migrated, thereby decreasing the host's latency. However, in the case where the host is underloaded (UL), the container selection algorithm notifies the consolidation module of the host IDs of every container that is running on that host. The purpose of this communication is to ascertain which containers are appropriate for migration. Furthermore, all containers remaining on the UL host are migrated to new destinations (hosts) if the consolidation module is capable of identifying such hosts. This allows the UL host to be shut down with minimal load. Additionally, this UL host has the capability of receiving excess containers from another host.

5.3.2.2 Containers selection for migration

When containers cause a host to become overloaded or underloaded, the suggested algorithm for container selection transfers them to a different host subsequent to designating the overloaded or underloaded host. To ensure the success of container migration, a variety of factors must be taken into account. Despite the fact that numerous containers are operational on hosts, the migration of every single container from that host would incur exorbitant expenses. As a result, the effectiveness of resource utilization will be compromised. Consequently, a strategy should be formulated to decrease superfluous energy consumption. As a result, we have put forth a container selection algorithm which operates on the subsequent criteria:

1. Containers with maximal CPU usage.
2. Containers with the highest memory utilization. These containers will be added to the migration list and evaluated for possible migration to another host with status as underloaded.

5.3.2.3 Container placement using dynamic inertia weight based PSO

After obtaining the specified containers for migration from the migration list, a new host, or host ID, is assigned to these containers. Selected containers for migration are placed on underloaded hosts. As a swarm intelligence algorithm, PSO iteratively seeks the optimal global position. During the iterative process, the position of each particle in the swarm is modified until a solution that is close to optimal is found. The positions are modified in accordance with the velocity of the particles, which is modified throughout the PSO iterations in accordance with equation 5.14. The change in velocity from the current

value V_{iter} to the new value V_{iter+1} is influenced by the current particle position X_{iter}, relative to both the optimal particle position (pBestPosition) and the optimal position globally (gBestPosition) within the swarm.

As illustrated in equation 5.15, the current location of the particle X_{iter} can be revised to X_{iter+1} using the value of V_{iter+1}. The real numbers produced by these equations are suitable for continuous search space problems.

Conversely, certain problems possess discrete search domains, which led to the development of a discrete binary iteration of PSO [117]. The proposed method aims to provide a near-optimal solution to the container placement problem by mapping the migrated container to the available VMs or PMs. As a result, the proposed work will utilize an approximation to transform the real numbers into integer ones, which correspond to the indexes of the available PMs. Thus, prior to being added to the current position, V_{iter+1} will undergo a rounding process to the nearest integer. Furthermore, in order to guarantee that the newly created position accurately reflects a valid PM index, a stringent constraint is implemented, as illustrated in equation 5.16. In this constraint, i_0 and i_n denote the indexes of the initial and final available PMs, respectively.

$$V_{iter+1} = (w * V_{iter}) + (r_1 * c_1 * (pBestPosition - X_{iter})) + (r_2 * c_2 * (gBestPosition - X_{iter})) \tag{5.14}$$

$$X_{iter+1} = X_{iter} + V_{iter+1} \tag{5.15}$$

$$i_0 \leq X_{iter} \leq i_n \tag{5.16}$$

The main objective of the proposed modification is to enhance particle velocity through the implementation of the inertia weight strategies shown in Table 5.3.

Table 5.3: Selected Inertia Weight for proposed dynamic inertia weight based PSO algorithm

SN	Name of Inertia Weight	Formula of Inertia Weight
1	Chaotic Inertia Weight [118]	$z = 4 * z * (1 - z)$ $w = (w_1 - w_2) * \frac{MAX_{iter} - iter}{MAX_{iter}} + w_2 * z$
2	Random Inertia Weight [119]	$w = 0.5 + \frac{rand()}{2}$
3	Constant Inertia Weight [120]	w = c c = 0.7 (considered for experiments)
4	Linear Decreasing Inertia Weight [121]	$w_k = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} * k$

5.4 Experimental Setup, Results Analysis and Discussion

This section provides an overview of the algorithms that have been incorporated into the proposed container consolidation framework. In addition, experimental findings and analyses follow to conclude the discussion.

5.4.1 Experimental Setup

We ran the experiments on two different machines. Both have a 2 TB hard drive. One with 12 GB of memory and the other with 16 GB of memory using Python and Keras, as shown in Table 6.1. The dataset has 10,000 records with ten performance parameter values, which can be seen in Table 6.4. These values were used to train a model for the prediction of host load using an ensemble model.

5.4.1.1 Detection of Load Status Using Ensemble Approach

Average Load of Hosts $\bar{Load}$ is used as threshold to determine whether a host is overloaded, under-loaded and balanced as shown in equation 5.17 respectively .

$$HostStatus = \begin{cases} Overloaded & \text{if } L_{h_j} > \bar{Load} \\ Underloaded & \text{if } L_{h_j} < \bar{Load} \\ Balanced & \text{if } L_{h_j} == \bar{Load} \end{cases} \tag{5.17}$$

The CPU consumption and memory utilization of the host h_j is regarded as vital components for the calculation of load on the host.

5.4.1.2 Selection of Containers for Migration using Select-Containers Algorithm

The procedure for selecting containers for migration is depicted in Algorithm 1. Since as the workload of the hosts differ, our objective is to identify the overloaded hosts whose workload exceed the threshold of the mean load. In particular, for each host h_j, we determine whether $L_{h_j} > \overline{Load}$, before selecting the list of containers to be migrated based on the load of $\sum_{i=1}^{m} y_{i,j} L_{c_i}$. Notice that $y_{i,j}$ represents whether the i_{th} container c_i on the j_{th} host should be migrated or not; hence, the term $\sum_{i=1}^{m} y_{i,j} L_{c_i}$ indicates the amount of load on the host for migration.

Algorithm 5.2: Select-Containers

Input: c_i^{CPU}, c_i^{RAM}, h_j^{CPU}, h_j^{RAM}, H, N, C, m
Output: Containers List need to be migrated
Calculate the load L_{h_j} by using equation $L_{h_j} = L_{h_j} + \sum_{c_i \in C} L_{c_i}$, $h_j \in$ Hosts
Calculate $\overline{Load}$ by using equation $\bar{Load} = \frac{1}{N} \sum_{h_j \in Hosts} \sum_{c_i \in C} (L_{h_j} + (c_i^{CPU} + c_i^{RAM}))$
for *j = 1 to N* **do**
    if $L_{h_j} > \overline{Load}$ **then**
        $y_{i,j} = (L_{h_j} = (L_{h_j} - \sum_{i=1}^{m} y_{i,j} L_{c_i}))$
    end
end
return Y

5.4.1.3 Proposed Dynamic Inertia Weight Based PSO for Container Migration

Each active host should be utilized optimally while operating at maximum capacity in order to achieve the goal of minimizing the number of active hosts during the placement procedure. The total utilization ratio of resources for all containers deployed on the host, however, must not fall below the threshold for resource utilization on each host. Let c_i be the i_{th} container that is prepared for placing, where i=1,2,...,m. Let H_j be the j_{th} host that is the host for containers,where j=1,2,...,N. Assuming a discrete PSO, the problem definition is as follows:

1. N represents the quantity of particles.
2. m represents the number of dimensions in the space occupied by a particle.

Every host is represented as a particle and every container is represented as one of the particle's dimension elements. Hence the j_{th} particle at iteration **iter** is represented as $X_{iter}^{j} = x_{iter}^{j1}$, x_{iter}^{j2},...,x_{iter}^{jN}), where $x_{iter}^{id} \epsilon$ (0,1), to be represented a result of container placement residing in H_j. If the d_{th} of container is placed in the H_j, there is x_{id}^{iter} is 1 and 0 otherwise. The velocity value is represented by the placement of the d_{th} container of particle 'j' at iteration 'iter'. As denoted by f_n, the Equation 5.8 presents fitness function. When the V_{iter}^{j} ceases to update, it indicates the addition of containers to the H_j. The completion of the procedure will occur when every container selected for migration from an overloaded host is hosted on another host. The solution of the placement problem is represented as a set H_j (H_1, H_2,...,H_k), with k $\leq$ N. The procedure of the proposed dynamic inertia weight-based PSO for the migration of containers is depicted in Algorithm 2. A summary of the variable notations is presented in Table 5.4.

Table 5.4: Notations of the variables used in dynamic inertia weight based PSO algorithm

Notation	**Definition**
H	Hosts
Ex-H	Excluded hosts in the container placement
C	Containers
MD	Migration Decision
N_p	Population total (particles quantity)
N_d	Number of migrated containers (quantity of dimensions)
N_t	The quantity of iterations or repetitions
W_{min}	Value of the minimum inertia weight coefficient
W_{max}	Value of the maximum inertia weight coefficient
c_1, c_2	cognitive parameter (Acceleration constants
r_1	A random number between zero and one
r_2	A random number between zero and one
X_{min}, X_{max}	Positional constraints for each particle
V_{min}	minimum velocity for a particle
V_{max}	maximum velocity for a particle
p	a particular particle within the swarm
P	Swarm of Particles
F_n	value of a particle's fitness function
X_0	initial position of the particle
iter	the present iteration
V_0	initial velocity of the particle
X_{iter}	present position of a particle
X_{iter+1}	position of the particle in the following iteration
V_{iter}	present velocity of a particle
V_{iter+1}	velocity of particles during the succeeding iteration
pBestValue	Optimal F_n value of a particle
pBestPosition	The coordinates of a particle that produce the greatest value of F_n
gBestValue	F_n with the greatest value among all particles
gBestPosition	Best position for containers among the swarm

5.4.1.4 Modification of number of Hosts

We can predict the CPU utilization of the hosts in future time slots, which provides guidelines to estimate the load balance information of the hosts. In our early work, we developed a multivariate time series ensemble model for load prediction on hosts using anomaly detection techniques to predict CPU utilization in the near future [122]. In this paper, we predict resource utilization for container management.

Based on the predicted outcomes, the number of hosts that must be increased to support the load of running containers can be determined. Our objective is to identify the optimal number of hosts H', taking into account the predicted load $\mathbf{Load}'$ and the existing host count N, in order to serve the applications contained within the containers. By calculating the mean predicted load, $\bar{Load}'$ across all hosts, it is possible to determine whether it exceeds the high threshold $Load_{high}$ or falls below the low threshold $Load_{low}$. If $\bar{Load}'$ is not excessively high, $\bar{Load}' > Load_{high}$ and $\bar{Load}' < 90\%$, the containers will be migrated

from one host to another in an effort to achieve load balancing across all hosts. If $\bar{Load'}$ exceeds 90%, the number of nodes will be doubled. Conversely, if $\bar{Load'} > Load_{Low}$, a reduction in the quantity of hosts will occur.

5.4.2 Experimental Findings and Analysis

This section presents the parameters used for the evaluation and performance evaluation of the container placement and container migration algorithms that have been proposed. The impact of the algorithms proposed in subsection 5.4.1.2 and 5.4.1.3 has been evaluated, taking into account experimental results. Finally, an analysis was conducted to compare the CPU utilization, frequency, energy consumption and temperature of hosts prior to and following the container migration process. Two heuristic algorithms developed by PSO and ABC to solve optimization problems are compared to the proposed algorithms for validation.

5.4.2.1 Parameters setting for evaluation

The performance of the proposed algorithms is evaluated in the subsequent experiments using the generated datasets listed in the Table 6.4. The running information of every container for a period of seven days comprises the container ID, CPU and memory information as shown in Table 5.2.

The number of containers chosen for performance evaluation lies in the range of 5 to 55. The specified range for the bandwidth between two hosts is 150 MB/s to 250 MB/s. The container's data size ranges from 50MB to 2GB. The numerous applications are executing on containers, making them fluctuate from underloaded to overloaded. For instance, applications such as video processing, CNN, and object detection cause the containers to become overloaded beyond the threshold values. The parameters α and β for the objective function shown in Equation 5.8 are specified as 0.2 and 0.8. Furthermore, we define the minimum and maximum load as $Load_{low}$ and $Load_{high}$, with threshold values of 0.3 and 0.7, respectively.

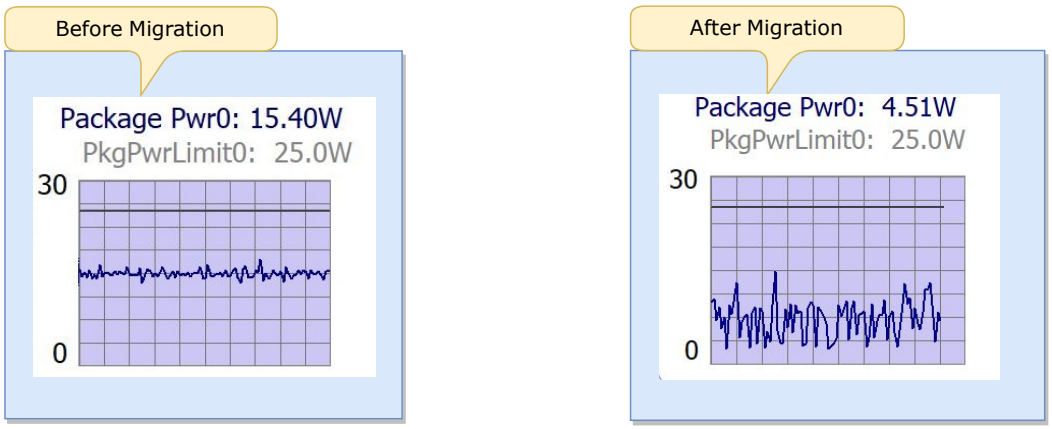
5.4.2.2 Performance Evaluation

The outcomes were acquired through the use of Intel Power Gadget version 3.6. The software is comprised of a multitude of components, drivers, and libraries that retrieve and modify the energy counter of the processor in order to provide precise values for power consumption (in watts), temperature (in degrees Celsius), and frequency (in gigahertz). Additionally, it permits the recording of power, frequency and other measurements in

CSV format.The performance of the proposed method was assessed by utilizing a host overload/underload detection algorithm to identify the OL host.

After obtaining the OL host, the determination was made regarding which containers, out of all those on the OL host, would be migrated to the new host. Subsequently, the OL host migrated the containers to the destination host. Initially, there are 55 operational containers present on our OL host; following container consolidation, the number of containers drops to 30. Table 5.2 presents an illustrative instance of utilizing containers to execute complex applications, including details regarding memory and CPU utilization. Following the evaluation of container migration results, it was ascertained that the host operates as a balanced host. As mentioned earlier, additional parameters were measured and compared at each time stamp prior to and subsequent to container migration.

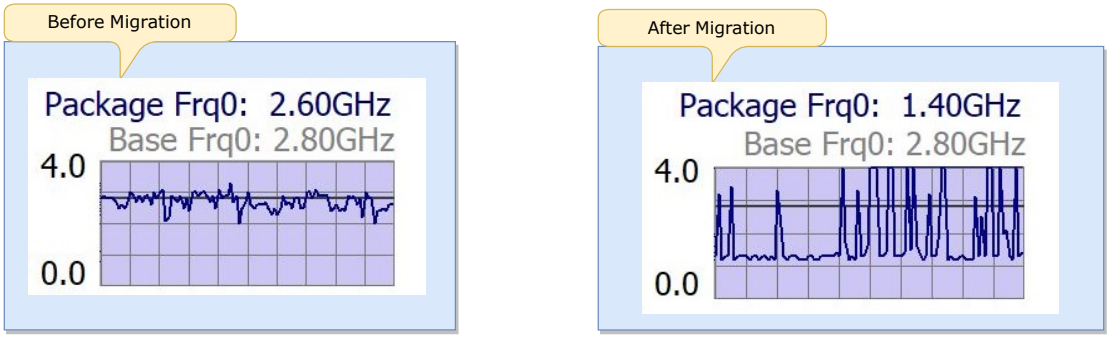
1. **Energy Consumption:** It signifies the estimated power usage of the central processing unit (CPU) cores. These facilitate the process of power profiling. The graph in figure 5.4 illustrates the Wattage-based packet energy consumption (PKG), which consists of the energy utilized by the graphics core of an Intel processor (GT energy), the processor energy and Intel Architecture (IA) energy. Prior to the implementation of the container migration process, the average total power consumption of the host consistently exceeded 15.40W, as illustrated in figure 5.4a. On the contrary, the total power consumption decreased substantially to 4.51W for a considerably reduced duration subsequent to the migration of 25 containers to alternative hosts that were either underloaded or balanced as shown in figure 5.4b. Container migration has led to a substantial reduction in energy consumption, achieving a decrease of 10.89%. Along with the processor energy, the IA energy is also decreased.



