Energy Efficient Container Placement and Consolidation Technique in Cloud Data Center

A thesis submitted to the

UPES

For the Award of

Doctor of Philosophy

in

Computer Science & Engineering

By

Avita Katal

Under the Supervision of

Supervisors

Dr. Tanupriya Choudhury

Dr. Susheela Dahiya

School of Computer Science

UPES

Energy Acres, P.O. Bidholi via Prem Nagar, Dehradun, 248007: Uttarakhand, India

Energy Efficient Container Placement and Consolidation Technique in Cloud Data Center

A thesis submitted to the

UPES

For the Award of

Doctor of Philosophy

in

Computer Science & Engineering

By

Avita Katal

(SAP ID. 500057714)

Under the Supervision of

Internal Supervisor

Dr. Tanupriya Choudhury

Ex-Professor, UPES & Professor, Symbiosis International University

External Supervisor

Dr. Susheela Dahiya

Associate Professor, Graphic Era Hill University

School of Computer Science

UPES

Energy Acres, P.O. Bidholi via Prem Nagar, Dehradun, 248007: Uttarakhand, India

UPES
UNIVERSITY OF TOMORROW

Declaration

I, the undersigned, hereby state that the research work titled "**Energy Efficient Container Placement and Consolidation in Cloud Data Center**" has been conducted by me under the supervision of Dr. Tanupriya Choudhury, Ex-Professor, UPES, Dehradun & Professor, Symbiosis International University, Pune and Dr. Susheela Dahiya, Associate Professor, Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun, Uttarakhand. No part of this thesis has formed the basis for the award of any degree or fellowship previously.

Avita Katal

AVITA KATAL

School of Computer Science,

UPES,

Energy Acres, P.O. Bidholi, via Prem Nagar, Dehradun, 248007: Uttarakhand, India

UPES, DEHRADUN

INDIA

Certificate

I hereby certify that the thesis titled "**Energy Efficient Container Placement and Consolidation in Cloud Data Center**" authored by **Avita Katal (SAP ID: 500057714)**, a research scholar at the UPES, Dehradun, Uttarakhand, represents original research conducted under my supervision and guidance in partial fulfillment of the requirements for the Doctor of Philosophy degree in School of Computer Science. I further attest that this work has not been submitted for the attainment of any other diploma or degree from this university or any other institution.

Internal Supervisor

[illegible]

Dr. Tanupriya Choudhury

Professor

Symbiosis International University, Pune, Maharashtra, 412115.

Ex-Professor

School of Computer Science,

UPES, Dehradun, Uttarakhand, 248007

Dated: 13th December, 2023

Graphic Era
HILL UNIVERSITY
Established by Act 12 of 2011 of the State Legislature of Uttarakhand
Society Area, Clement Town, Dehradun
www.gehu.ac.in

GRAPHIC ERA HILL UNIVERSITY, DEHRADUN

(INDIA)

Certificate

I hereby certify that the thesis titled "**Energy Efficient Container Placement and Consolidation in Cloud Data Center**" authored by **Avita Katal (SAP ID: 500057714)**, a research scholar at the UPES, Dehradun, Uttarakhand, represents original research conducted under my supervision and guidance in partial fulfillment of the requirements for the Doctor of Philosophy degree in School of Computer Science. I further attest that this work has not been submitted for the attainment of any other diploma or degree from this university or any other institution.

External Supervisor

[illegible]

Dr. Susheela Dahiya

Associate Professor

Department of Computer Science and Engineering

Graphic Era Hill University, Dehradun.

Uttarakhand, 248002

Dated: 13th December, 2023

Abstract

Cloud Computing (CC) is a revolutionary computing model wherein consumers may dynamically expand or decrease computer resources based on their demands. Users only need to pay for the services they have really utilized. Cloud companies have lately provided Container as a Service (CaaS) as a new delivery model in addition to standard cloud services. The majority of current CC is based on Virtual Machine (VM) technology that is a cloud platform in the form of VM images with varying configurations and resources. Although VMs are often employed in the cloud, a new CC paradigm based on containers has progressively grown in past years as a versatile and economical approach of distributing energy-efficient resources. The advancement of Docker has greatly accelerated container-based virtualization. Reduced data center power usage is one of the primary difficulties that cloud providers confront. So far, current study has concentrated on energy-aware management of resources via the Infrastructure as a Service (IaaS) concept and VM aggregation. Containers, on the other hand, are gaining traction and will be a key installation mechanism in the cloud, notably in Platform as a Service (PaaS). This research work focusses on energy efficient container placement and consolidation in cloud data center. This research work focusses on identification and implementation of best-suited workload categorization technique followed by development of energy efficient techniques for containers placement and consolidation onto physical machines while maintaining Quality of Service (QoS) in cloud data centers (DC).

The first part of the research included clustering cloud workloads (Bits Brain dataset) of various types using two distinct clustering techniques (K means and Gaussian Mixture Model). The two techniques are compared in terms of Calinski-Harabasz Index, Davies-Bouldin Index and execution time. To further evaluate the efficiency of the approach used for workload management and resource utilization, the number of VMs used were calculated. According to the experimental results, there was a notable difference in the number of virtual machines (VMs) used in VM clusters created using K-means clustered workload data in comparison to the number of randomly sized VMs created. Following the clustering phase, classification was performed using several classification algorithms. Random Forest (RF), Logistic Regression (LR), K Nearest Neighbor (KNN), Support Vector Machine (SVM), Decision Tree (DT), Multi-Layer Perceptron (MLP), and Back Propagation Neural Network classification algorithms were compared. The classification of workloads is an important step in workload analysis. The accuracy of the models built is crucial for workload analysis, resource usage prediction and provisioning. The classification accuracy analysis aids us in determining which algorithm is best for a given data center workload. Among these techniques, the decision tree achieved the highest accuracy, reaching 99.18%.