(a) Power Consumption Before Migration (b) Power Consumption after Migration

Figure 5.4: The variance in energy/power consumption of the OL host prior to and following container migration

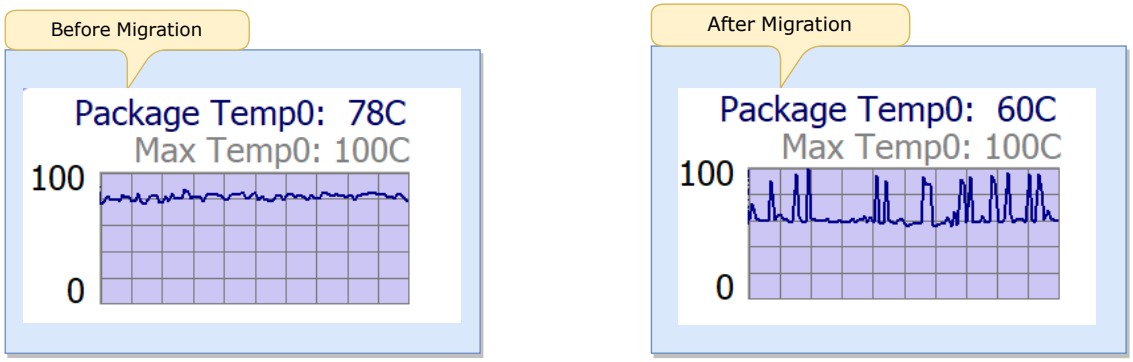
2. **Frequency:** The frequencies of the graphics processing unit (GT) and processor (IA, Intel Architecture) are specified. Prior to the execution of the suggested container migration, the processor oscillated at 2.60 GHz for a considerable duration (as illustrated in the graph, its default clock speed was 2.80 GHz). The overhead was typically incurred when the clock speed was decreased to 1.40 GHz below the base clock frequency subsequent to the migration of the containers. Figure 5.5 depicts the utilization of frequency prior to and following container consolidation.



(a) Frequency Before Migration (b) Frequency after Migration

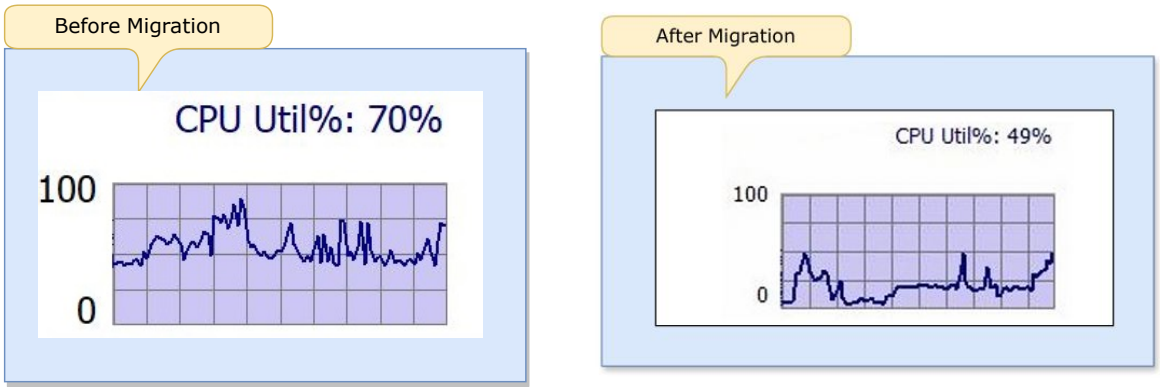
Figure 5.5: The variance in frequency of the OL host prior to and following container migration

3. **Temperature:** Additionally, temperature is a significant factor in determining consumption of energy. Higher temperatures result in increased energy consumption by the host, which ultimately contributes to a rise in the cost of energy consumed. The temperature variation prior to and subsequent to container consolidation is illustrated in figure 5.6. Prior to container migration, the temperature of the host remained high at 78°C for a significant duration. Nevertheless, the temperature fell precipitously to the range of 60 °C immediately following container migration.



(a) Temperature Before Migration (b) Temperature after Migration

Figure 5.6: The variance in Temperature of the OL host prior to and following container migration



(a) CPU Utilization Before Migration (b) CPU Utilization after Migration

Figure 5.7: The variance in CPU Usage of the OL host prior to and following container migration

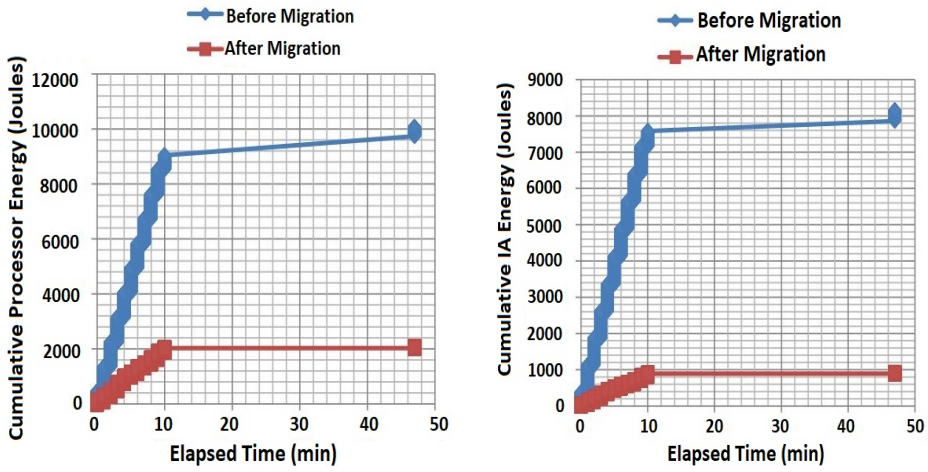
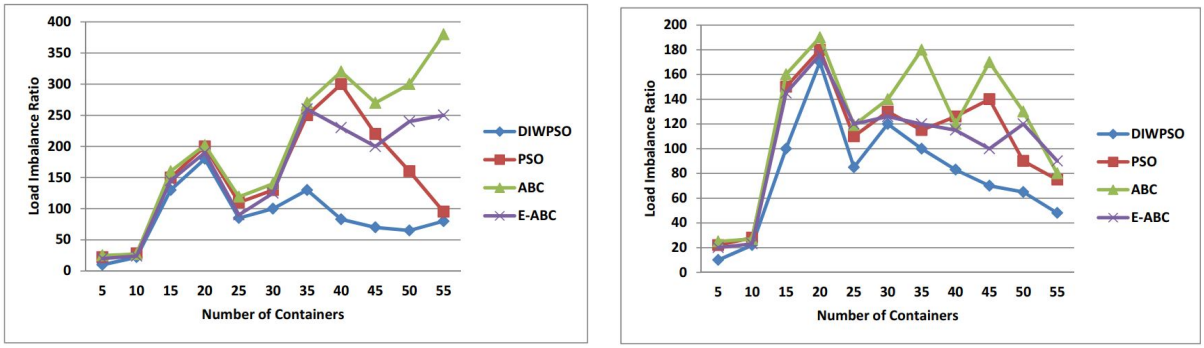


Figure 5.8: Cumulative consumption of energy of the OL host prior to and following container migration

4. **CPU Utilization:** CPU utilization is further diminished through container migration. In the past, the distribution of containers running on a specific host was limited to that host, resulting in increased amounts of CPU usage, clock speed, and memory consumption. As a consequence of the migration of containers hosting resource-intensive applications, the CPU utilization decreased. Now, if a container requires additional resources in the future to complete a specific task, this can be readily accomplished by utilizing available resources. In terms of CPU utilization, Figure 5.7 illustrates the reduction of host overhead.

   As depicted in figures 5.4–5.7, there is a decrease in power and frequency during and following container migration. Following container migration that results in the host becoming overloaded, there is a substantial reduction in the host's overall power

consumption. Additionally, the clock speed is regulated; it approaches the basal clock speed following migration. Furthermore, container migration has caused an 18 °C decline in the average temperature. As the utilization of the CPU decreases, resources become accessible for subsequent tasks. The figure 5.8 illustrates the cumulative consumption of energy of the host prior to and subsequent to container migration.



(a) Load Imbalance Ratio without Migration (b) Load Imbalance Ratio with Migration

Figure 5.9: Load Imbalance Ratio with and without Container Migration

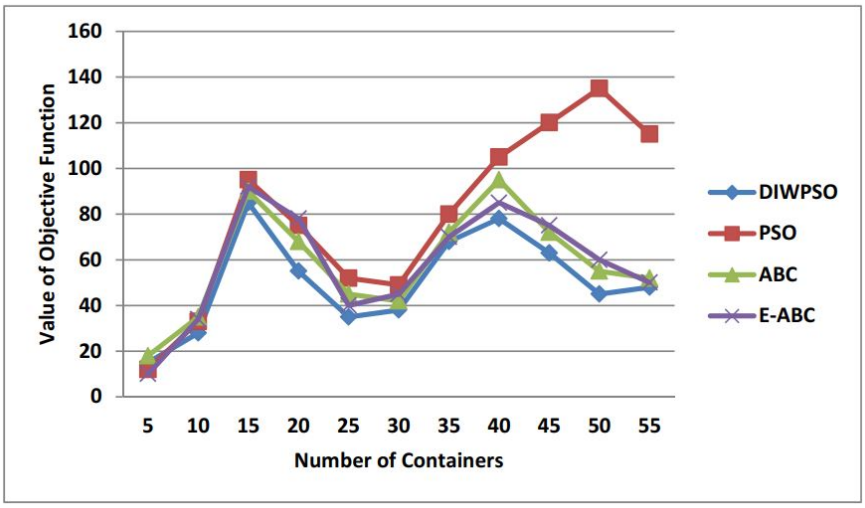


Figure 5.10: Value of objective function with varying number of containers

In order to validate the efficiency of the DIW-based PSO algorithm proposed for container migration, a comparative analysis is conducted with other heuristic algorithms, namely PSO[123], Artificial Bee Colony (ABC)[124] and Enhanced Bee Colony Approach (E-ABC)[125] to solve the optimization problems from Equations 5.8 to 5.13, respectively.

Initially, the load imbalance ratio is compared between the four algorithms without or with migration. Figure 5.9a illustrates the ratio of load imbalance among hosts in the absence of container migration. In Figure 5.9b, a substantial reduction in the load im-

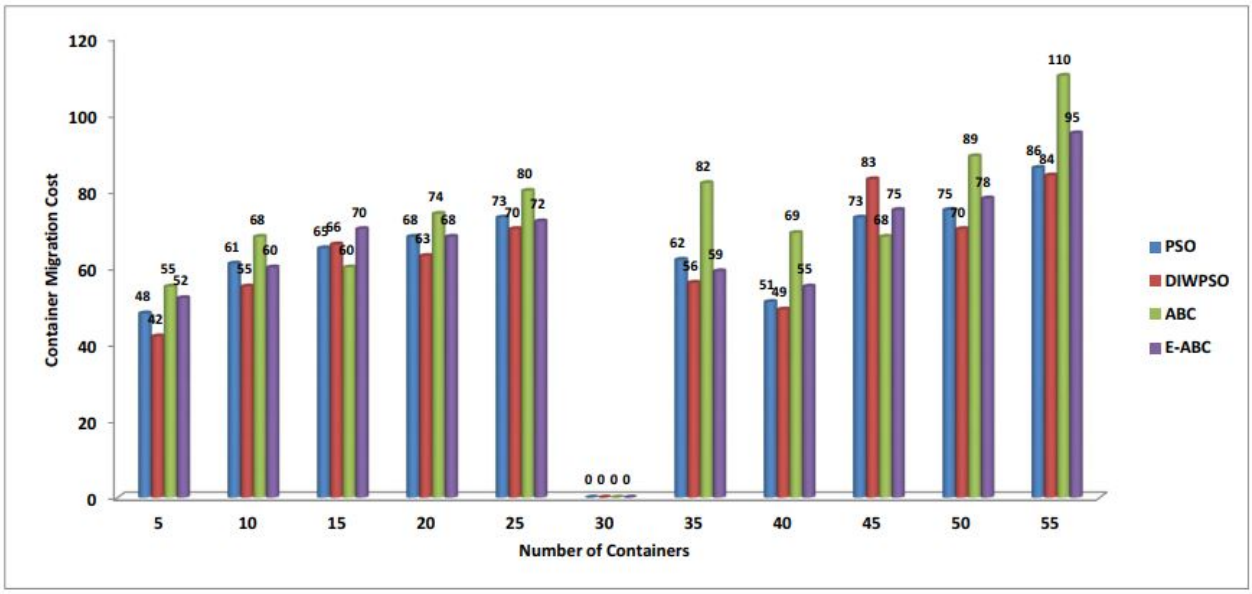


Figure 5.11: Cost of Migration with varying number of containers

balance ratio is observed for the four algorithms that implement container migration, as compared to Fig. 5.9a. As shown in Figure 5.9b, the load imbalance ratio of the proposed DIWPSO algorithm is substantially lower than that of the PSO, ABC and E-ABC after migration.

The container migration process involves the migration of containers from an overloaded host to another in order to reduce the load imbalance ratio and minimize migration costs. Figure 5.10 illustrates the optimal value of the objective function, which combines the load balancing degree and migration cost. As the number of containers increases, the objective function value of our proposed DIWPSO algorithm is significantly lower than that of PSO, ABC and E-ABC.

Additionally, the migration costs of the four algorithms are compared. The costs associated with container migration for containers of different quantities are illustrated in Figure 5.11. In the case of 30 containers, none of the algorithms involve container migration; therefore, the migration cost is negligible. Furthermore, the proposed DIWPSO algorithm demonstrates a reduced migration cost compared to alternative algorithms, except for scenarios involving 15 and 45 containers. The proposed algorithm for container migration is therefore more effective.

5.5 Summary

Container migration and containerization are emerging as key advancements in cloud and fog computing. They are essential for transferring containers from overloaded hosts to new ones with adequate resources to handle applications at network edges. Despite the growing use of containers, managing host energy consumption remains under-explored and challenging. This article explores techniques to optimize fog computing container management, aiming to reduce host load. Two frameworks were proposed: one for selecting containers when a host is overloaded and another for container migration. The container selection algorithm reduced migrations by 35% and energy consumption by 10.89%. Significant differences in power consumption, temperature, frequency, and CPU utilization were noted. The dynamic inertia weight-based PSO (DIWPSO) technique demonstrated faster and more efficient solutions for large-scale applications compared to existing methods such as PSO, ABC, and E-ABC.

Algorithm 5.3: Proposed Dynamic inertia weight based PSO Technique for Container Migration

Input: H, Ex-H and C
Output: MD

for *j = 1 to N* **do**
  Set L_{h_j} = L_{h_j} - $\sum_{i=1}^{m} y_{i,j} L_{c_i}$
end
for *i = 1 to m* **do**
  for *j = 1 to N* **do**
    if $y_{i,j}$ == *1* **then**
      put container c_i in to list M
      put host h_j into list Ex-H
    end
  end
end
Initialize of parameters : N_d, N_p, N_{iter}, W_{min} , W_{max}, c_1, c_2
Descending Sorting migrating containers M by CPU usage
Initialize X_{min} X_{max} V_{min} and V_{max} according to indexes of available hosts i.e. (H - (Ex-H))
gBestValue ← MaxValue
foreach $p \epsilon P$ **do**
  pBestValue ← MaxValue
  foreach $c_i \epsilon C$ **do**
    X_0 ← rand(X_{min}, X_{max})
    V_0 ← rand(V_{min}, V_{max})
  end
end
iter ← 0
while $iter < N_{iter}$ **do**
  foreach $p \epsilon P$ **do**
    foreach $c_i \epsilon M$ **do**
      if $X_{iter} > X_{max}$ **then**
        X_{iter} ← X_{max}
      end
      else if $X_{iter} < X_{min}$ **then**
        X_{iter} ← X_{min}
      end
    end
    Calculate Fitness function F_n for p based on equation 5.8
  end
  foreach $p \epsilon P$ **do**
    if $F_n < pBestValue$ **then**
      pBestValue ← F_n
      foreach $c_i \epsilon M$ **do**
        pBestPosition ← X_{iter}
      end
    end
    if $F_n < gBestValue$ **then**
      gBestValue ← F_n
      foreach $c_i \epsilon M$ **do**
        gBestPosition ← X_{iter}
      end
    end
  end
  foreach $p \epsilon P$ **do**
    Select the inertia weight (w) utilizing the following method.
    i = randInteger(1,4) `// 1 and 4 are inclusive`
    W = DIW [i]
    foreach $c_i \epsilon M$ **do**
      r_1 ← random(0,1)
      r_2 ← random(0,1)
      Calculate particle V_{iter+1} ← (W * V_{iter})* (r_1 * c_1 * (pBestPosition - X_{iter})) + (r_2 * c_2 * (gBestPosition - X_{iter}))
      X_{iter+1} = X_{iter} + V_{iter+1}
    end
  end
  iter = iter + 1
end
foreach $c_i \epsilon M$ **do**
  MD ← map(c_i, gBestPosition)
end
return MD

Chapter 6

Multivariate Time Series Ensemble Model for Load Prediction on Hosts using Anomaly Detection Techniques

The purpose of this study is to address the challenge of host load prediction in fog computing, which is essential for improving resource utilization and achieving service level agreements. Due to variations in load and the inefficiency of feature extraction, prediction of load on hosts presents a significant challenge. A predictive model capable of handling variable load patterns can better estimate future resource needs, which is crucial for capacity planning, meeting service-level goals, and enhancing energy efficiency. To improve the accuracy of workload prediction, this study proposes a time-series-based multivariate ensemble model utilizing anomaly detection techniques. The proposed framework involves deploying virtual machines on different platforms and extracting numerous parameters such as CPU utilization, number of cores, RAM, allocated memory, available memory, disk I/O, and network I/O. Given the potential inconsistencies in load prediction due to the large volume of data, various anomaly detection techniques are employed to reduce data redundancy. The performance of the proposed ensemble model is compared with various time series models using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and Accuracy. The effectiveness of the proposed ensemble model is demonstrated on the generated dataset, and its performance is measured based on these evaluation metrics. The ensemble model exhibits higher accuracy in workload prediction compared to current state-of-the-art models, achieving the lowest mean absolute percentage error (MAPE) and providing an accuracy of approximately 88%.