The second part of the research focused on design and development efficient cloud container placement and consolidation algorithms. Optimal placement strategy for containers i.e. Discrete Firefly algorithm (DFF) and Discrete Firefly algorithm using Local Search Mechanism (DFFLSM) have been proposed. The proposed versions of firefly algorithm are compared with First Fit (FF), First Fit Decreasing (FFD), Random algorithm and Ant Colony algorithm (ACO). The comparison is done based on average energy consumption, average active VM, average active PM and average overall Service Level Agreement (SLA) violations in the cloud DC. The results show that DFFLSM performs better than all pre-existing container placement algorithms in terms of energy efficiency. It reduces average energy consumption of DC by 9.32% and 40.85% and average active PM by 18.30% and 21.89% in homogenous and heterogeneous environment respectively. Also, two optimized metaheuristic algorithms named Energy Efficient Ant Colony Optimization (EEACO) and Energy Efficient Firefly Optimization (EEFFO) that selects the hosts/PMs to consolidate containers are proposed to increase data center energy savings. The algorithms are compared with the pre-existing algorithms i.e. Random Host Selection Algorithm (RHS), First Fit Host Selection Algorithm (FFHS), Least Full Host Selection Algorithm (LFHS) and Correlation Threshold Host Selection Algorithm (CorHS) on the basis of energy efficiency, average active VMs, average active PMs, average container migrations and average overall SLA violations. The energy efficiency results show that proposed algorithms EEFFO and EEACO outperforms all preexisting host selection algorithms for container consolidation. However, EEFFO is better than EEACO. EEFFO reduces energy consumption of data center by 8.34% and 9.38% in homogenous and heterogenous environment respectively as compared to all other algorithms. It also reduces the average PM used by 16.35% and 11.65% in homogenous and heterogenous environment respectively as compared to the pre-existing algorithms and proposed algorithms. The SLA violations for EEFFO and EEACO algorithms is less in comparison to pre-existing ones, thus improving QoS also.

Acknowledgement

"Success is a tapestry woven with the threads of collective effort and unwavering support. It is a testament to the belief and contributions of those who accompany us on our journey, recognizing and nurturing our true potential. Together, we reach heights that surpass the bounds of individual accomplishment."

Embarking on the fulfilling path of a Ph.D. is a testament to the power of support and inspiration. As I near the end of this remarkable journey, I would like to extend my heartfelt appreciation to the exceptional individuals who have accompanied me through the ups and downs, providing unwavering encouragement and guidance. Their presence has been instrumental in shaping this memorable experience, and I am forever grateful for their inspiring influence.

I would like to take this opportunity to express my sincere gratitude to all those who have supported and contributed to the completion of my Ph.D. thesis. First, I would like to extend my heartfelt appreciation to my supervisors, Dr. Tanupriya Choudhury and Dr. Susheela Dahiya. Their exceptional guidance, unwavering support, and valuable insights have been invaluable throughout this research endeavor. I am truly grateful for their expertise, patience, and dedication in mentoring me and shaping the tra-

jectory of my research.

I would like to express my heartfelt gratitude to Dr. Sunil Rai, Chancellor, UPES, Dehradun, Uttarakhand for his unwavering support during my Ph.D. journey at UPES. His guidance, encouragement, and leadership has been invaluable to me, and I am deeply appreciative of the opportunities and resources he has provided.

I want to express my deep appreciation to Dr. Ram Sharma, Vice Chancellor, UPES, Dehradun, Uttarakhand for the invaluable support and guidance he has provided throughout my Ph.D. journey at UPES. Your exceptional leadership and unwavering dedication to uphold academic excellence have consistently inspired me during my research pursuits.

I would like to extend my heartfelt appreciation to Dr. Ravi S Iyer, Dean, School of Computer Science, UPES, Dehradun, Uttarakhand for his outstanding leadership and support during my academic journey at the School of Computer Science. His guidance and expertise have been instrumental in shaping my educational experience.

I am immensely grateful to my parents, Mr. Daleep Singh Katal and Mrs. Santosh Kumari for their unwavering support throughout my educational journey. Their love and encouragement have been the pillars that have guided me through every challenge and triumph. I am also indebted to my siblings, Deepaish Katal and Himali Kotwal and for their unwavering belief in my potential, their love, and their comforting words during times of uncertainty. Their prayers have been my source of strength and have propelled me forward on this remarkable path.

My heartfelt gratitude goes out to my students especially to Vitesh Sethi

for his invaluable support during my Ph.D. journey. Their relentless determination, infectious enthusiasm, and uplifting spirits have been a constant reminder of the resilience and camaraderie that can be found in our academic community.

I am immensely thankful to the members of my research committee, for their time, expertise, and valuable feedback. Their constructive criticism and insightful suggestions have significantly enhanced the quality and rigor of this work.

I would also like to express my deep appreciation to School of Computer Science, UPES for providing a conducive academic environment and access to resources that have facilitated my research journey.

Thanks to my friends and colleagues for their encouragement, support, and intellectual exchanges.

I am deeply grateful to all those who have played a part, big or small, in the completion of this Ph.D. thesis. Your support, guidance, and belief in my abilities have been invaluable. Thank you all for being an integral part of my academic journey and for making this achievement possible.

Avita Katal

Dehradun, India

13th December, 2023

Contents

Declaration ii

Certificate iii

Certificate iv

Abstract v

Acknowledgement ix

LIST OF FIGURES xxii

LIST OF TABLES xxvii

1 Introduction **1**

1.1 Introduction 1

1.1.1 Virtualization in Cloud 3

1.1.2 Container Perspective: Paradigm Shift 5

1.1.3 Energy Consumption in Data centre 7

1.1.4 Energy Efficient Techniques in Cloud Data Centers . 10

1.2 Research Problems and Objectives 11

1.2.1 Objectives 12

1.2.2 Sub Objectives 12

1.2.3 Research contribution 13

1.2.4 Thesis Organization 15

2 Background and Literature Review 18

2.1 Background and Literature Review 18

2.1.1 Hardware based techniques 20

2.1.2 Software based techniques 32

2.2 Summary 41

3 Workload Characterization and Categorization 42

3.1 Introduction 42

3.2 Literature Review 44

3.3 Methodology 54

3.3.1 Dataset Characteristics 54

3.3.2 Clustering Algorithms 55

3.3.3 Classification Algorithms 55

3.3.4 Process Flow 56

3.4 Experimental Setup 58

3.5 Results 59

3.6 Summary 70

4 Container Placement in Cloud Data Center 72

4.1 Introduction 72

4.2 Literature Review 74

4.3 Methodology 85

4.3.1 Problem formulation Container Placement in Cloud Data Center 85

4.3.2 Process Flow 88

4.3.3 Algorithms 89

4.4 Experimental Setup 102

4.4.1 Scenario 1: Homogenous Environment 103

4.4.2 Scenario 2: Heterogenous Environment 103

4.5 Results 104

4.5.1 Homogenous Environment 105

4.5.2 Heterogenous Environment 111

4.6 Summary 114

5 Container Consolidation in Cloud Data Center 117

5.1 Introduction 117

5.2 Literature Review 119

5.3 Methodology 129

5.3.1 Problem Formulation of Container Migration in Cloud Data Center 129

5.3.2 Process Flow 131

5.3.3 Algorithms 134

5.4 Experimental Setup 142

5.4.1 Scenario 1: Homogenous Environment 142

5.4.2 Scenario 2: Heterogenous Environment 144

5.5 Results 144

5.5.1 Homogenous Environment 146

5.5.2 Heterogenous Environment 155

5.6 Summary 165

6 Statistical Analysis 167

6.1 Statistical Analysis 167

6.1.1 Container Placement in Cloud Data Center 168

6.1.2 Container Consolidation in Cloud Data Center 171

6.2 Summary 182

7 Discussion **183**

7.1 Discussion 183

8 Conclusion and Future Work **191**

8.1 Conclusion and Future Work 191

8.2 Limitations and Future Work 193

9 References **195**

9.1 References 195

List of Publications i

Plagiarism Report ii

List of Figures

1.1 Comparison between traditional, hypervisor and container architecture 8

1.2 Data Center demand of global electricity 2010–2030 9

1.3 Breakdown of data center energy demand modeling at the hardware level based on taxonomies. 12

1.4 An overview of data center energy consumption modeling at the software level based on taxonomy. 13

3.1 Categories of Workload 44

3.2 Methodology for categorization and characterization for workload 58

3.3 Attribute score of different attributes 60

3.4 Value of K vs inertias on high significant attributes 60

3.5 K vs Inertia (CPU usage) 61

3.6 K vs Inertia (Memory usage) 61

3.7 K vs Inertia (Disk usage) 62

3.8 K vs Inertia (Network usage) 63

3.9 AIC and BIC plot for all attributes 65

3.10 AIC and BIC plot (CPU usage) 66

3.11 AIC and BIC plot (Memory usage) 67

3.12 AIC and BIC plot (Disk usage) 67

3.13 AIC and BIC plot (Network usage) 67

4.1 Methodology for container placement in cloud data center . 86

4.2 Flowchart of FFD Container Placement algorithm 89

4.3 Flowchart of Random Container Placement algorithm 92

4.4 Flowchart of FF Container Placement algorithm 93

4.5 Flowchart of ACO Container Placement algorithm 94

4.6 Flowchart of DFF Container Placement algorithm 96

4.7 Flow chart of DFFLSM Container Placement algorithm . . . 101

4.8 Comparison of average energy consumption in homogenous environment for different container placement algorithms . . 106

4.9 Comparison of average active VM in homogenous environment for different container placement algorithms 107

4.10 Comparison of average active PM in homogenous environment for different container placement algorithms 107

4.11 Comparison of average overall SLA violations in homogenous environment for different container placement algorithms . . 109

4.12 Impact of different container placement algorithms on data center (a) average energy consumption and (b) average active VM for different values of overbooking factor (RAM Threshold= 80 percent) in homogenous environment 110

4.13 Impact of different container placement algorithms on data center (a) average active PM and (b) average overall SLA violations for different values of overbooking factor (RAM Threshold= 80 percent) in homogenous environment 110

4.14 Comparison of average energy consumption in heterogenous environment for different container placement algorithms . . 111

4.15 Comparison of average active VM in heterogenous environment for different container placement algorithms 112

4.16 Comparison of average active PM in heterogenous environment for different container placement algorithms 114

4.17 Comparison of average overall SLA violations in heterogenous environment for different container placement algorithms 114

4.18 Impact of different container placement algorithms on data center (a) average energy consumption and (b) average active VM for different values of overbooking factor(RAM Threshold= 80) in heterogenous environment 115

4.19 Impact of different container placement algorithms on data center average active PM (a) and average overall SLA violations (b) for different values of overbooking factor (RAM Threshold= 80) in heterogenous environment 115