6.1 Overview

The growth of cloud-based online services has been increased by numerous organizations in response to the expansion of the Internet. The ability of cloud computing to offer on-demand network access empowers Enterprise Information System (EIS) to utilize service

components, including servers supplied by Amazon, Microsoft and Alibaba, without the need for the development of software or refactoring. As stated in their service-level agreement (SLA), these servers guarantee a high degree of availability with a 99.95% probability. However, maintaining such a high level of service while allocating minimal resources presents an immense challenge [126, 127].

In parallel with the tremendous growth in data volume, the rate at which data is generated is also rapidly increasing. According to a recent study of a healthcare-related IoT application, 30 million users generate up to 25,000 tuples of data per second [95]. As a result of the massive data volume, today's processing and storage capabilities cannot satisfy the needs. As cloud environment generate enormous amount of data volumes, so conventional computing methodologies, like distributed computing and cloud computing are unable to handle challenges such as reducing latency, response time and consumption of energy.

Fog computing, which links network edge devices and cloud centers efficiently, is presented as a more practical approach to address these problems [8]. Like the cloud, fog computing provides IoT users with data processing and storage services. Fog computing is focused on providing fog devices with on-site data processing and storage rather than transmitting it to the cloud. Both fog and cloud provide networking, processing, and storage resources.

The goal of fog computing on the IoT is to improve efficiency and performance while decreasing the amount of data sent to the cloud for processing, analysis and storage [96]. Figure 4.1 shows the hierarchical architecture of fog computing.

However, given the growing demand for a variety of IoT applications, fog nodes frequently become overloaded, degrading the response time of IoT apps with latency constraints and as a result, compromising users' quality of experience [97].

It is essential to predict the host load to achieve workload consolidation and allign with SLA. By employing an intelligent resource prediction methodology, the challenges of increasing resource utilization, decreasing operational expenses and maintaining the quality of service (QoS) can be effectively resolved. Accurate prediction of the host load can improve resource allocation and enhance application performance [38].

The deployment of virtual machines (VMs) into distributed and scalable resource allocation strategies is of the utmost importance for large-scale data centres, as it ensures load balancing and energy-performance compromises while reducing costs and power consumption [1]. Therefore, prediction of future VM workload can not only improve resource utilization but also address the issue of energy consumption.

In our proposed work, we generate a dataset using various VMs designed to handle numerous applications. The emergence of containers in the cloud era represents a paradigm shift, thanks to their lightweight nature, simplified configuration and management, and the potential to significantly reduce startup times [103]. Podman is a container engine for running and managing containers. Podman has been used to execute containerized applications on virtual machines in our proposed work. Table 6.1 illustrates the two platforms utilized in the proposed work. Due to the high resolution, heavy applications, such as image-related applications, can be efficiently operated through containers. The container-based virtual machines are shown in Figure 4.2.

Table 6.1: Platforms and Software Used for DataSet Creation

Platform 1	**Platform 2**	**Software**
Processor-11th Gen Intel(R) Core(TM)i7-1165G7 2.80 GHz	INTEL(R) XEON(R) CPU E5-2650 V2 @ 2.60GHZ	Oracle VM VirtualBox, Python(OpenCV)
RAM 12GB	RAM 16GB	

The majority of current approaches utilized to forecast host resource utilization are univariate time series prediction models. These models solely rely on historical data that is relevant to the target variable in order to generate predictions, with CPU usage being the primary resource consumption metric. Due to the correlation between various variables, multivariate time series models offer a higher degree of predictive accuracy in comparison to univariate models [128, 129].

These inter-dependencies emphasize the significance of forecasting multivariate time series and facilitate the extraction of secondary features, thereby enhancing the accuracy of predictions. To better understand host workload, the dataset is generated by extracting multiple random parameters such as CPU usage, number of cores, RAM, allocated memory, available memory, disk I/O, and network I/O etc. acquired from numerous VMs in a real-time environment.

Various time series models are introduced to forecast host workload by leveraging historical workload traces. Our proposed framework involves creating an ensemble model that can predict host load, utilizing six time series models and selecting the top three models based on their performance or topsis score. The ensemble model that is being proposed will acquire information of the inbuilt characteristics of virtual machine (VM) workload traces in order to enhance its ability to forecast the workload of forthcoming VMs. Prior to implementing proposed ensemble model into the decision-making process, the predicted load data will be transmitted to a workload analyzer as shown in Figure 6.1.

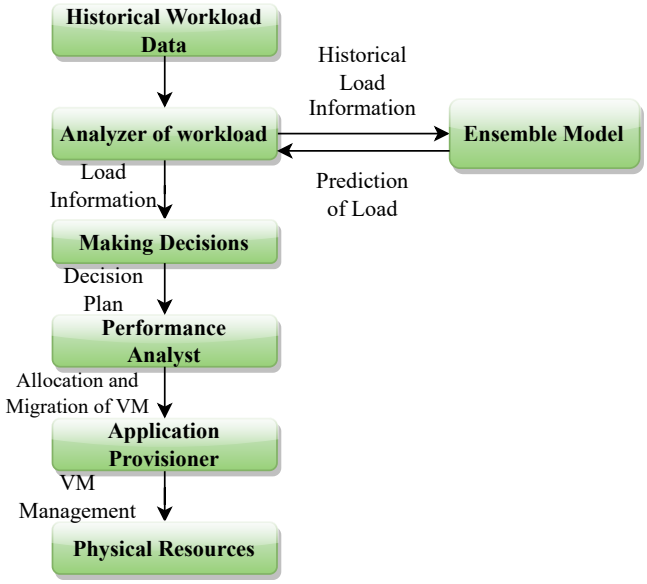


Figure 6.1: Framework for Workload Prediction

It will produce a decision plan for virtual machine (VM) administration and forward it to the performance analyst prior to delegating the selection of the allocation and migration plan to the application provisioner. The application provisioner will proceed to allocate the accepted user request to physical servers in accordance with the designated VM placement strategy. When overloading occurs, the most optimal migration strategies will be selected. As depicted in Figure 4.3, workload forecasting can assist decision-makers in capacity planning and selecting appropriate VM placement and migration techniques.

6.2 Problem Definition

Let Z = $\{Z_1, Z_2, Z_3, ...Z_n\}$ represents a multivariate time series, consisting of individual series that are correlated. In the generated dataset, Z consists of $Z_1, Z_2, ...Z_{10}$, where Z_1 is the goal variable representing CPU utilization and the remaining are correlated time series, as illustrated in Table 6.2. The objective is to forecast future observations of the target time series in Z, using a multivariate time series dataset. Usually, analysts select the CPU utilization time series Z_1 as the target prediction time series. We suppose that the target time series Z contains historical data from a window $[t_{curr-l}, t_{curr}]$, with "l" timestamps in the past, where t_{curr} represents the current time step. The objective is to forecast Z_1 observations within the future window $[t_{curr+1}, t_{curr+\Delta}]$, denoted as the Δ-steps ahead prediction problem, where Δ represents the intended horizon preceding the current timestamp.

Table 6.2: The variables in the multivariate time series of workload

Time Series (Z_n)	Parameters
Z_1	Total Usage of CPU
Z_2	user
Z_3	nice
Z_4	idle
Z_5	Network Transmitted Throughput
Z_6	Network Received Throughput
Z_7	cpu cores
Z_8	MemFree
Z_9	MemAvailable
Z_{10}	Disk Write Throughput

6.3 Proposed Work

This section describes the proposed work. It provides an overview of Performance Modelling for usage-aware Forecasting, dataset generation using modeling and simulation, workload prediction techniques and various anomaly detection techniques used.

6.3.1 Performance Modelling for Usage-Aware Forecasting

Let X(t) represents the collection of all prior CPU utilization traces at 5-minute intervals. Y(t) denotes the CPU utilization at the subsequent time (t+1). In order to forecast CPU utilization at time (t+1), historical data related to CPU utilization at time intervals (t-1) and (t-2) along with current data at time t will be used. By providing the predicted output value (t + 1) and the input time values (t - 2), (t - 1) and t, this issue can be represented as a regression problem. The Mean Absolute Error (MAE) is a robust metric that is used to assess the precision of regression models. The mean absolute error(MAE) can be calculated using the Equation 6.1.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\frac{A_t - F_t}{A_t} \tag{6.1}$$

where n represents the prediction intervals, A_t represents the actual CPU utilisation value, and F_t represents the forecasted CPU utilization value. The performance analyst will use this forecasted CPU utilization for VM allocation or migration at the destination

Table 6.3: Notations of the variables used in various Equations

Symbol	**Definition**
VM	Virtual Machine
PM	Physical Machine
THU_{cpu}(PM)	Upper threshold value of PM
THL_{cpu}(PM)	Lower threshold value of PM
FU_{cpu}(VM)	VM's predicted CPU utilization
CU_{cpu}(PM)	PM's current CPU utilization
TH_{cpu}	Threshold value of CPU utilization
TU_{cpu}(PM)	PM's total CPU utilization

host, subject to the constraints shown in Equation 6.2 [130].

$$FU_{cpu}(VM) + CU_{cpu}(PM) <= TH_{cpu} * TU_{cpu}(PM) \tag{6.2}$$

Notations of the variables used in various Equations is shown in Table 6.3.
Equation 6.3 indicates that the destination PM's status will be updated when the VM is deployed to it.

$$TU_{cpu}(PM) = FU_{cpu}(VM) + CU_{cpu}(PM) \tag{6.3}$$

The new PM's must be searched for (or the idle PMs must be switched into the active state) for hot spot migration or VM migration. The violation can be formulated by Equation 6.4.

$$FU_{cpu}(VM) + CU_{cpu}(PM) >= TH_{cpu} * TU_{cpu}(PM) \tag{6.4}$$

The case of a balanced host, i.e., total CPU load, lies between specified upper and lower thresholds, as shown in Equation 6.5.

$$THL_{cpu}(PM) <= TU_{cpu}(PM) <= THUcpu(PM) \tag{6.5}$$

The case of host under-load, i.e., CPU utilization is less than the specified threshold, is shown in Equation 6.6.

$$TU_{cpu}(PM) <= THL_{cpu}(PM) \tag{6.6}$$

This is also referred to as a "cold spot"; VM migration is necessary to disable the PM or transition to a sleeping state so that the rest PM can be utilized, if it is fulfilled. It will increase energy efficiency while decreasing the quantity of active PMs. The case of host overload, i.e., CPU utilization is more than the specified threshold, is shown in Equation

6.7.

$$TU_{cpu}(PM) >= THU_{cpu}(PM) \tag{6.7}$$

This is also known as a hot spot case; if it is satisfied, VM migration is required to avoid an unnecessary SLA violation, and other PMs will be sought to meet the increased demand for a specific VM.

6.3.2 DataSet Generation using Modeling and Simulation

To acquire a deeper understanding of host load, various virtual machines are deployed to generate the dataset. The dataset contains 10,000 records with ten performance parameter values. VM Creation takes into account the following parameters: RAM, CPU cores, disk read/write throughput, disk size and memory. As illustrated in Figure 6.2, a variety of features of operational hosts are extracted, including CPU utilization, number of cores, RAM, allocated memory, available memory, disk I/O and network I/O.

Table 4.3 describes performance parameters and their range used for the construction of VMs. Table 6.1 shows the platform and software used to generate the dataset. The proposed model is designed for a Container as a Service (CaaS) environment, where applications are executed on containers. These containers run various applications and are executed on virtual machines. CPU utilization with a timestamp is used as the target variable for prediction. The dataset consists of CPU utilization data from various virtual machines, which is gathered in a time period of five minutes and stored in files as shown in Table 6.4. The basic statistics for the features utilized in this work are presented in Table 4.1.

Table 6.4: Sample of DataSet

Date	user[1](%)	nice[2](%)	idle[3](%)	Total Usage of CPU(%)	Network transmitted Throughput (MBITS/s)	Network recieved Throughput (MBITS/s)	cpu cores[4]	MemFree[5] (KB)	MemAvailable[6] (KB)	Disk Write Throughput(KB\s)
15-06-2022	15.76	0.1	48.97	51.03	20.6	18.8	9	144280	386204	0.060524225
16-06-2022	15.88	0.1	48.85	51.15	18.6	16.3	9	77232	217432	0.090074539
17-06-2022	16.05	0.1	48.59	51.41	17.0	15.3	9	54756	37168	1.06206131
18-06-2022	16.11	0.1	48.02	51.98	20.4	16.9	9	61428	47468	0.985242367
19-06-2022	15.96	0.1	47.27	52.73	12.7	10.3	9	54820	21752	0.130439281
20-06-2022	15.82	0.1	46.58	53.42	11.3	11.2	9	55792	41660	0.442959309

1) The percentage of CPU consumption that happened while the user was executing (application)
2) The percentage of CPU utilization while executing at the user level with nice priority.
3) The percentage of time the CPU or CPUs were idle when the system had no pending disk I/O requests.
4) Number of virtual CPU cores provisioned.
5) The memory that is actively not used in terms of KB.
6) An estimate of how much memory is available for running new apps without swapping in KB.

6.3.3 Workload Prediction Techniques Used

It is assumed that the problem of workload prediction is a time series-based regression problem that can be effectively addressed through the implementation of deep learning

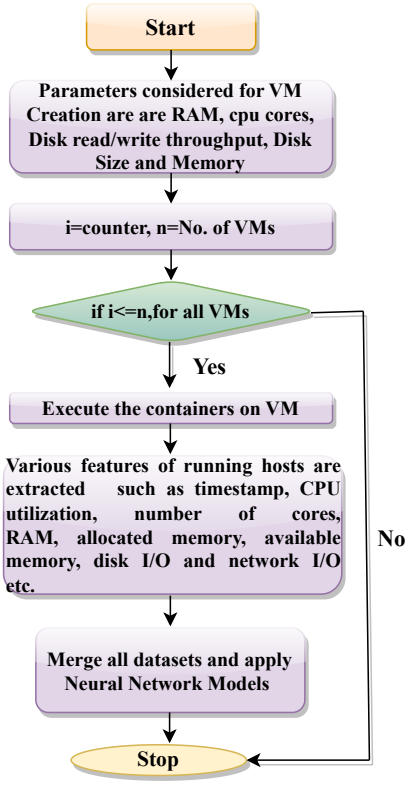


Figure 6.2: Flow Chart of Data Generation

methodologies. Numerous workload' implicit variation patterns and inherent characteristics can be uncovered by the deep learning model. As a result, the deep learning model exhibits greater efficacy in discerning features from the workload data containing intricate relationships compared to the shallow model. The workload traces of virtual machines captured at various time intervals act as input to these models and the result is the predicted future load of VMs. These methodologies utilize the unsupervised learning strategy, wherein a restricted understanding of the available resources is supplied. Here, for the experiment, we use ARIMA, Prophet, XgBoost, RNN LSTM, ConvLSTM and DLM as the baseline models to compare with the proposed model. Various workload prediction models used in proposed work are discussed as follows:

1. **ARIMA Model-** The ARIMA model (Auto-Regressive Integrated Moving Average) creates a linear equation that describes and forecasts time series data. Three terms: p, d and q define an ARIMA model, where "p" denotes the order of the "Auto Regressive" (AR) term. It refers to the quantity of Y delays to be applied as forecasters. Moreover, "q" denotes the Moving Average (MA) term's order. A model where Y_t depends only on its lags is a pure auto-regressive (AR alone) model.

In other words, Y_t is a function of the 'lags of Y_t,' shown in Equation 6.8.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \varepsilon_1 \tag{6.8}$$

where $Y_t - 1$ is the series' lag1, β_1 is the lag1 coefficient that the model predicts, and α is the intercept term that the model also calculates.

2. **Prophet-** An open-source library called prophet is explicitly made for forecasting time series datasets. It is easy to use and develop to automatically find optimal hyper parameters for the model to forecast data with trends and seasonal structure. The three primary model elements in prophet are trends, seasonality, and holidays. These components are combined with Equation 6.9.

$$y(t) = g(t) + s(t) + h(t) + e(t) \tag{6.9}$$

Here,

- g(t) g(t) is a trend-modeling function that can be linear or logistic;
- s(t) is a Fourier-based daily, weekly, or annual seasonality function;
- h(t) is a holiday function that accounts for irregular holidays;
- e(t) is an error change that the model does not fit.