5.1 Host status module and its components in Cloudsim 4.0 (Piraghaj et al., 2015b) 132

5.2 Consolidation Module and its components in Cloudsim 4.0 (Piraghaj et al., 2015b) 133

5.3 Impact of host selection algorithms on average energy consumption for different OL combinations (a) MU and (b) MCor in homogenous environment 147

5.4 Impact of Host selection algorithms on average energy consumption for different UL combinations (a) MU and (b) MCor in homogenous environment 148

5.5 Impact of Host selection algorithms on average container migration for different OL combinations (a) MU and (b) MCor in homogenous environment 148

5.6 Impact of Host selection algorithms on average container migration for different UL combinations (a) MU and (b) MCor in homogenous environment 150

5.7 Impact of Host selection algorithms on average VMs used for different OL combinations (a) MU and (b) MCor in homogenous environment 150

5.8 Impact of Host selection algorithms on average VMs used for different UL combinations (a) MU and (b) MCor in homogenous environment 152

5.9 Impact of Host selection algorithms on average PMs used for different OL combinations (a) MU and (b) MCor in homogenous environment 152

5.10 Impact of host selection algorithms on average PMs used for different UL combinations (a) MU and (b) MCor in homogenous environment 153

5.11 Impact of host selection algorithms on average overall SLA violations for different OL combinations (a) MU and (b) MCor in homogenous environment 153
5.12 Impact of host selection algorithms on average overall SLA violations for different UL combinations (a) MU and (b) MCor in homogenous environment 154
5.13 Impact of host selection algorithms on (a) average energy consumption of data center and (b) average active VM for different values of overbooking factor in homogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm) 154
5.14 Impact of host selection algorithms on (a) average active PM and (b) average container migrations for different values of overbooking factor in homogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm) 155
5.15 Impact of host selection algorithms on average overall SLA violations for different values of overbooking factor in homogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm) 155
5.16 Impact of host selection algorithms on average energy consumption for different OL combinations (a) MU and (b) MCor in heterogenous environment 157
5.17 Impact of host selection algorithms on average energy consumption for different UL combinations (a) MU and (b) MCor in heterogenous environment 157

5.18 Impact of Host selection algorithms on average container migration for different OL combinations (a) MU and (b) MCor in heterogenous environment 158

5.19 Impact of Host selection algorithms on average container migration for different UL combinations (a) MU and (b) MCor in heterogenous environment 158

5.20 Impact of Host selection algorithms on average VMs used for different OL combinations (a) MU and (b) MCor in heterogenous environment 160

5.21 Impact of Host selection algorithms on average VMs used for different UL combinations (a) MU and (b) MCor in heterogenous environment 160

5.22 Impact of Host selection algorithms on average PMs used for different OL combinations (a) MU and (b) MCor in heterogenous environment 161

5.23 Impact of Host selection algorithms on average PMs used for different UL combinations (a) MU and (b) MCor in heterogenous environment 162

5.24 Impact of host selection algorithms on average overall SLA violation for different OL combinations (a) MU and (b) MCor in heterogenous environment 162

5.25 Impact of host selection algorithms on average overall SLA violation for different UL combinations (a) MU and (b) MCor in heterogenous environment 163

5.26 Impact of host selection algorithms on (a) average energy consumption of data center and (b) average active VM for different values of overbooking factor in heterogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm) . . . 163
5.27 Impact of host selection algorithms on (a) average active PM and (b) average container migrations for different values of overbooking factor in heterogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm) . . . 164
5.28 Impact of host selection algorithms on average overall SLA violations for different values of overbooking factor in heterogenous environment (UL=70 percent, OL=80 ercent and MU CS algorithm) . . . 164

List of Tables

1.1 Difference between Virtualization and Containerization . . . 8

3.1 Comparison of work done in the domain of workload characterization 48

3.2 Percentage of tasks in each cluster (K Means) 64

3.3 Percentage of tasks in each cluster (GMM) 68

3.4 Scores for performance evaluation parameters 69

3.5 Accuracy percentage of classification algorithms 70

4.1 Comparison of work done in the domain of container placement in cloud data centers 78

4.2 Objectives and Experiment sets for Container Placement . . 102

4.3 Parameter values used in algorithms (ACO, DFF and DF-FLSM) 103

4.4 PM configurations in homogenous environment for Container Placement 103

4.5 VM and container configurations in homogenous environment for Container Placement 104

4.6 PMs configurations in heterogenous environment for Container Placement 104

4.7 VMs and container configurations in heterogenous environment for Container Placement . . . 104

4.8 Percentage Reduction in average energy consumption, average active VM and average active PM of DFFLSM in comparison to other container placement algorithms in homogenous environment . . . 108

4.9 Percentage reduction in average energy consumption, average active VM and average active PM of DFF in comparison to other container placement algorithms in homogenous environment . . . 108

4.10 Percentage reduction in average energy consumption, average active VM and average active PM of DFFLSM in comparison to other container placement algorithms in heterogenous environment . . . 113

4.11 Percentage reduction in average energy consumption, average active VM and average active PM of DFF in comparison to other container placement algorithms in heterogenous environment . . . 113