3. **XgBoost-** XgBoost is a fast gradient-boosting implementation for classification and regression problems. It can also anticipate time series, however the dataset must be turned into a supervised learning model. Numbers are used to reorganise the time series dataset for supervised learning. The past time steps might be input variables and the next time step the output variable to forecast load on host.
The Classification and Regression Trees (CARTs) that makeup XGBoost are numerous. The object of CART is Gini coefficient which is shown in Equation 6.10.

$$Gini(D) == 1 - \sum_{z=1}^{Z} P_{\hat{z}}^2 \tag{6.10}$$

where **P** is probability with limits and **Z** is the scale of labels. [131].

4. **RNN LSTM-** LSTM is used for load forecasting in the proposed work. The LSTM has feedback connections, unlike conventional feed-forward neural networks. The architecture of RNN LSTM is shown in Figure 6.3.

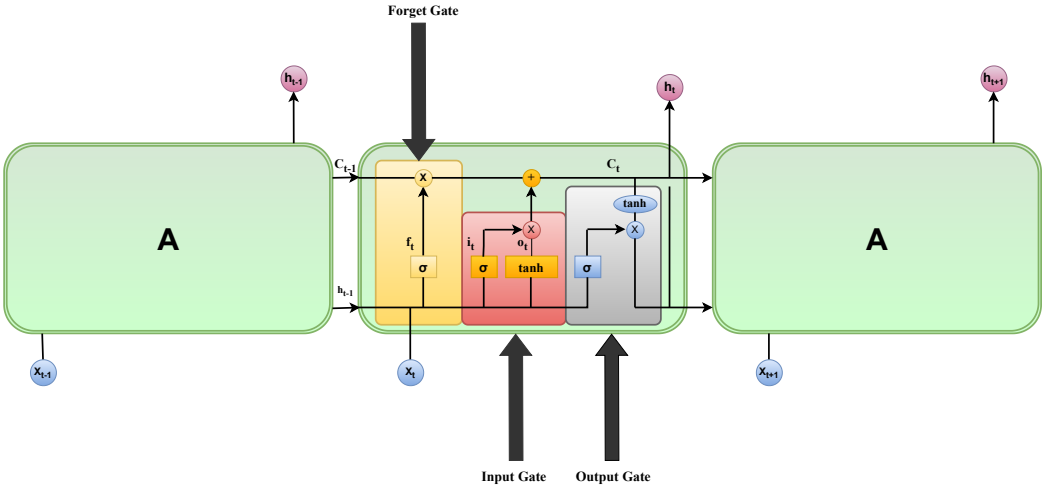


Figure 6.3: Basic Architecture of RNN LSTM

For time step t, the input gate i_t, forget gate f_t and output gate o_t are calculated iteratively using the following equations, shown in Equation 6.11.

$$\begin{aligned} f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f) \\ i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \\ \tilde{c}_t &= tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}) \\ c_t &= f_t.c_{t-1} + i_t.\tilde{c}_t \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \\ h_t &= o_t.tanh(c_t) \end{aligned} \tag{6.11}$$

where x_t is the second variable type, which is only temporally changing but spatially static. In this study, x_t represents the input historical CPU utilization values; c_t represents the activation vectors for each cell; h_t represents the corresponding predicted value of CPU utilization. W_{ix} denotes the weight matrix between the input CPU variable and the gate output. b_i is the bias value of the input gate. Similarly, W_{ih}, W_{fx}, W_{fh}, $W_{\tilde{c}x}$, $W_{\tilde{c}h}$, W_{ox} and W_{oh} represent the weight matrices connecting the vectors of the first and second subscripts. The bias values $b_{\tilde{c}}$, b_f,b_i, and b_o indicate the relevant bias values. σ and tan h represent the activation functions used in LSTM neural network.

5. **ConvLSTM-** ConvLSTM is a spatiotemporal recurrent neural network that includes convolutional structures in input-to-state and state-to-state transitions. Load forecasting with ConvLSTM uses 1-D convolution, unlike image processing. Combining a sequence with a window size of kernel_size with the original sequence

yields a new one. A pooling function in the convolution network extracts relevant data features. The inner structure of ConvLSTM is shown in Figure 6.4.
One distinctive characteristic of ConvLSTM architecture is that all of the ConvLSTM's inputs χ_1,..., χ_t which are CPU utilization values, cell outputs C_1,..., C_t, hidden states H_1,..., H_t and gates i_t, f_t, o_t are 3D tensors with the last two dimensions representing spatial dimensions. These are represented as vectors on a spatial grid to better understand the inputs and states. H_t represents the corresponding predicted value of CPU utilization. The ConvLSTM predicts a grid cell's future CPU utilization based on its neighbors' inputs and historical states. This is accomplished by employing a convolution operator in state-to-state and input-to-state transition Equation 6.12 shows the main equations of ConvLSTM, where '*' denotes the convolution operator and 'o', as previously, signifies the Hadamard product:

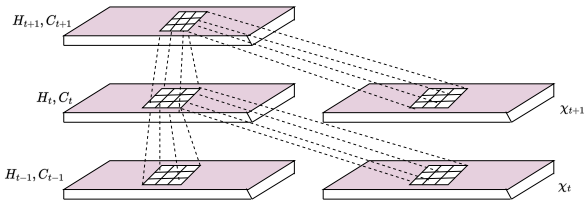


Figure 6.4: Inner Structure of ConvLSTM

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * \chi_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * \chi_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \\
C_t &= f_t \circ C_{t-1} + i_t \circ tanh(W_{xc} * \chi_t + W_{hc} * H_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * \chi_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o) \\
H_t &= o_t \circ tanh(C_t)
\end{aligned}
\tag{6.12}
$$

6. **The Dynamic Linear Model (DLM)-** The Dynamic Linear Model is a time series model commonly used for econometrics and forecasting. The DLM represents a time series as a sequence of latent states and observable measurements. The observed measurements are modeled as a linear combination of the latent states, whereas the evolution of the latent states across time is modeled using a set of linear equations as follows:

$$x_t = F_t * x_{t-1} + G_t * w_t \tag{6.13}$$

- At time t, x_t is the vector of latent states.
- F_t is the state transition matrix between t and t-1.
- x_{t-1} represents the vector of latent states at time t-1.

- G_t The state noise coefficient matrix indicates uncertainty or randomness
- w_t is the vector of random state noises at time t, assuming a multivariate normal distribution.

6.3.4 Anomaly Detection Techniques Used

The detection of anomalous and unexpected data points in a system is known as fault detection. It is a fundamental approach for detecting fraud, suspicious activity, network infiltration and other malicious activities. As a result, the following detection methodologies for anomalies are used in proposed work.

1. **Isolation Forest-** It is an unsupervised machine-learning technique used to find abnormalities in data. On an Isolation Tree Forest (I-TREES), it has been tested. It draws random attributes from the provided dataset based on the Decision Tree (DT). The split value will be chosen randomly from the selected values, which are the maximum and minimum values. Anomaly scores in I-trees are calculated according to Equation 6.14.

$$S(x, m) = 2^{\frac{-E(h(x))}{c(m)}} \tag{6.14}$$

   where x = data point, m = data sample, h(x) = average search height for x from I-trees, and c(m) = average value of h(x). If the average value is 0, then $2^0 = 1$ denotes an anomalous point, and if it is 1, then $2^{-1} = 0.5$ indicates a regular point.

2. **K-Nearest Neighbors (KNN)-** It is a supervised machine-learning technique. This method can be used to spot odd occurrences. This method discusses a generative model for finding anomalies in dataset. A generative model can produce new data occurrences. The length of a data point's from K_{th} Nearest Neighbour could be regarded as the data point's outlying score. The anomalous score for new points is calculated using equation 6.15 :

$$E(D) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{6.15}$$

3. **Open Support Vector Machine (OSVM)-** It is a machine learning model that can analyze data and identify designs. The SVM technique is used to handle classification and regression tasks. This model forecasts by identifying problems on one side of the gap and comparing them to points on the other side. Based on a few samples, it is only possible to forecast some new aberrant data patterns. As a result, in one-class SVM, just one class of data, known as the "Normal class," is

trained that helps to identify anomalies [132].

4. **Histogram-** It is a visual representation of numerical data distribution. In this technique, the data is binned and counted for each bin. A histogram is a bar chart aggregated using numerous aggregation methods such as sum, average, and count. Histogram-Based Outlier Score (HBOS) [133] is an unsupervised statistical anomaly identification technique.

5. **Local Outlier Factor (LOF)-** Local outliers are a non-supervised method of detecting anomalies. For each data point, it computes the local density deviation. The data points are measured in relation to their neighbors. An outlier is a sample with a lower density than its neighbors. The LOF method analyses data point density and finds abnormal or deviating values by calculating their local deviation and comparing them to their neighbors.

6.4 System Model

A summary of our system model is illustrated in Figure 6.5. We opted for a data-driven, machine-learning strategy in the proposed system that predicts the future workload level by learning from past application workload. For the purpose of forecasting the CPU utilization of multiple devices operating across fog nodes in an IoT environment, an ensemble model is suggested.

In a Container as a Service (CaaS) environment, where applications are executed on containers, the proposed model is intended to function. In contrast to univariate models, multivariate time series models exhibit enhanced predictive accuracy due to the degree of interdependence and correlation that exist among numerous variables. In order to generate the dataset, a number of random parameters are utilized, including disk I/O, network I/O, CPU utilization, number of cores, RAM, allocated memory and available memory, all of which are obtained from different virtual machines in a real-time environment. Machine learning algorithms acquire knowledge from past data and support data-driven decision-making processes. The utilization of prediction models, as elaborated in Section 6.3.3, enables real-time workload prediction on the host.

In order to compare various models, additional performance metrics, including MAE, MSE, RMSE, MAPE, and accuracy, are assessed. Utilizing means and standard deviations, Topsis compares and evaluates prediction methods. Our research centres on the prediction of workload, for which we examine a range of machine learning techniques. In order to construct an ensemble model for load prediction, the top three models with the

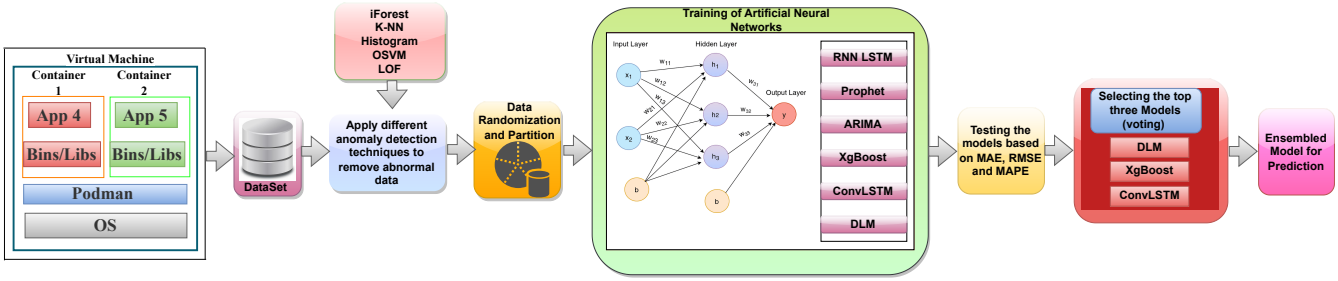


Figure 6.5: System Model

lowest MAPE value and highest accuracy are chosen based on their topsis scores. The mathematical modelling of the system model is shown in Section 6.4.1.

6.4.1 Mathematical Modelling of the System Model

1. **DataSet Generation:** Let set Z consists of $\{Z_1, Z_2, ...Z_{10}\}$ representing multivariate time series data, where "Z_1" is the goal variable representing CPU utilization and the remaining are correlated time series parameters, as illustrated in Table 6.2.

2. **Load prediction:** The objective is to forecast future observations of the target time series in set Z, using historical and current data from a window $[t_{curr-l}, t_{curr}]$, with "l" timestamps in the past, where t_{curr} represents the current time step.

3. **Dataset Preprocessing:** $\forall Y_k, \therefore k \in s$, *where* ***'s'*** *represent models used for load prediction as follows* ($s = Y_{ARIMA}, Y_{XgBoost} \ldots . Y_{ConvLSTM}$), $Y_k.AnomalyDetect()$ *as described in section 6.3.4* .
   $\{$*Anomaly Detection Techniques-iforest, KNN, Histogram, OSVM and LOF*$\}$

4. **Training of Models:** $\forall Y_k, \therefore k \in s$, *where* ($s = Y_{ARIMA}, Y_{XgBoost} \ldots . Y_{ConvLSTM}$), $Y_k\ [X \rightarrow Y] : Y_k[x_i \rightarrow y_i]$, where X→Input and Y→Output.

5. **Models Comparison:** Models are compared based on Mean Absolute Error (MAE) using Equation 6.18 , Mean Square Error (MSE) using Equation 6.19, Root Mean Square Error (RMSE) using Equation 6.20 , Mean Absolute Percentage Error (MAPE) using Equation 6.21 and Accuracy using Equation 6.22

6. The top three models are chosen based on the above performance metrics and topsis score. These three best models are DLM, XgBoost and ConvLSTM model. The tuning parameters of these selected models are as follows:
   Xgboost parameters: *no of estimators=100, max depth=5, learning rate=0.1, epochs=50,60,70*
   DLM Parameters: *freq='D", epochs=50,60,70, learning rate=1e-3*

ConVLSTM parameters: *activation=relu, optimizer=adam, loss='mse', epochs=50,60,70*

7. Let $f_{XGBoost}$, f_{DLM}, and $f_{ConvLSTM}$ denote the prediction functions of the XGBoost, DeepAR (DLM), and ConvLSTM models, respectively. Given a time series input x, the ensemble prediction $\hat{y}_{ensemble}$ can be expressed as:

$$\begin{aligned} \hat{y}_{ensemble}(x) = \alpha \cdot f_{XGBoost}(x) + \\ \beta \cdot f_{DLM}(x) + \gamma \cdot f_{ConvLSTM}(x) \end{aligned} \tag{6.16}$$

   where α, β, and γ are the weights assigned to each individual model's predictions.

8. **Final output of Proposed System Model**

$$\hat{y}_{ensemble}(x) = FutureLoadPrediction \tag{6.17}$$

6.5 Methodology

The following section shows the methodology of the proposed approach. Figure 6.6 presents the flowchart of the proposed approach. In the proposed framework, an ensemble model is proposed for the prediction of the CPU Utilization of numerous machines running over fog nodes in an IoT environment. The proposed model is designed for a Container as a Service (CaaS) environment, where applications are executed on containers. Multivariate time series models provide superior predictive accuracy when compared to univariate models, owing to the interdependence and correlation among multiple variables. So,the dataset is generated using multiple random parameters such as CPU usage, number of cores, RAM, allocated memory, available memory, disk I/O and network I/O obtained from various VMs in a real-time environment. A sample dataset has been shown in Table 6.4.

Among these parameters, CPU utilization with a timestamp is used as the target variable for prediction. Anomaly detection techniques as discussed in previous section are applied to remove abnormal data and improve the efficacy of prediction models.

Further, in partitioning phase, data is split into two parts: training (70%) and testing (30%). Prediction models discussed in Section 6.3.3 are used for workload prediction on host in real time. Moreover, several performance metrics, such as MAE, MSE, RMSE, MAPE and accuracy are evaluated to compare various models. Topsis ranks and compares prediction methods by analysing means and standard deviations. Techniques for machine learning can be optimized with topsis score which is discussed in section 6.6.4. Finally,

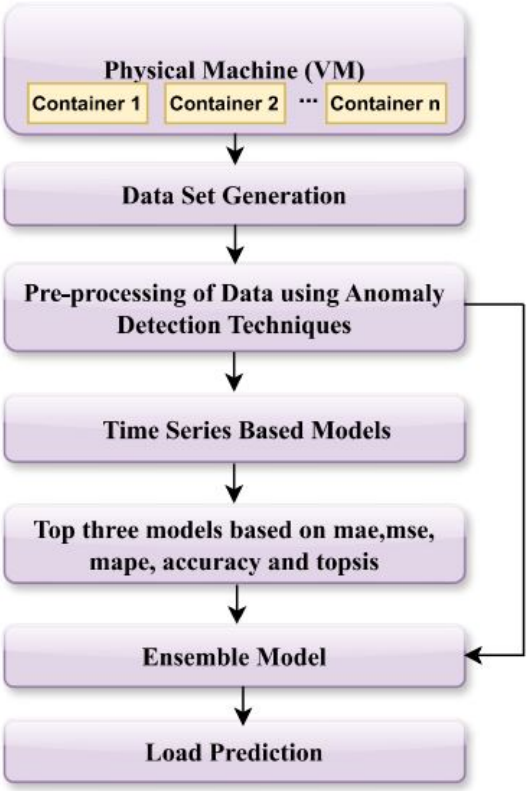


Figure 6.6: Flowchart of the proposed Framework

the top three models are selected on the basis of topsis score to build an ensemble model for load prediction.