5.1 Comparison of work done in the domain of container consolidation . . . 121

5.2 Parameters of EEACO . . . 136

5.3 Objectives and Experiment sets for Container Consolidation 143

5.4 PM configurations in homogenous environment for Container Consolidation . . . 144

5.5 VM and container configurations in homogenous environment for Container Consolidation 144
5.6 PMs configurations in heterogenous environment for Container Consolidation 144
5.7 VMs and container configurations in heterogenous environment for Container Consolidation 145
5.8 Comparison of EEFFO for average energy consumption reduction in Homogenous environment (in percent) 146
5.9 Comparison of EEACO for average energy consumption reduction in Homogenous environment (in percent) 147
5.10 Comparison of EEFFO average active VM reduction in homogenous environment 149
5.11 Comparison of EEACO average active VM reduction in homogenous environment 149
5.12 Comparison of EEFFO average active PMs reduction in Homogenous environment (in percentage) 151
5.13 Comparison of EEACO average active PMs reduction in Homogenous environment (in percentage) 151
5.14 Comparison of EEFFO average energy consumption reduction in Heterogenous environment (in percent) 156
5.15 Comparison of EEACO average energy consumption reduction in Heterogenous environment (in percent) 156
5.16 Comparison of EEFFO average active VM reduction in heterogenous environment 159

5.17 Comparison of EEACO average active VM reduction in heterogenous environment 159

5.18 Comparison of EEFFO average active PMs reduction in Heterogeneous environment (in percent) 159

5.19 Comparison of EEACO average active PMs in Heterogeneous environment (in percent) 161

6.1 Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous and heterogeneous data center for different container placement algorithms 169

6.2 Tukey multiple comparisons of significance values for energy consumption and SLA violations in homogenous and heterogeneous data center for different container placement algorithms 170

6.3 Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous data center for different container consolidation algorithms with MU as container selection algorithm 172

6.4 Tukey multiple comparisons of mean difference for energy consumption and SLA violations in data center for different container consolidation algorithms with MCor as container selection algorithm 173

6.5 Tukey multiple comparisons of significance values for energy consumption and SLA violations in data center for different container consolidation algorithms with MU as container selection algorithm 174

6.6 Tukey multiple comparisons of significance values for energy consumption and SLA violations in data center for different container consolidation algorithms with MCor as container selection algorithm 175

6.7 Tukey multiple comparisons of mean difference for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MU as container selection algorithm 178

6.8 Tukey multiple comparisons of mean difference for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MCor as container selection algorithm 179

6.9 Tukey multiple comparisons of significance values for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MU as container selection algorithm 180

6.10 Tukey multiple comparisons of significance values for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MCor as container selection algorithm 181

Chapter 1

Introduction

In this chapter, the concept of energy efficiency in cloud data centers is introduced, highlighting its significance in today's digital landscape. The chapter provides a roadmap for the entire thesis, outlining research objectives, discussing the contributions made by this study, and presenting the organizational structure of the thesis to guide the reader through the research journey.

1.1 Introduction

Cloud computing is the most innovative mechanism for delivering computer resources as a service. It signifies a transition away from computing as a bought product and approaching computation as a utility provided to clients via the Internet by large data centres. It has grown into a new computing paradigm that enables customers to have on demand access to computer services such as CPU/GPU, RAM, and disk storage. The cloud computing paradigm arose as a result of outsourcing and transformation operations, as well as the adoption of virtualization technology. Cloud resources were previously packed into several sorts of Virtual Machines (VMs) to serve cloud users. This indicates that cloud adoption has hit a tipping point, with cloud now hosting the vast majority of business workloads. To carry out their everyday activities, any organisation, whether a corporation or an educational institution, needs IT infrastructure (hardware and software) for computing and storage. Moreover, the cost of maintaining both hardware and software is always significantly higher than the cost of installing the same. Over time, it is possible that on-premises storage solutions may struggle to accommodate the continuous generation of data at high volumes. Even if they attempt to do things following the traditional strategy, they will need to spend considerably in infrastructure to achieve success. Most crucially, on-premises data warehouses may be incapable of supporting modern analytics solutions that require rapid data processing. As a result, there is a significant expense and stress load on the Information Technology (IT) staff accountable for administering an organization's IT infrastructure leading to many organizations/enterprises adopting a multi-cloud strategy. The amount of data and transaction volume stored in institutional data centres is continually expanding (Sahinaslan et al., 2022). Cloud computing signifies a substantial shift in the approach to design, create, deploy, expand, improve, monitor and manage IT solutions. The current state of computing presents a paradox: computers are increasingly becoming more efficient while the cost of computing per unit is rapidly declining, to the extent that computing power is now widely regarded as a utility(Lasica and Firestone, 2009) (Avram, 2014). Firms choose cloud computing and data storage over traditional computing and data storage approaches for a variety of reasons. Cloud migration, like any new technology, comes with risks and drawbacks, such as security issues. However, the benefits greatly exceed the disadvantages. Cloud computing offers various advantages to IT professionals, including portability, upgradeability, scalability, high availability, sharing resources, and, most crucially, cost effectiveness (Carroll et al., 2011). Because of these factors, operators and software providers are migrating their architecture to the cloud in order to gain greater flexibility while cutting costs. A corporation must examine many things before moving to the cloud. To start, the client must follow the current cloud system's technical criteria. Second, data transmission to the cloud must not break any national security legislation or jeopardise client data privacy. Third, the internal environment must allow for workload execution on the cloud.

1.1.1 Virtualization in Cloud

Virtualization allows a single physical system to host and manage several virtual servers, increasing resource efficiency and decreasing total power consumption (Barrett and Kipper, 2010; D. Huang and Wu, 2018; Jain and Choudhary, 2016; Shroff and Shroff, 2011). A rising number of enterprises are moving to virtualization to streamline workloads and make IT systems more adaptable and agile. Virtualization in computing refers to the establishing a virtual (rather than a real) replica of a resource or item, such as a server, OS, file server, or networking.
The roots of virtual memory extend to the late 1950s when the University of Manchester's Atlas system introduced automatic page replacement in a transistorized mainframe (Campbell and Jeronimo, 2006). Atlas automated the existing concept of paging, becoming the first functional prototype for virtual memory. The 1960s witnessed the emergence of the term "virtual machine" as IBM introduced the System/360 model 67, featuring virtual memory and a self-virtualizing processor instruction set (Campbell and Jeronimo, 2006). The accompanying CP-67 evolved into virtual machine (VM) operating systems, allowing concurrent execution of multiple operating systems on a single processor. IBM's mid-1960s M44/44X project explored time-sharing with virtual machines, laying the groundwork for widely used VM/timesharing systems like VM/370. Simultaneously, hardware virtualization took shape, enabling virtual machine monitors to run in isolated environments with minimal resource overhead. By the mid-1970s, virtualization gained widespread acceptance, addressing challenges like virtual storage and enhancing system capacity (Campbell and Jeronimo, 2006). The 1980s and 1990s saw a decline in hardware-level virtual machines with the rise of affordable minicomputers and personal computers (Campbell and Jeronimo, 2006). However, the era introduced a new category of virtual machines, exemplified by Sun Microsystems' Java Virtual Machine (JVM) and Microsoft's Common Language Runtime (CLR). These machines, emerging in the 1990s, operated in software, providing portability across different hardware platforms and marking a significant expansion of virtual machines into various domains, particularly in soft-

ware development (Campbell and Jeronimo, 2006).
The use of virtualization and virtual environments are essential concepts in cloud computing for data transfer. It helps both the guest user and the provider, while the former receives the components needed to complete the request; the latter is able to lodge several guests at no additional cost. Virtualization technology allows customers to easily abstract the physical features of computer resources and effectively multiplex underlying hardware resources. This service is provided by a hypervisor/Virtual Machine Monitor (VMM), which is a software layer that allows many guests OS to operate concurrently on a single physical host. Virtualization is required for the management of resources in a cloud-based environment since it multiplexes the resources. The first and most crucial advantage of virtualization is the isolation it provides. A flaw in one programme may cause other applications on the same system to fail. Most malware operates in this manner in order to crash other programmes, hack other programmes, and steal sensitive data. Virtualization in cloud computing improves security by safeguarding the integrity of cloud components as well as guest virtual machines. The second benefit of virtualization is improved performance. User-level sharing is possible in a classic multiuser system. In user-level sharing, different programmes compete for processing resources, memory, network bandwidth, and disk I/O. Cloud service providers also offers high use of pooled resources, resource sharing, and fast provisioning. Because VMs have exclusive access to resources, they offer stronger performance guarantees than shared infrastructure. Virtualized cloud component machines may also be scaled up or down on demand and offer dependability. Even in the case of application running within a virtual machine (VM), the resources of the host computer are utilized. It requires a high configuration host machine to efficiently run multiple virtual machines on a single host. Inadequate power can lead to performance instability problems. A virtual computer is less efficient in terms of hardware accessibility. It does not have direct access to the hardware. Its speed is insufficient for most IT businesses. Because of the challenges outlined above, the paradigm has shifted from virtualization to containerization. Containers are substantially lighter

than virtual machines and require much less time to boot up. This also allows more applications to be packed into a server—hundreds or thousands of containers may operate on a server, whereas just a couple dozen VMs can run on the same hardware.
Virtualization plays a crucial role in optimizing resource management within cloud computing environments, thereby improving overall operational efficiency. In the realm of cloud computing, it serves as a cornerstone for streamlined resource maintenance. This technology substantially enhances security by ensuring the integrity of both cloud components and guest virtual machines. Moreover, the dynamic scalability of virtualized machines associated with cloud components supports reliable and on-demand resource adjustments. In the context of cloud computing, the discourse on virtualization is extensive, with businesses leveraging a combination of servers for various applications. Virtualization becomes imperative in maximizing server utilization and efficiently servicing requests, ensuring that a significant portion of server time is dedicated to productive tasks. This approach provides transparency to applications running on virtualized environments, reducing maintenance costs and minimizing energy wastage. The reduction in the number of physical servers through virtualization directly translates into energy savings. In desktop virtualization, updates can be implemented swiftly, impacting multiple instances simultaneously. This multifaceted approach not only makes server maintenance more economical but also aligns with sustainability goals by reducing the environmental footprint associated with energy consumption and hardware disposal.