Algorithm 1 explains the data processing and prediction phases of the proposed work and methodology. The conventions used in the algorithm are shown in Table 6.5. The load prediction algorithm discusses workload prediction in four phases, as follows:

- **Phase I:** The preprocessing of the generated dataset takes place. Various anomaly detection techniques, such as Isolation Forest (iForest), K-Nearest Neighbors (KNN), Open Support Vector Machine (OSVM), Histogram and Local Outlier Factor (LOF), are employed to identify abnormal data. The utilization of these techniques aids in minimizing redundancy and scaling up storage space that enables the storage of relevant time-series data.
- **Phase II:** The dataset is split into two parts, i.e., the training and testing datasets, using splitting criteria of 70–30%, where 70 percent of the data is used to train the models and 30 percent is used for testing purposes.
- **Phase III:** To forecast workload, RNN LSTM, ConvLSTM, ARIMA, XgBoost, DLM, and Prophet models are utilized. Then, several performance indicators, such

as MAE, RMSE, MSE, MAPE and accuracy are calculated to compare different models as shown in Figure 6.5.

- **Phase IV:** The top three models are chosen based on the topsis score and which are further utilized to develop an ensemble model for load prediction.

6.6 Experiment Analysis

This section includes the experimental description, which covers the experimental setup, performance evaluation, evaluation criteria for various prediction models and topsis score evaluation.

6.6.1 Experimental setup

The experiments are executed on a machine with a 2 TB hard drive, 12 GB of memory, and is conducted on an 11th Gen Intel(R) Core(TM) i7-1165G7 using Python and Keras with Tensorflow etc using Google Colab, for model training. The dataset comprises 10,000 records with ten performance parameter values which are shown in Table 6.4. Various tuning parameters used for load prediction models are shown in Table 6.6.

6.6.2 Performance Evaluation

In this section, the performance of various models is evaluated. The dataset used for evaluation includes the CPU utilization of virtual machines. This information is collected at the timestamp of five minutes and saved in csv files. A single VM collects 288 observations per day. The PlanetLab dataset is also used for testing and comparison. The PlanetLab is an open platform for deploying and accessing planetary-scale services. The CPU utilization data of all VMs in a history window is used as the input vector for the workload prediction model.

6.6.3 Evaluation Criteria for Prediction Models

In proposed work, the evaluation of prediction results is performed using performance metrics as: mean absolute error (MAE), mean absolute percentage error (MAPE), mean square error (MSE) and root mean square error (RMSE) and accuracy.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6.18}$$

Algorithm 6.4: Workload Prediction Algorithm

Input: μ = DataSet

Output: Load Prediction

1 **while** μ*!=0* **do**

1: **Phase I:Dataset Pre-processing**

2: *for each* M_k,$k \in s$. *(s=* M_1, M_2 .... M_n*)*

3: M_k*.AnomalyDetect()*

4: **Phase II:Data Partition**

5: μ_1=*random(*μ, *fraction =0.70)*

6: $\mu_2 = \mu - \mu_1$

7: μ_1= *Training Set*

8: μ2= *Testing Set*

9: **Phase III:Training Models**

10: M_k, *for* $k \in s$. *(s=* M_1, M_2 .... M_n*)*

11: M_k *[X*$\rightarrow$*Y] :* $M_k[x_i \rightarrow y_i]$, *where* $(x_j, y_j) \in \mu_{train}$

****Testing Models in Training****

12: $S=M_k(x_i)=y_i$' *where* $x_i \in \mu_{test}$

*************Error Rate************

13: *absolute(*y_i- y_i')=e

14: $MAE=\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

15: $MSE=\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

16: $RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

17: $MAPE=\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$

18: $Accuracy=(100 - \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|)$

19: **Result:**EM(x)

20: **Phase IV: N-semble and**

21: Testing Phase

22: M_k *for* $k \in$ *top(*M_m*Acc%,3)*

23: $M_{EM}[y_i' \rightarrow y_i]$, *where* $i \in \mu_{test}$

\$ *top(*M_m*Acc%,3)*

24: $EM(x_i) = M_{EM}(M_k(x_i))$,*where* $x_i \in \mu_2$

25: *Since,* $(M_k(x_i) = y_i'$, *therefore,* $EM(x_i)=M_{EM}(y_i')$

2 **end**

Table 6.5: Conventions Used

Conventions	**Description**
i	Counter of feature vectors
X, Y	Input space and output space
x, y	Input and target vectors
µ	DataSet
µ1, µ2	Data partitions: Training and testing datasets i.e. µ1 and µ2.
M	Representation of Model i.e., M: X->Y
MEM	Represents the final ensemble model created
EM(x)	Final output of ensemble model
e	Acceptable error

Table 6.6: Various Tuning Parameters of Prediction Models

Model	**Package**	**Tuning Parameters and Their Values**
RNN LSTM	Keras	MinMax Scaler, Activation Function = Relu, Optimizer = Adam, epochs=50,60 and 70, Loss Function = 'MSE', No. of Layers = 3, No. of Neurons = 20
Prophet	fbProphet	None
ARIMA	Pmdarima	ADF = -2.166543, P-value = 0.2186, No. of lags = 6, Number of observations used for ADF Regression and critical value calculator = 164
XgBoost	XgBoost	None
CONV LSTM	Keras	MinMax Scaler, Activation Function = Relu, Optimizer = Adam, Epochs = 50,60 and 70, Loss Function = 'MSE', No. of Layers = 3, No. of Neurons = 20
DLM	pydlm	F_t = [[1, 1],[0, 1]], G_t = [[1], [0.5]], H_t = [[0.5]], lag length = 2

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.19}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.20}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{6.21}$$

$$Accuracy = (100 - \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|) \tag{6.22}$$

where n is the number of load predictions on host, y_i is the actual load value and $\hat{y}_i$ is the predicted load value.

6.6.4 Topsis

Hwang and Yoon developed the technique for Order Preference by Similarity to Ideal Solution (TOPSIS) decision analysis method. TOPSIS, one of many Multi-Criteria Decision Making (MCDM) and Multi-Criteria Decision Aid (MCDA) methods developed to solve real-world decision problems, works satisfactorily in various application areas [134]. It is based on the idea that the solution chosen should be as close to the positive best solution as much as possible while remaining as far away from the negative best solution. It compares alternatives by assigning weights to various criteria, normalizing their scores, and then calculating the total score and rank. Table 6.7 displays the topsis analysis results for the workload prediction models with and without anomalies. It provides details of the results obtained from different models. The performance comparison of various models based on topsis scores using a graph is shown in Figure 6.7.

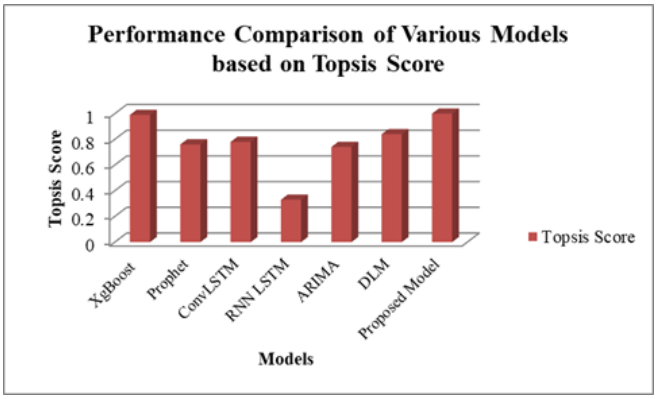


Figure 6.7: Performance comparison of various prediction models using topsis score

6.7 Result Analysis, Comparison and Discussion

The input of this work is the CPU utilization of virtual machines (VMs) and the output is the CPU utilization in the future. Two subsets comprise the data set: the training subset and the testing subset. The prediction of load for a single virtual machine (VM) requires a total of 288*30 time intervals, out of which 30% is allocated for testing and 70% is used for training. As experimental baseline models, we employ ARIMA, Prophet, XgBoost, RNN LSTM, ConvLSTM, and DLM in comparison to the proposed ensemble model. The different prediction models undergo testing, using testing data and training data containing varied hyper-parameter values.

Table 6.7: Comparative Performance Study of Various Models with Different Anomaly Detection Techniques using topsis analysis

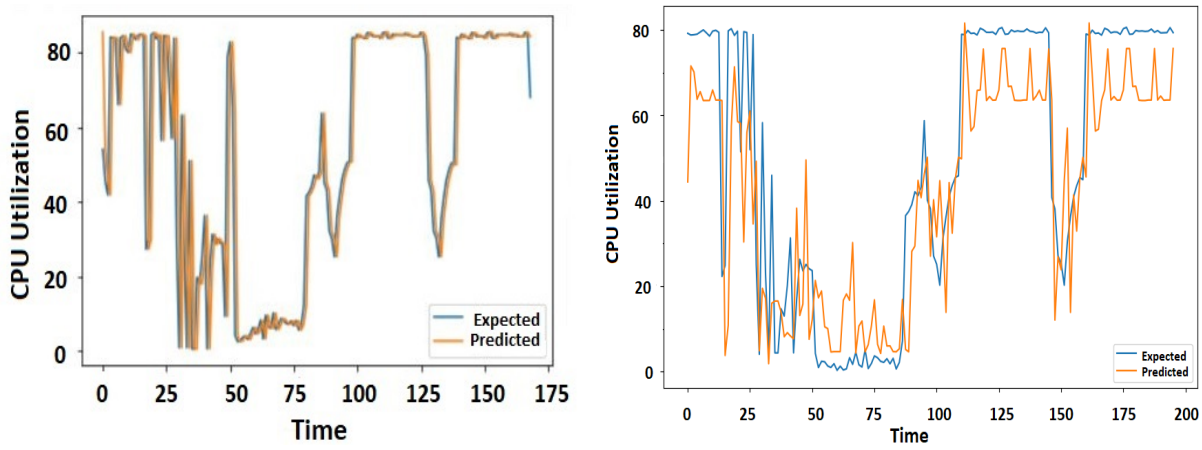
Models	MAE	MSE	RMSE	MAPE	Accuracy	Topsis Score	Rank
ARIMA							
ARIMA	10.1	280.96	16.76	0.37	0.63	0.74	20
AR + iforest	8.46	304.47	17.45	0.39	0.61	0.73	22
AR+ KNN	16.49	317.94	17.83	0.42	0.58	0.68	29
AR + LOF	13.42	280.23	16.74	0.30	0.60	0.72	24
AR + Histogram	10.1	280.97	16.76	0.38	0.62	0.74	21
AR + OSVM	33.28	1249.83	35.35	0.60	0.40	0.26	37
Prophet (Pro)							
Prophet	27.96	29.7	5.45	0.34	0.66	0.72	23
Pro + iforest	25.36	26.7	5.17	0.36	0.64	0.74	19
Pro+ KNN	22.81	23.51	4.85	0.31	0.69	0.76	12
Pro + LOF	23.87	24.53	4.95	0.35	0.65	0.75	17
Pro + Histogram	40	33.93	5.82	0.39	0.61	0.64	30
Pro + OSVM	30.35	32.97	5.74	0.34	0.66	0.7	26
RNN LSTM							
RNN LSTM	24.18	734.08	27.09	0.68	0.32	0.33	31
RNN + iforest	23.34	814.47	28.54	0.42	0.58	0.3	32
RNN+ KNN	23.98	814.7	28.54	0.48	0.52	0.3	34
RNN + LOF	23.79	815.62	28.56	0.42	0.58	0.3	33
RNN + Histogram	24.24	817.52	28.59	0.58	0.42	0.29	36
RNN + OSVM	24.16	813.85	28.53	0.55	0.45	0.3	35
Conv LSTM							
Conv LSTM	6.55	100.08	10	0.22	0.78	0.78	9
Conv + iforest	8.91	120.16	10.96	0.30	0.70	0.74	18
Conv + KNN	11.23	160.1	12.65	0.40	0.60	0.69	27
Conv + LOF	6.45	118.57	10.89	0.30	0.70	0.77	11
Conv + Histogram	10.58	151.09	12.29	0.38	0.62	0.7	25
Conv + OSVM	11.17	164.45	12.82	0.42	0.58	0.68	28
DLM							
DLM	8.65	15.66	3.96	0.35	0.65	0.76	13
DLM + iforest	8.72	15.13	3.89	0.37	0.63	0.76	15
DLM + KNN	8.95	15.13	3.89	0.37	0.63	0.75	16
DLM + LOF	7.74	14.14	3.76	0.23	0.77	0.77	10
DLM + Histogram	8.65	15.66	3.96	0.36	0.64	0.76	14
DLM + OSVM	8.43	15.66	3.96	0.21	0.79	0.84	8
XgBoost							
XgBoost	1.45	3.42	1.85	0.17	0.83	0.99	5
XgBoost + iforest	1.08	2.67	1.63	0.17	0.83	1	4
XgBoost + KNN	1.03	2.25	1.5	0.16	0.84	1	3
XgBoost+ LOF	0.93	1.89	1.37	0.14	0.86	1	2
XgBoost + Histogram	1.45	3.42	1.85	0.18	0.82	0.99	6
XgBoost + OSVM	1.56	4.83	2.2	0.20	0.80	0.99	7
Proposed Model	0.9	1.78	1.2	0.12	0.88	1	1

For the VM workload prediction task, MAE, MSE, RMSE, MAPE and accuracy are utilized as the performance metrics. The summary of the best results obtained from the comparative performance study of various models with and without different anomaly detection techniques is shown in Table 6.8.

Many models, such as ARIMA, may be incapable of simulating the nonlinear components when the data becomes extremely volatile. The resulting error will have a significant impact. As a result, cloud multi-step advance load forecasting may be compromised. ANN

Table 6.8: Summary of the best obtained results from previous experiments

Models	RMSE	MAPE	Accuracy
XgBoost	1.37	0.14	0.86
DLM	3.96	0.21	0.79
ConvLSTM	10	0.22	0.78
Proposed Model	1.2	0.12	0.88



(a) Expected vs. Predicted on Train Data (b) Expected vs. Predicted on Test Data

Figure 6.8: CPU Utilization of Proposed Ensemble Model without Anomaly

enhances the prediction of nonlinear patterns. On the contrary, prediction requires the implementation of training data. The efficacy of ANN could potentially be compromised due to this limitation. Consequently, by combining the most advantageous attributes of the various models, the proposed ensemble model captures both linear and nonlinear components of the data more accurately than the alternative models. The top three models are chosen based on their performance and are utilized to build the proposed ensemble model. The proposed ensemble model captures both linear and nonlinear components of the data more accurately than other existing models by combining the best features of these three leading models. The CPU Utilization of Proposed ensemble model without anomaly is shown in Figure 6.8.

The comparison of various time series models based on accuracy is shown in Figure 6.9.

6.7.1 Comparison of Various Prediction Models

To compare the proposed ensemble model performance with and without anomalies, we trained various workload prediction models, such as ARIMA, XgBoost, Convolutional LSTM, Prophet, DLM and RNN LSTM models, which are used as baseline models;

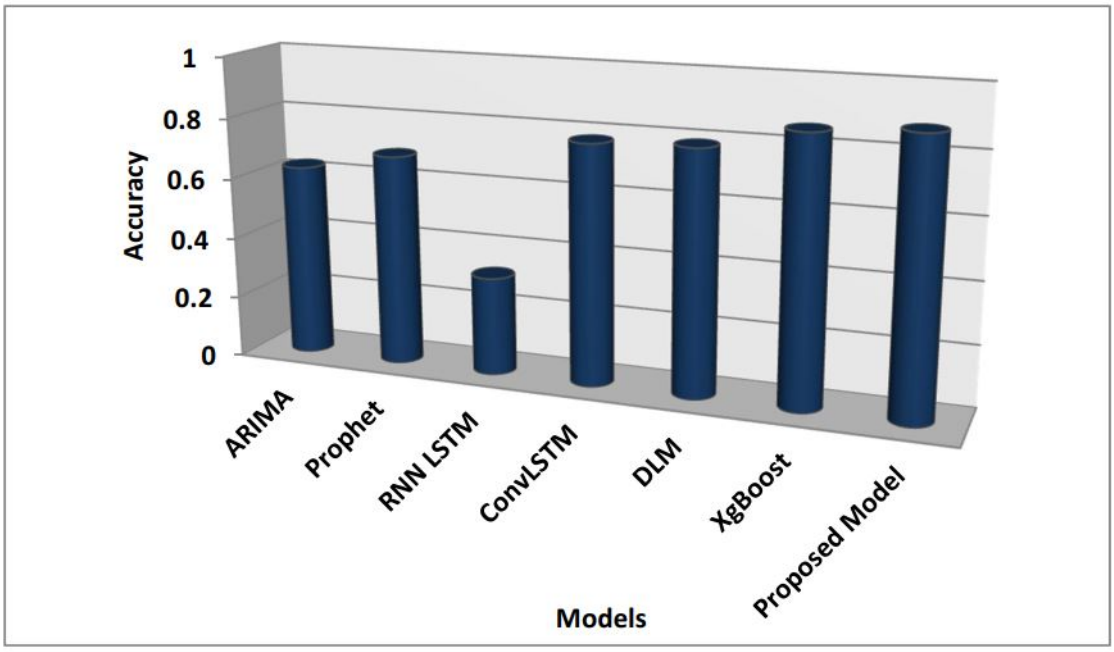


Figure 6.9: Comparison of various time series models based on accuracy

these models MSE, RMSE, MAE and MAPE error rates are shown in Table 6.7. Then the proposed ensemble model is compared with ARIMA, convolutional LSTM, DLM, XgBoost, Prophet and RNN LSTM models for VM workload prediction.

Upon comparing the mean absolute percentage error (MAPE) rate of our proposed ensemble method to that of existing methods, it becomes evident that our method exhibits a lower average absolute error rate. The reduced error rate can be attributed to our method's efficacy in identifying long-term dependencies, which leads to enhanced prediction accuracy and diminished absolute error.

The experimentation also utilized four distinct machine learning models— BiLSTM [34], ANN [135], CNN-LSTM [35] and EQNN [37] to assess the performance and precision of the ensemble model that was proposed. These methodologies were chosen because they are multivariate approaches built upon neural networks and deep learning concepts. Tests were performed utilizing a generated dataset. We conduct comparisons using the outcomes of one-step predictions, as this is how the majority of prior research has generated predictions. Table 6.9 summarizes the results of comparing the prediction errors of some of the existing approaches with the proposed model. The suggested model, which utilized multivariate time series data and deep learning techniques, effectively captured correlations across variables, leading to a substantial enhancement in prediction accuracy compared to previous studies.

Table 6.9: Comparison between existing and proposed model predictions

Models	MAPE	RMSE	MSE	Accuracy
CNN-LSTM [35]	0.19	2.9	3.42	0.812
Bi-LSTM[34]	0.16	2.3	2.41	0.842
ANN [135]	0.22	2.21	3.25	0.781
EQNN [37]	0.136	1.9	1.96	0.86
Proposed Model	0.12	1.2	1.78	0.88

To facilitate a comparative analysis with prior research, the mean absolute error of each is computed for the proposed method. The absolute error values for different approaches in the case of one-step prediction are illustrated in Figure 6.10. In contrast to the mean absolute error rates of alternative approaches, it is indisputable that our method exhibits the lowest absolute error rate on average.

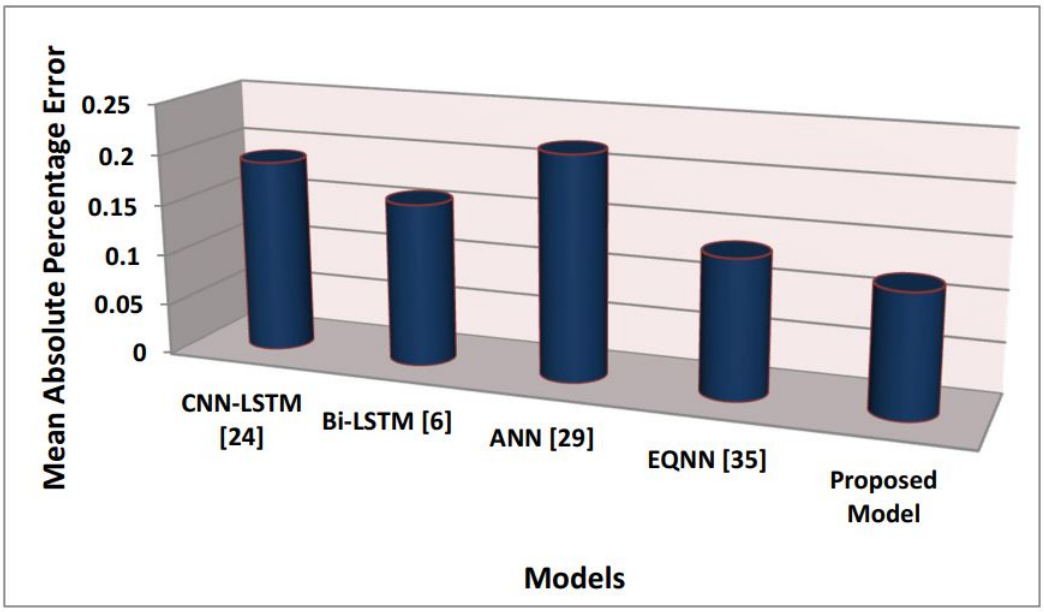


Figure 6.10: Comparison of proposed approach with various existing methods in regard of absolute error rates

The proposed ensemble model, which uses deep learning and multivariate time series data, is better than other hybrid studies because it effectively finds correlations between variables that are related to each other. This makes predictions a lot more accurate.

6.7.2 Forecasting the CPU Utilization

Figures 6.11 - 6.16 show the forecasted workload and the actual observations plot of different models used for workload prediction. Figure 6.8 illustrates both a long-term CPU usage prediction time series curve and the actual CPU usage workload curve. The

interval length in this case is five minutes.

It has been observed from Figure 6.8 that the proposed ensemble model predicts a CPU utilization value too close to the actual workload. Figure 6.8a shows the actual(expected) versus predicted load on the train data, and Figure 6.8b shows the results on the test data. So, according to the forecasts, the true CPU utilization rate observations are accurately predicted using the proposed ensemble model with an accuracy of approximately 88%. The XgBoost model performs better as compared to other existing time series models as shown in Figure 6.12 and provides an accuracy of about 86%.

As shown in Figure 6.15, the DLM model also performs admirably and predicts CPU utilization value of host that is excessively near to the actual workload and provides accuracy of about 79%. The ARIMA and ConvLSTM models also exhibit superior performance and provide CPU utilization predictions that closely resemble the actual CPU utilization predictions, as depicted in Figure 6.11 and Figure 6.14, respectively. According to the topsis results shown in Table 6.7, our proposed ensemble model outperforms the other methods in MAE and RMSE rates. As shown in Figure 6.16, the RNN LSTM model exhibits the highest error rate among the models evaluated and performs poorly in comparison to the others and provides only 58% accuracy.

Furthermore, its predictions regarding CPU utilization deviate significantly from the actual values, yielding a nearly linear graph for CPU prediction. It should be noted that the Prophet model also exhibits a greater error rate in comparison to the other cutting-edge workload prediction models illustrated in Figure 6.13. So, ARIMA, XgBoost, ConvLSTM and DLM models exhibit superior performance in comparison to Prophet and RNN LSTM models.

By examining the predicted and actual CPU utilization through the use of visual representations, we assess the performance of the proposed ensemble model in comparison to previous artificial neural network models. Comparing the MAE, MSE, RMSE, MAPE and accuracy of our proposed model to those of previously developed models reveals that our proposed method produces a lower average error rate.

For validation purposes, the proposed ensemble model is also tested on the Bitbrains dataset[19]. The result obtained is shown in Figure 6.17 and achieves 75% accuracy.

6.7.3 Case Study

The case study of proposed approach focuses on improving host load prediction in fog computing environment, which is crucial for optimizing resource utilization and meeting

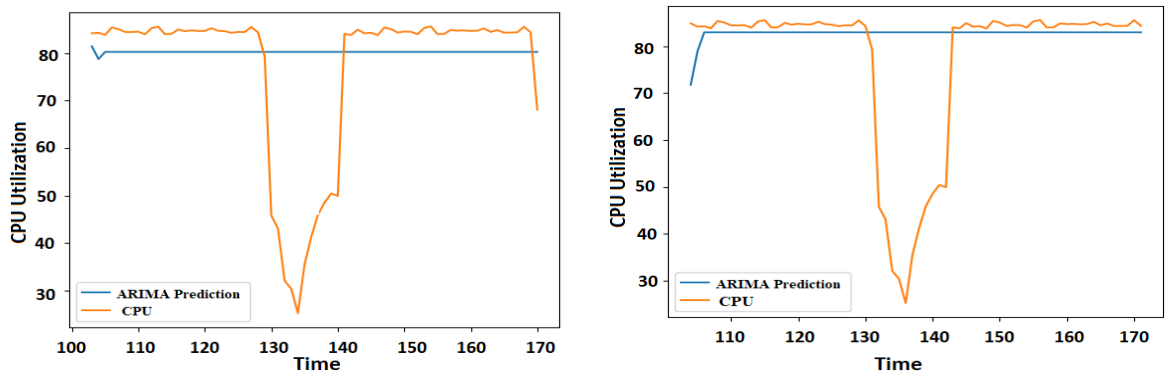


Figure 6.11: CPU Utilization of ARIMA Model with and without anomaly a) With Anomaly b) Without Anomaly

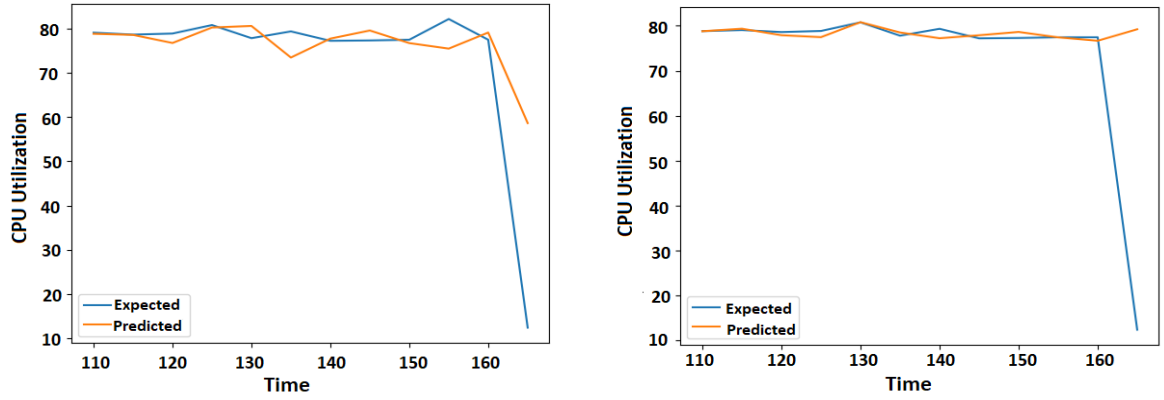


Figure 6.12: CPU Utilization of XgBoost Model with and without anomaly a) With Anomaly b) Without Anomaly

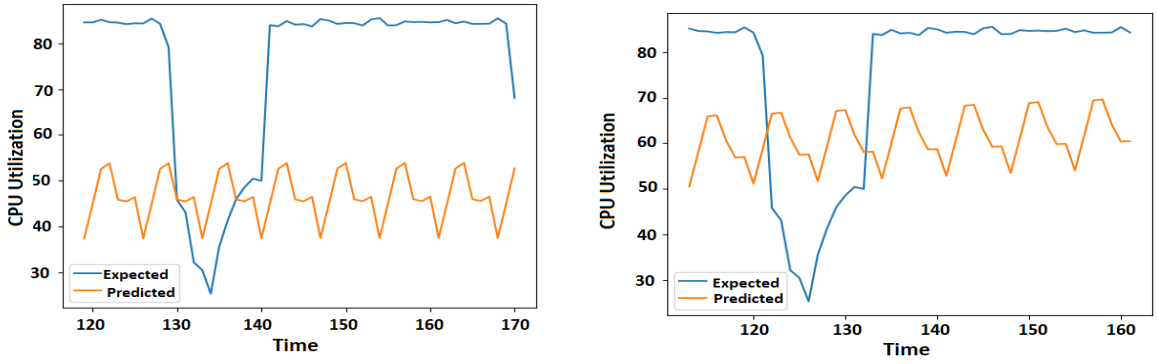


Figure 6.13: CPU Utilization of Prophet Model with and without anomaly a) With Anomaly b) Without Anomaly

service level agreements. Three distinct categories of cases are evaluated: Different scenarios involve varying number of virtual machines operating in parallel, and these VMs

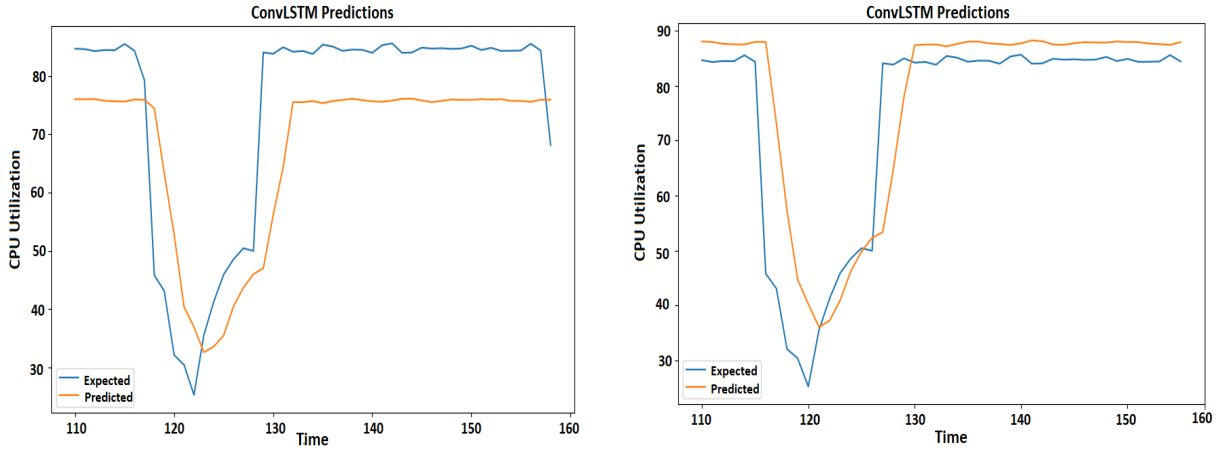


Figure 6.14: CPU Utilization of ConvLSTM Model with and without anomaly a) With Anomaly b) Without Anomaly

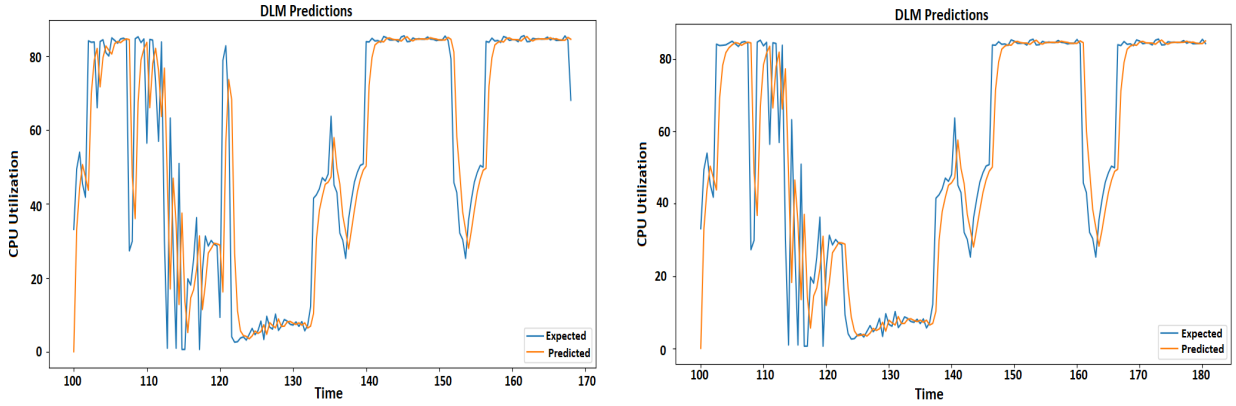


Figure 6.15: CPU Utilization of DLM Model with and without anomaly a) With Anomaly b) Without Anomaly

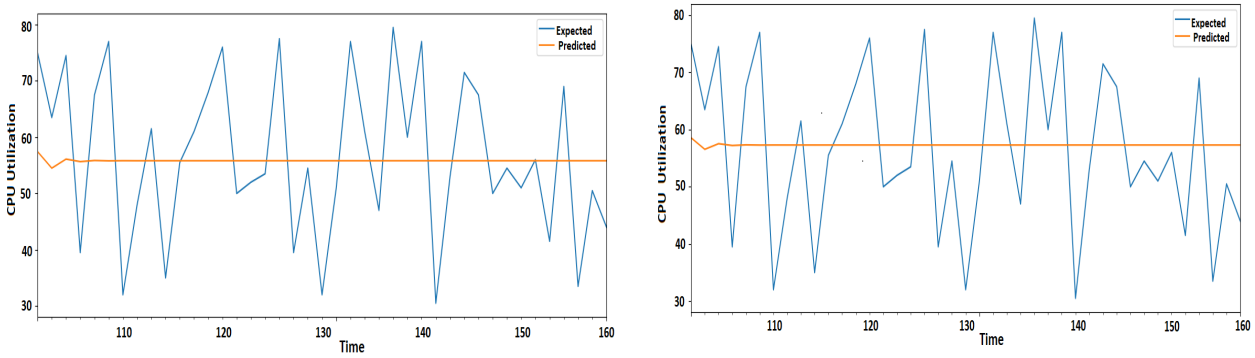


Figure 6.16: CPU Utilization of RNN LSTM Model with and without anomaly a) With Anomaly b) Without Anomaly

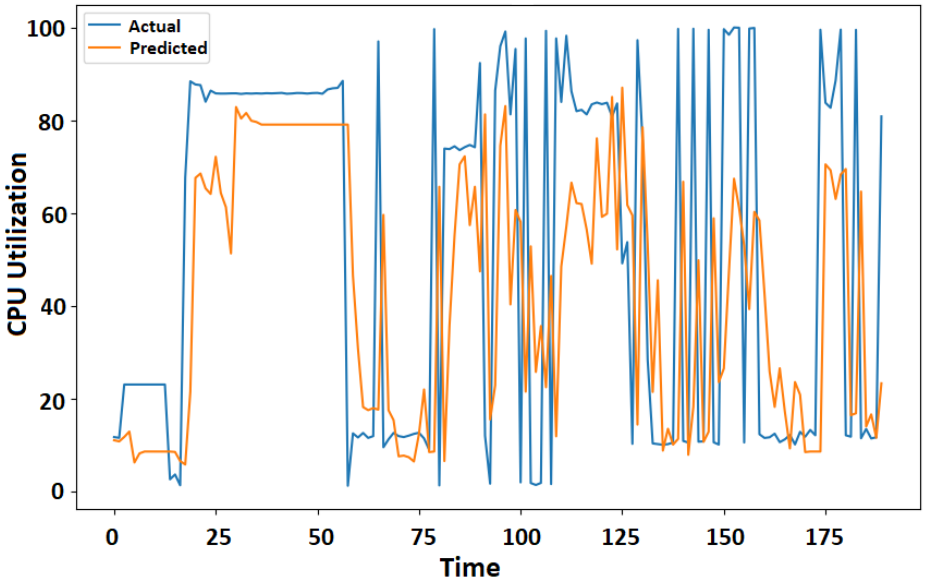


Figure 6.17: Testing of the proposed ensemble model on Bitbrains dataset

are deployed on two distinct types of platforms as shown in Table 6.1. In the first scenario, four VMs operate concurrently: two VMs on the first platform and other two are on the second platform. In the second case, six VMs operate concurrently, with three VMs being executed on the initial platform and the remaining three on the second. In the third scenario, ten VMs are operating concurrently, with five on the initial platform and the remaining five on the secondary platform.