1.1.2 Container Perspective: Paradigm Shift

Cloud computing relies on virtualization technology, which partitions physical assets like servers into virtual components referred to as VMs. Certain extra degrees of abstraction in virtualization, on the other hand, has lowered work performance. Containerization (Sturm et al., 2017) is a novel virtualization technology paradigm that has received a lot of interest recently. Containerization allows applications to share their host OS kernel and only include the binaries and libraries required, resulting in a lower footprint than a VM. While cloud-based VMs make it simple to grow computing capacity on demand, clients must plan ahead of time since VMs take time to spin up. Waiting until demand rises can result in revenue loss due to unfulfilled demands, while acquiring too much in advance can result in wasted capacity and greater non-revenue producing expenses - exactly the situation that a cloud solution is designed to address. Containers are a lightweight solution that keeps all of the benefits of VMs, such as user and application separation in their own silos on a shared hardware platform, while increasing speed and server resource consumption for fluctuating loads. Containerization allows various virtual instances and resources to coexist on a single host operating system, sharing its associated libraries, drivers, and binaries. It, therefore, reduces the number of resources wasted during the computing process. Containers use the same host OS repeatedly, as opposed to installations. Containerization also makes it easier to distribute and manage apps and services across multiple infrastructures, such as edge/fog, cloud, and IoT. Containerization is revolutionizing how companies function due to its benefits in the collection and efficient use of resources, savings, mobility, reduced energy consumption, and speedy startup. While traditional VMs enhance the efficiency of physical servers, they come with significant cost and effort overhead. On the other hand, a container-based approach allows data center owners to distribute only the necessary code for an application's functionality, eliminating unnecessary dependencies. As a result, data center systems are utilized more efficiently. In typical VMs, it's the guest operating system, rather than the actual application, that consumes the majority of resources. The containers' smaller footprint offers several advantages across the data centre. Adopting a container-based approach results in several advantages, including reduced rack space requirements, lower energy consumption for cooling and electricity, decreased software licensing costs, and reduced maintenance efforts. Containers outshine other virtualization solutions when it comes to delivering high-quality services. Furthermore, due to their lower resource demands compared to virtual machines (VMs), more instances can be hosted and consolidated on a single server. This efficiency leads to decreased energy usage since fewer servers are necessary to support the same number of services. For instance, when configured with a maximum latency of 3000 ms, Docker, a container engine, can run up to 21 percent more services than KVM, a hypervisor. What's more, Docker achieves this level of performance while consuming 11.33 percent less energy compared to KVM, as documented by Cuadrado et al. [Cordero et al. in 2017]. Figure 1.1 illustrates the architectural contrast between traditional hypervisor-based virtualization and container-based virtualization, while Table 1.1 provides a comprehensive comparison between Virtualization and Containerization.

Mulahuwaish (Mulahuwaish, 2022) compared containerized microservice-based architectures with monolithic applications, using the "EasyTravel" application on AWS. The study included load generation, Dynatrace monitoring, and automated scaling.

Results indicate the microservice architecture outperformed the monolithic in key metrics:

- Throughput: Microservices handled nearly double the requests, outpacing the monolithic architecture.
- Response Time: Monolithic response time soared to almost 50 seconds under load, while microservices maintained around 300 ms on average.
- Errors: Monolithic architecture showed increasing errors, while microservices had brief periods of 4xx errors and minimal 5xx errors.

The findings underscore the scalability, reliability, and performance advantages of containerized microservices, emphasizing their efficiency in handling high loads, maintaining response times, and managing errors. The research provides a comprehensive methodology and implementation insights, supporting the adoption of containerized microservice architectures for modern datacenter deployment.

1.1.3 Energy Consumption in Data centre

The IT infrastructure of a business includes processing units, networking, and data storage. The current cloud service design is very centralised, which implies that several types of services may be run through a single

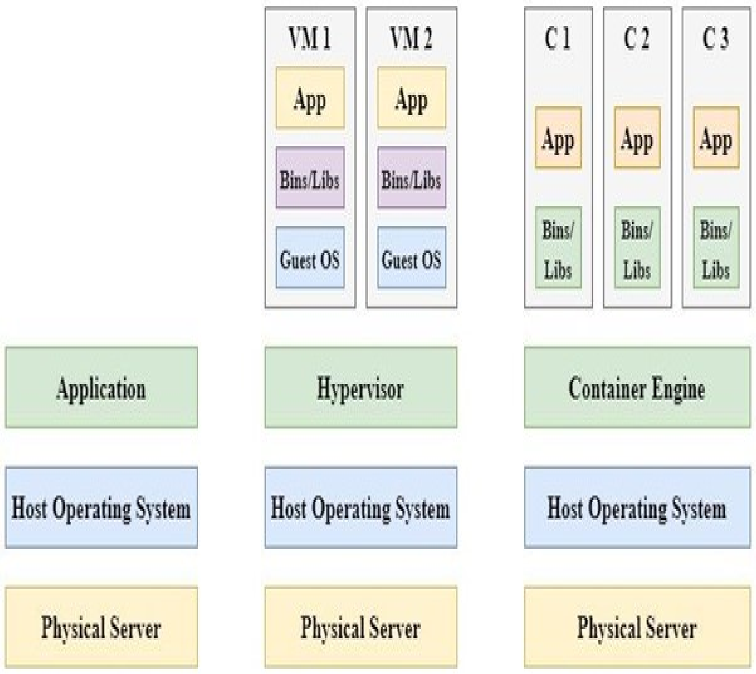


Figure 1.1: Comparison between traditional, hypervisor and container architecture

Table 1.1: Difference between Virtualization and Containerization

Parameters	**Virtual Machine**	**Container**
Guest Operating System	Hypervisor allows multiple and distinct OS to run on the same PM. A single VM is given a certain amount of RAM on which to run the Kernel.	All the guests share the same base OS and its Kernel. The image of the Kernel is loaded into the physical memory.
Security	Security of the VMs depends on how the hypervisor is implemented.	Container software like Docker have built-in security features that can be leveraged.
Performance	The VMs have a small overhead when compared to containers as the translation of machine instructions occurs from the host OS to guest OS.	There is little to no overhead in using containers as the applications are executed in the base OS itself.
Isolation	Hypervisor isolates each VM from host OS as well as from other VMs. This means files, libraries etc. cannot be shared between guests and the host.	Each container has its own set of file systems that can be shared between other applications.
Startup time	VM takes sufficient time to boot.	Containers take less time to boot as compared to VMs.
Storage	VMs take ample storage as the whole Kernel and the secondary programs associated with the OS need to be installed.	Since the base OS is shared, containers require less storage.