We evaluated the performance of our proposed model in various scenarios by augmenting the number of virtual machines (VMs) in real time scenario at different time intervals. Increasing the number of virtual machines (VMs) produces a huge amount of dataset for model training and testing purpose, further enhancing the performance and accuracy of our proposed load prediction model. The performance of the proposed ensemble model in different cases is shown in Table 6.10.

Table 6.10: Different cases considered for evaluation

Case Study	**No. of VMs running in Parallel**	**Dataset (10 performance parameters)**	**RMSE**	**MAPE**	**Accuracy**
Case I	4 VMs (2 on Platform 1 and other 2 on 2nd Platform)	4500 records	3.2	0.21	0.793
Case II	6 VMs (3 on Platform 1 and other 3 on 2nd Platform)	7000 records	2.5	0.16	0.841
Case III	10 VMs (5 on Platform 1 and other 5 on 2nd Platform)	10,000 records	1.2	0.12	0.88

6.8 Summary

This work introduces a multivariate time series ensemble model for the prediction of CPU utilization across multiple fog nodes. The model employs a variety of anomaly detection methods to address inconsistencies in load forecasting from large data volumes. These techniques identify anomalous data and select top-performing time series models based on TOPSIS scores to create the ensemble model. The model's efficiency is assessed using MAE, MSE, RMSE, MAPE and accuracy metrics. Results show that the proposed ensemble model outperforms existing models with lower error rates and higher accuracy.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

This work introduces a novel reliability framework that involves multiple implementation phases. The framework commences with the generation of virtual machines using a variety of random settings on the command line, followed by the generation of a dataset, the application of machine learning techniques for load prediction on hosts and the analysis of the results. Precision, recall, and accuracy were the criteria used to assess the models.

An ensemble model was developed by combining ten machine learning models, resulting in the most precise and optimal results for the classification of host load. The Random Forest (RF), AdaBoost (AB), Gradient Boost (GB), and Decision Tree (DT) models all performed equally well across three case studies in terms of enhanced model performance as a result of normalizing datasets. Nevertheless, the proposed ensemble model demonstrated a marginally higher level of performance, attaining an accuracy of approximately 82%. The 10-fold cross-validation method was employed to further assess the ensemble model's robustness.

The emergence of container migration and containerization as the next significant revolution in cloud and fog computing environment necessitates the transfer of containers from overloaded hosts to new hosts in order to ensure that there are sufficient resources to execute consumer applications at the network edges and in fog computing and mobile edge clouds. An extensive investigation of algorithms to manage the excessive energy consumption of hosts has not been conducted, despite the increasing prominence of containers. The optimization of the energy consumption efficacy of hosts continues to be a critical and challenging endeavour.This investigation investigates methods for optimizing fog computing container administration, with an emphasis on minimizing the load on hosts.

Two frameworks were recommended: one for container selection from the overloaded host and another for container migration. The energy consumption was significantly reduced

by 10.89% as a result of the 35% reduction in container migrations that the proposed container selection algorithm achieved. Furthermore, the proposed container migrations were distinguished from existing methods by the substantial variations in power consumption, temperature, frequency, and CPU utilization. The container migration algorithm's dynamic inertia weight-based PSO (DIWPSO) technique offered swifter solutions, particularly for the computational requirements of large-scale applications. The DIWPSO algorithm outperformed existing optimization methods, such as PSO, ABC, and E-ABC, on the benchmarks for container migration, as demonstrated by numerical analysis.

Additionally, this study introduces a multilevel time series ensemble model that is capable of predicting CPU utilization for multiple hosts that are operating on fog nodes. The data volume was so extensive that a variety of anomaly detection methods were implemented to mitigate potential inconsistencies in load forecasting.

Anomaly detection techniques were implemented to identify anomalous data in the dataset that was generated, and a variety of time series-based models were implemented to forecast workload. The ensemble model was developed by selecting the best three models based on the TOPSIS score. The proposed model's efficacy was verified through the evaluation and comparison of MAE, MSE, RMSE, MAPE, and accuracy metrics with other state-of-the-art models. The proposed ensemble model demonstrated a higher level of accuracy and a lower error rate in comparison to other existing workload prediction models, as evidenced by the results.

7.2 Scope for future work

The following are a number of potential future research directions for this work:

1. **Address complexity and accessibility:** Investigate methods to simplify the dynamic inertia weight-based PSO (DIWPSO) algorithm to enhance accessibility for smaller organizations.

2. **Enhance security:** Focus on mitigating security concerns associated with container-based virtualization, particularly due to shared kernel access. Explore strategies for improving hardware security in a container environment.

3. **Advanced container technology:** Continue research into container technology, particularly in the early stages of development. Prioritize advancements in failure management and cluster-based container orchestration using technologies such as Kubernetes.

4. **Optimize Container Migration:** Develop and refine techniques to further op-

timize container migrations, aiming to improve resource utilization, reduce operational costs, and enhance performance.

5. **Improved Isolation and Security Measures:** When deploying containers in data centers, emphasize secure practices, including cgroups, file systems, network, and memory isolation, to limit container access and safeguard hardware.
6. **Refine Optimization Techniques:** Continue refining other optimization techniques to address evolving challenges in energy efficiency, especially in real-time computing environment.
7. **Explore Real-World Deployment Insights:** Provide insights into deploying container technology in real-world computing scenarios, with a focus on practical applications in cloud, IoT, fog, and edge computing environment.

List of Publications

1. Shabnam Bawa, Prashant Singh Rana, and Raj Kumar Tekchandani, *"Multivariate time series ensemble model for load prediction on hosts using anomaly detection techniques"*, Cluster Computing, Springer, 1-24, 2024. [SCI, IF 3.6, Published]
2. Shabnam Bawa, Prashant Singh Rana, and Raj Kumar Tekchandani, *"Migration of Containers on the Basis of Load Prediction with Dynamic Inertia Weight based PSO Algorithm"*, Cluster Computing, Springer, 1-25, 2024. [SCI, IF 3.6, Published]
3. Shabnam Bawa, Prashant Singh Rana, and Raj Kumar Tekchandani, *"Multilevel Ensemble Model for Load Prediction on Hosts in Fog Computing Environment"*, Computing and Informatics, Slovak Academy of Sciences, 1-28, 2024. [SCIE, IF 0.7, Accepted (In Press)]

References

[1] Rodrigo N Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE transactions on cloud computing*, 3(4):449–458, 2014.

[2] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

[3] Samir K Safi and Olajide Idris Sanusi. A hybrid of artificial neural network, exponential smoothing, and arima models for covid-19 time series forecasting. *Model Assisted Statistics and Applications*, 16(1):25–35, 2021.

[4] Shaifu Gupta, Aroor Dinesh Dileep, and Timothy A Gonsalves. A joint feature selection framework for multivariate resource usage prediction in cloud servers using stability and prediction performance. *The Journal of Supercomputing*, 74:6033–6068, 2018.

[5] Lele Ma, Shanhe Yi, Nancy Carter, and Qun Li. Efficient live migration of edge services leveraging container layered storage. *IEEE Transactions on Mobile Computing*, 18(9):2020–2033, 2018.

[6] Xiao-Fang Liu, Zhi-Hui Zhan, Jeremiah D Deng, Yun Li, Tianlong Gu, and Jun Zhang. An energy efficient ant colony system for virtual machine placement in cloud computing. *IEEE transactions on evolutionary computation*, 22(1):113–128, 2016.

[7] Zitong Ma, Sujie Shao, Shaoyong Guo, Zhili Wang, Feng Qi, and Ao Xiong. Container migration mechanism for load balancing in edge network under power internet of things. *IEEE Access*, 8:118405–118416, 2020.

[8] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42, 2017.

[9] Hasan Ali Khattak, Hafsa Arshad, Ghufran Ahmed, Sohail Jabbar, Abdullahi Mohamud Sharif, Shehzad Khalid, et al. Utilization and load balancing in fog servers for health applications. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–12, 2019.

[10] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. Efficient virtual machine sizing for hosting containers as a service (services 2015). In *2015 IEEE World Congress on Services*, pages 31–38. IEEE, 2015.

[11] Hsin-Cheng Huang and Noel Cressie. Spatio-temporal prediction of snow water equivalent using the kalman filter. *Computational Statistics & Data Analysis*, 22(2):159–175, 1996.

[12] Maryam Chehelgerdi-Samani and Faramarz Safi-Esfahani. Pcvm. arima: predictive consolidation of virtual machines applying arima method. *The Journal of Supercomputing*, 77:2172–2206, 2021.

[13] Bhatti Dhaval and Anuradha Deshpande. Short-term load forecasting using method of multiple linear regression. 2021.

[14] Mandeep Kaur, Pankaj Deep Kaur, and Sandeep Kumar Sood. Energy efficient iot-based cloud framework for early flood prediction. *Natural Hazards*, 109:2053–2076, 2021.

[15] Man-Chun Tan, Sze Chun Wong, Jian-Min Xu, Zhan-Rong Guan, and Peng Zhang. An aggregation approach to short-term traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 10(1):60–69, 2009.

[16] Deepak Janardhanan and Enda Barrett. Cpu workload forecasting of machines in data centers using lstm recurrent neural networks and arima models. In *2017 12th international conference for internet technology and secured transactions (ICITST)*, pages 55–60. IEEE, 2017.

[17] Jin Fan, Ke Zhang, Yipan Huang, Yifei Zhu, and Baiping Chen. Parallel spatio-temporal attention-based tcn for multivariate time series prediction. *Neural Computing and Applications*, 35(18):13109–13118, 2023.

[18] Jitendra Kumar, Ashutosh Kumar Singh, and Rajkumar Buyya. Self directed learning based workload forecasting model for cloud resource management. *Information Sciences*, 543:345–366, 2021.

[19] Tahseen Khan, Wenhong Tian, Shashikant Ilager, and Rajkumar Buyya. Workload forecasting and energy state estimation in cloud data centers: Ml-centric approach. *Future Generation Computer Systems*, 128:320–332, 2022.

[20] Yashwant Singh Patel and Rajiv Misra. Performance comparison of deep vm workload prediction approaches for cloud. In *Progress in Computing, Analytics, and Networking*, pages 149–160. Springer, 2018.

[21] Jiechao Gao, Haoyu Wang, and Haiying Shen. Machine learning based workload prediction in cloud computing. In *2020 29th international conference on computer communications and networks (ICCCN)*, pages 1–9. IEEE, 2020.

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

[23] Fahimeh Ramezani and Mohsen Naderpour. A fuzzy virtual machine workload

prediction method for cloud environments. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2017.

[24] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16:7–18, 2014.

[25] Feng Qiu, Bin Zhang, and Jun Guo. A deep learning approach for vm workload prediction in the cloud. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 319–324. IEEE, 2016.

[26] Jhu-Jyun Jheng, Fan-Hsun Tseng, Han-Chieh Chao, and Li-Der Chou. A novel vm workload prediction using grey forecasting model in cloud data center. In *The International Conference on Information Networking 2014 (ICOIN2014)*, pages 40–45. IEEE, 2014.

[27] Yongjia Yu, Vasu Jindal, I-Ling Yen, and Farokh Bastani. Integrating clustering and learning for improved workload prediction in the cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 876–879. IEEE, 2016.

[28] Vincenzo Eramo and Tiziana Catena. Application of an innovative convolutional/l-stm neural network for computing resource allocation in nfv network architectures. *IEEE Transactions on Network and Service Management*, 19(3):2929–2943, 2022.

[29] Eddy Caron, Frederic Desprez, and Adrian Muresan. Forecasting for grid and cloud computing on-demand resources based on pattern matching. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 456–463. IEEE, 2010.

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

[31] Jing Chen, Yinglong Wang, et al. A hybrid method for short-term host utilization prediction in cloud computing. *Journal of Electrical and Computer Engineering*, 2019, 2019.

[32] Shaifu Gupta and Dileep Aroor Dinesh. Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks. In *2017 IEEE international conference on advanced networks and telecommunications systems (ANTS)*, pages 1–6. IEEE, 2017.

[33] Minxian Xu, Chenghao Song, Huaming Wu, Sukhpal Singh Gill, Kejiang Ye, and Chengzhong Xu. esdnn: deep neural network based multivariate workload predic-

tion in cloud computing environments. *ACM Transactions on Internet Technology (TOIT)*, 22(3):1–24, 2022.

[34] Nhat-Minh Dang-Quang and Myungsik Yoo. Multivariate deep learning model for workload prediction in cloud computing. In *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 858–862. IEEE, 2021.

[35] Soukaina Ouhame, Youssef Hadi, and Arif Ullah. An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model. *Neural Computing and Applications*, 33:10043–10055, 2021.

[36] SR Shishira and A Kandasamy. Beem-nn: An efficient workload optimization using bee mutation neural network in federated cloud environment. *Journal of Ambient Intelligence and Humanized Computing*, 12:3151–3167, 2021.

[37] Ashutosh Kumar Singh, Deepika Saxena, Jitendra Kumar, and Vrinda Gupta. A quantum approach towards the adaptive prediction of cloud workloads. *IEEE Transactions on Parallel and Distributed Systems*, 32(12):2893–2905, 2021.

[38] Zheng Huang, Jiajun Peng, Huijuan Lian, Jie Guo, Weidong Qiu, et al. Deep recurrent model for server load and performance prediction in data center. *Complexity*, 2017, 2017.

[39] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh. Long short term memory recurrent neural network (lstm-rnn) based workload forecasting model for cloud datacenters. *Procedia Computer Science*, 125:676–682, 2018.

[40] Nhuan Tran, Thang Nguyen, Binh Minh Nguyen, and Giang Nguyen. A multivariate fuzzy time series resource forecast model for clouds using lstm and data correlation analysis. *Procedia Computer Science*, 126:636–645, 2018.

[41] Jiechao Gao, Haoyu Wang, and Haiying Shen. Task failure prediction in cloud data centers using deep learning. *IEEE transactions on services computing*, 15(3):1411–1422, 2020.

[42] Md Ebtidaul Karim, Mirza Mohd Shahriar Maswood, Sunanda Das, and Abdullah G Alharbi. Bhyprec: a novel bi-lstm based hybrid recurrent neural network model to predict the cpu workload of cloud virtual machine. *IEEE Access*, 9:131476–131495, 2021.

[43] Rong Zhang, Yaxing Chen, Bo Dong, Feng Tian, and Qinghua Zheng. A genetic algorithm-based energy-efficient container placement strategy in caas. *IEEE Access*, 7:121360–121373, 2019.

[44] Zhaorui Wu, Yuhui Deng, Hao Feng, Yi Zhou, and Geyong Min. Blender: A traffic-aware container placement for containerized data centers. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 986–989.

IEEE, 2021.

[45] Yanal Alahmad, Anjali Agarwal, Marzia Zaman, and Nishith Goel. Container placement for resource utilization in cyber physical cloud systems. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2021.

[46] Yang Hu, Cees De Laat, and Zhiming Zhao. Multi-objective container deployment on heterogeneous clusters. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 592–599. IEEE, 2019.

[47] Yang Hu, Huan Zhou, Cees de Laat, and Zhiming Zhao. Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. *Future Generation Computer Systems*, 102:562–573, 2020.

[48] Adrian Asensio, Xavier Masip-Bruin, Jordi Garcia, and Sergio Sánchez. On the optimality of concurrent container clusters scheduling over heterogeneous smart environments. *Future Generation Computer Systems*, 118:157–169, 2021.

[49] Mohamed K Hussein, Mohamed H Mousa, and Mohamed A Alqarni. A placement architecture for a container as a service (caas) in a cloud environment. *Journal of Cloud Computing*, 8:1–15, 2019.

[50] Weiwen Zhang, Lei Chen, Jinzhou Luo, and Jianqi Liu. A two-stage container management in the cloud for optimizing the load balancing and migration cost. *Future Generation Computer Systems*, 135:303–314, 2022.

[51] Tao Shi, Hui Ma, and Gang Chen. Energy-aware container consolidation based on pso in cloud data centers. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[52] Neeraj Kumar, Gagangeet Singh Aujla, Sahil Garg, Kuljeet Kaur, Rajiv Ranjan, and Saurabh Kumar Garg. Renewable energy-based multi-indexed job classification and container management scheme for sustainability of cloud data centers. *IEEE Transactions on Industrial Informatics*, 15(5):2947–2957, 2018.

[53] Boxiong Tan, Hui Ma, and Yi Mei. Novel genetic algorithm with dual chromosome representation for resource allocation in container-based clouds. In *2019 IEEE 12th international conference on cloud computing (CLOUD)*, pages 452–456. IEEE, 2019.

[54] Chao Chen, Kejing He, and Quanlong Guan. Minimum migration time selection algorithm for container consolidation. In *2018 IEEE International Conference on Information and Automation (ICIA)*, pages 1664–1668. IEEE, 2018.

[55] Prateek Chhikara, Rajkumar Tekchandani, Neeraj Kumar, and Mohammad S Obaidat. An efficient container management scheme for resource-constrained intelligent iot devices. *IEEE Internet of Things Journal*, 8(16):12597–12609, 2020.

[56] Kai Li, Chunlei Chang, Kai Yun, and Jianye Zhang. Research on container mi-

gration mechanism of power edge computing on load balancing. In *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, pages 386–390. IEEE, 2021.

[57] Taewoon Kim, Motassem Al-Tarazi, Jenn-Wei Lin, and Wooyeol Choi. Optimal container migration for mobile edge computing: Algorithm, system design and implementation. *IEEE Access*, 9:158074–158090, 2021.

[58] Gursharan Singh, Parminder Singh, Anas Motii, and Mustapha Hedabou. A secure and lightweight container migration technique in cloud computing. *Journal of King Saud University-Computer and Information Sciences*, 36(1):101887, 2024.

[59] Ashutosh Kumar Singh, Smruti Rekha Swain, and Chung Nan Lee. A metaheuristic virtual machine placement framework toward power efficiency of sustainable cloud environment. *Soft Computing*, 27(7):3817–3828, 2023.

[60] Sudheer Mangalampalli, Ganesh Reddy Karri, Mohit Kumar, Osama Ibrahim Khalaf, Carlos Andres Tavera Romero, and GhaidaMuttashar Abdul Sahib. Drlbtsa: Deep reinforcement learning based task-scheduling algorithm in cloud computing. *Multimedia Tools and Applications*, 83(3):8359–8387, 2024.

[61] Simon Pickartz, Carsten Clauss, Jens Breitbart, Stefan Lankes, and Antonello Monti. Prospects and challenges of virtual machine migration in hpc. *Concurrency and Computation: Practice and Experience*, 30(9):e4412, 2018.

[62] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and YC Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th international middleware conference*, pages 1–13, 2016.

[63] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. A framework and algorithm for energy efficient container consolidation in cloud data centers. In *2015 IEEE International conference on data science and data intensive systems*, pages 368–375. IEEE, 2015.

[64] Flávio Ramalho and Augusto Neto. Virtualization at the network edge: A performance comparison. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2016.

[65] Gursharan Singh and Pooja Gupta. A review on migration techniques and challenges in live virtual machine migration. In *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 542–546. IEEE, 2016.

[66] Damien Carver, Julien Sopena, and Sebastien Monnet. Acdc: Advanced consolidation for dynamic containers. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2017.

[67] Ilias Mavridis and Helen Karatza. Performance and overhead study of containers

running on top of virtual machines. In *2017 IEEE 19th Conference on Business Informatics (CBI)*, volume 2, pages 32–38. IEEE, 2017.

[68] Shalu Singh and Dinesh Singh. A bio-inspired vm migration using re-initialization and decomposition based-whale optimization. *ICT Express*, 9(1):92–99, 2023.

[69] Mustafa Can Çavdar, Ibrahim Korpeoglu, and Özgür Ulusoy. A utilization based genetic algorithm for virtual machine placement in cloud systems. *Computer Communications*, 214:136–148, 2024.

[70] Sayyid Shahab Nabavi, Sukhpal Singh Gill, Minxian Xu, Mohammad Masdari, and Peter Garraghan. Tractor: Traffic-aware and power-efficient virtual machine placement in edge-cloud data centers using artificial bee colony optimization. *International Journal of Communication Systems*, 35(1):e4747, 2022.

[71] Mohd Sha Alam Khan and R Santhosh. Hybrid optimization algorithm for vm migration in cloud computing. *Computers and Electrical Engineering*, 102:108152, 2022.

[72] Deepika Saxena, Ishu Gupta, Jitendra Kumar, Ashutosh Kumar Singh, and Xiaoqing Wen. A secure and multiobjective virtual machine placement framework for cloud data center. *IEEE Systems Journal*, 16(2):3163–3174, 2021.

[73] Shanky Goyal, Shashi Bhushan, Yogesh Kumar, Abu ul Hassan S Rana, Muhammad Raheel Bhutta, Muhammad Fazal Ijaz, and Youngdoo Son. An optimized framework for energy-resource allocation in a cloud environment based on the whale optimization algorithm. *Sensors*, 21(5):1583, 2021.

[74] Meng Yue. An integrated anomaly detection method for load forecasting data under cyberattacks. In *2017 IEEE power & energy society general meeting*, pages 1–5. IEEE, 2017.

[75] Lenka Benova and Ladislav Hudec. Web server load prediction and anomaly detection from hypertext transfer protocol logs. *International Journal of Electrical and Computer Engineering (IJECE)*, 13(5):5165–5178, 2023.

[76] Shaifu Gupta, Neha Muthiyan, Siddhant Kumar, Aditya Nigam, and Dileep Aroor Dinesh. A supervised deep learning framework for proactive anomaly detection in cloud workloads. In *2017 14th IEEE India Council International Conference (INDICON)*, pages 1–6. IEEE, 2017.

[77] Yongmin Tan, Xiaohui Gu, and Haixun Wang. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 173–182, 2010.

[78] Peter Appiahene, Yaw Marfo Missah, and Ussiph Najim. Predicting bank operational efficiency using machine learning algorithm: comparative study of decision tree, random forest, and neural networks. *Advances in fuzzy systems*,

2020(1):8581202, 2020.

[79] Y Liang and X Ma. iacp-ge: accurate identification of anticancer peptides by using gradient boosting decision tree and extra tree. *SAR and QSAR in Environmental Research*, 34(1):1–19, 2023.

[80] Dewan Md Farid, Nabila Sabrin Sworna, Ruhul Amin, Nazifa Sadia, Moshiur Rahman, Nazmul Khan Liton, Md Saddam Hossain Mukta, and Swakkhar Shatabda. Boosting k-nearest neighbour (knn) classification using clustering and adaboost methods. In *2022 IEEE Region 10 Symposium (TENSYMP)*, pages 1–6. IEEE, 2022.

[81] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien. Linear discriminant analysis: A detailed tutorial. *AI communications*, 30(2):169–190, 2017.

[82] Dehua Wang, Yang Zhang, and Yi Zhao. Lightgbm: an effective mirna classification method in breast cancer patients. In *Proceedings of the 2017 international conference on computational biology and bioinformatics*, pages 7–11, 2017.

[83] Barenya Bikash Hazarika, Deepak Gupta, and Parashjyoti Borah. An intuitionistic fuzzy kernel ridge regression classifier for binary classification. *Applied Soft Computing*, 112:107816, 2021.

[84] Shenglei Chen, Geoffrey I Webb, Linyuan Liu, and Xin Ma. A novel selective naïve bayes algorithm. *Knowledge-Based Systems*, 192:105361, 2020.

[85] Zhiyuan Zhang, Xuehai Tang, Jizhong Han, and Peng Wang. Sibyl: Host load prediction with an efficient deep learning model in cloud computing. In *Algorithms and Architectures for Parallel Processing: 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part II 18*, pages 226–237. Springer, 2018.

[86] Mustafa Daraghmeh, Anjali Agarwal, Ricardo Manzano, and Marzia Zaman. Time series forecasting using facebook prophet for cloud resource management. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2021.

[87] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 17–32, 2017.

[88] Hoang Minh Nguyen, Gaurav Kalra, and Daeyoung Kim. Host load prediction in cloud computing using long short-term memory encoder–decoder. *The Journal of Supercomputing*, 75(11):7592–7605, 2019.

[89] Weixin Luo, Wen Liu, and Shenghua Gao. Remembering history with convolutional lstm for anomaly detection. In *2017 IEEE International conference on multimedia and expo (ICME)*, pages 439–444. IEEE, 2017.

[90] Santhanam Ramraj, Nishant Uzir, R Sunil, and Shatadeep Banerjee. Experimenting xgboost algorithm for prediction and classification of different datasets. *International Journal of Control Theory and Applications*, 9(40):651–662, 2016.

[91] Roberto Rivera. A dynamic linear model to forecast hotel registrations in puerto rico using google trends data. *Tourism Management*, 57:12–20, 2016.

[92] Pawan Whig, Ketan Gupta, Nasmin Jiwani, Hruthika Jupalle, Shama Kouser, and Naved Alam. A novel method for diabetes classification and prediction with pycaret. *Microsystem Technologies*, 29(10):1479–1487, 2023.

[93] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[94] Huansheng Ning, Hong Liu, Jianhua Ma, Laurence T Yang, and Runhe Huang. Cybermatics: Cyber–physical–social–thinking hyperspace based science and technology. *Future generation computer systems*, 56:504–522, 2016.

[95] Rudyar Cortés, Xavier Bonnaire, Olivier Marin, and Pierre Sens. Stream processing of healthcare sensor data: studying user traces to identify challenges from a big data perspective. *Procedia Computer Science*, 52:1004–1009, 2015.

[96] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[97] Eder Pereira, Ivânia A Fischer, Roseclea D Medina, Emmanuell D Carreno, and Edson Luiz Padoin. A load balancing algorithm for fog computing environments. In *Latin American High Performance Computing Conference*, pages 65–77. Springer, 2019.

[98] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of network and computer applications*, 52:11–25, 2015.

[99] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH computer architecture news*, 35(2):13–23, 2007.

[100] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[101] Amany Abdelsamea, Ali A El-Moursy, Elsayed E Hemayed, and Hesham Eldeeb. Virtual machine consolidation enhancement using hybrid regression algorithms.

Egyptian Informatics Journal, 18(3):161–170, 2017.

[102] Xialin Liu, Junsheng Wu, Gang Sha, and Shuqin Liu. Virtual machine consolidation with minimization of migration thrashing for cloud data centers. *Mathematical Problems in Engineering*, 2020, 2020.

[103] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. Containercloudsim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience*, 47(4):505–521, 2017.

[104] Rahul Yadav, Weizhe Zhang, Keqin Li, Chuanyi Liu, and Asif Ali Laghari. Managing overloaded hosts for energy-efficiency in cloud data centers. *Cluster Computing*, 24(3):2001–2015, 2021.

[105] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[106] Orestis Akrivopoulos, Na Zhu, Dimitrios Amaxilatis, Christos Tselios, Aris Anagnostopoulos, and Ioannis Chatzigiannakis. A fog computing-oriented, highly scalable iot framework for monitoring public educational buildings. In *2018 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2018.

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

[108] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE, 2015.

[109] Zhibo Cao and Shoubin Dong. An energy-aware heuristic framework for virtual machine consolidation in cloud computing. *The Journal of Supercomputing*, 69:429–451, 2014.

[110] Willis Dale, F Arkodeb Dasgupta, and Suman Banerjee. Paradrop: a multi-tenant platform for dynamically installed third party services on home gateways. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, 2014.

[111] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate iot edge computing with lightweight virtualization. *IEEE network*, 32(1):102–111, 2018.

[112] Qi Zhang, Ling Liu, Calton Pu, Qiwei Dou, Liren Wu, and Wei Zhou. A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages

178–185. IEEE, 2018.

[113] Zhou Zhou, Zhi-gang Hu, Jun-yang Yu, Jemal Abawajy, and Morshed Chowdhury. Energy-efficient virtual machine consolidation algorithm in cloud data centers. *Journal of Central South University*, 24(10):2331–2341, 2017.

[114] Doug Chamberlain. Containers vs. virtual machines (vms): what's the difference, 2018.

[115] Deepak Puthal, Mohammad S Obaidat, Priyadarsi Nanda, Mukesh Prasad, Saraju P Mohanty, and Albert Y Zomaya. Secure and sustainable load balancing of edge data centers in fog computing. *IEEE Communications Magazine*, 56(5):60–65, 2018.

[116] Carlo Puliafito, Carlo Vallati, Enzo Mingozzi, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Container migration in the fog: A performance evaluation. *Sensors*, 19(7):1488, 2019.

[117] Jianen Yan, Hongli Zhang, Haiyan Xu, and Zhaoxin Zhang. Discrete pso-based workload optimization in virtual machine placement. *Personal and Ubiquitous Computing*, 22:589–596, 2018.

[118] Yong Feng, Gui-Fa Teng, Ai-Xin Wang, and Yong-Mei Yao. Chaotic inertia weight in particle swarm optimization. In *Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007)*, pages 475–475. IEEE, 2007.

[119] Russell C Eberhart and Yuhui Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, volume 1, pages 94–100. IEEE, 2001.

[120] Xuedan Liu, Qiang Wang, Haiyan Liu, and Lili Li. Particle swarm optimization with dynamic inertia weight and mutation. In *2009 Third International Conference on Genetic and Evolutionary Computing*, pages 620–623. IEEE, 2009.

[121] Jianbin Xin, Guimin Chen, and Yubao Hai. A particle swarm optimizer with multi-stage linearly-decreasing inertia weight. In *2009 International joint conference on computational sciences and optimization*, volume 1, pages 505–508. IEEE, 2009.

[122] Shabnam Bawa, Prashant Singh Rana, and RajKumar Tekchandani. Multivariate time series ensemble model for load prediction on hosts using anomaly detection techniques. *Cluster Computing*, pages 1–24, 2024.

[123] SP Usha Kirana and Demian Antony D'Mello. Energy-efficient enhanced particle swarm optimization for virtual machine consolidation in cloud environment. *International Journal of Information Technology*, 13(6):2153–2161, 2021.

[124] Luocheng Shen, Jiazhou Li, Yan Wu, Zhenyu Tang, and Yi Wang. Optimization of artificial bee colony algorithm based load balancing in smart grid cloud. In *2019 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*, pages 1131–1134.

IEEE, 2019.

[125] Suruchi Talwani and Jimmy Singla. Enhanced bee colony approach for reducing the energy consumption during vm migration in cloud computing environment. In *IOP Conference Series: Materials Science and Engineering*, volume 1022, page 012069. IOP Publishing, 2021.

[126] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[127] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1:7–18, 2010.

[128] Yashwant Singh Patel, Rishabh Jaiswal, and Rajiv Misra. Deep learning-based multivariate resource utilization prediction for hotspots and coldspots mitigation in green cloud data centers. *The Journal of Supercomputing*, 78(4):5806–5855, 2022.

[129] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, 2021.

[130] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. Utilization prediction aware vm consolidation approach for green cloud computing. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 381–388. IEEE, 2015.

[131] X Wang and X. Lu. A host-based anomaly detection framework using xgboost and lstm for iot devices. *Wireless Communications anhd Mobile Computing*, 2020:1–13, 2020.

[132] Abhijit Guha and Debabrata Samanta. Hybrid approach to document anomaly detection: an application to facilitate rpa in title insurance. *International Journal of Automation and Computing*, 18(1):55–72, 2021.

[133] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 1:59–63, 2012.

[134] Subrata Chakraborty. Topsis and modified topsis: A comparative analysis. *Decision Analytics Journal*, 2:100021, 2022.

[135] R Pushpalatha and B Ramesh. Amalgamation of neural network and genetic algorithm for efficient workload prediction in data center. In *Advances in VLSI, Signal Processing, Power Electronics, IoT, Communication and Embedded Systems: Select Proceedings of VSPICE 2020*, pages 69–84. Springer, 2021.