Data Centre (DC). Because of the exponential development in data gathering and consumption, the demand for data centres is increasing. Cloud computing rapid advancement in turn increases the data centers. The demand for data centre energy will rise from 200 TWh in 2016 to 2967 TWh in 2030 (Andrae, 2019). Figure 1.2 shows the DC demand of global electricity 2010–2030 (Andrae and Edler, 2015).

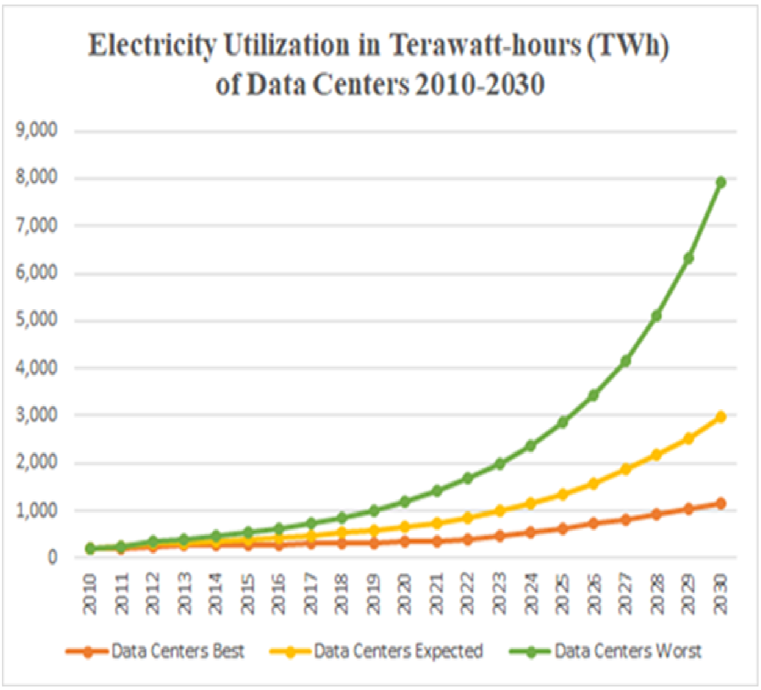


Figure 1.2: Data Center demand of global electricity 2010–2030

Cloud computing is heavily reliant on a multitude of data center servers that are necessary to provide services to a vast clientele on a pay-per-use basis. However, these servers consume a significant amount of energy and produce heat that requires cooling measures to ensure their optimal functioning. The physical space occupied by these resources, along with the networking devices, cooling systems, displays, and server farms, among other components, demands substantial electricity. As of now, cloud data center power consumption accounts for approximately 1.1 to 1.5 percent of global electricity usage, and this figure is expected to rise considerably in the future (How Much Energy Do Data Centers Really Use? - Energy Innovation: Policy and Technology, n.d.). As the number of Internet of Things (IoT) based applications rises, the utilisation of cloud services grows rapidly, increasing cloud DCs' power consumption by 20 percent to

25 percent every year. Existing research estimates that data centres produce 78.7 million metric tonnes of CO_2, accounting for 2 percent of world emissions (Gill and Buyya, 2018). These concerning numbers need a reconsideration of infrastructure's energy efficiency. Making resources green using green technology has therefore become a top priority for a number of government and business organisations. Green IT, from an environmental standpoint and to solve IT-related environmental problems, offers a diverse set of strategies and practises through a variety of green initiatives. Energy efficiency is one of the simplest and most efficient ways to save money, reduce greenhouse gas pollutants, create jobs, and satisfy growing power requirements. Energy usage in cloud DCs is inefficient owing to underloading and overloading of infrastructure resources. Energy is utilised mostly while certain resources are idle, raising the cost of cloud services. Furthermore, data centres contribute to a number of environmental challenges related to CO_2 emissions. As a result, cloud DCs must provide low-carbon cloud services that generate less heat in the form of greenhouse gas emissions.

1.1.4 Energy Efficient Techniques in Cloud Data Centers

Power saving has recently been a top priority for data centres. According to survey, an average data centre consumes the same amount of power as 25,000 homes (Dayarathna et al., 2016). Clearly, the emergence of innovative solutions suitable of efficiently reducing energy usage in data centres is required for the future advancement of cloud computing. Many cloud providers have promised to attain carbon neutrality and are researching innovative methods to make cloud DCs and cloud-based services more ecologically friendly. Most of the current cloud computing infrastructure relies on VM architecture, where cloud providers offer VM images with different specifications and resource levels. While VMs are commonly utilized in the cloud, a novel approach to cloud computing based on containers has rapidly gained prominence in recent years. This container-based methodology offers a flexible and efficient solution for the distribution of resources with a focus on energy efficiency. The power use of data centers is determined by a variety of factors, including hardware attributes, workload, networking infrastructure, application kinds, storage architecture, and many more. By utilizing cutting-edge technologies like virtualization, resource management, infrastructure auto-scaling, energy-aware scheduling, storage space management, live migration of virtual machines, carbon footprint reduction, and others, cloud computing can significantly enhance energy efficiency. These techniques can be implemented at various levels of components and can be integrated to create a comprehensive system referred to as a green data center. A green data center not only addresses environmental concerns by reducing energy consumption but also delivers financial benefits by enhancing cost-effectiveness in cloud computing operations. Every computer system comprises two fundamental components, namely hardware and software. Similarly, a data center has two primary elements: hardware and software, and both can be optimized to minimize energy consumption. The software layer can be subdivided into three additional levels: the operating system (OS) layer, the virtualization layer, and the application layer. Energy usage in the data centre may be considerably lowered by enhancing or altering these layers. Figure 1.3 and figure 1.4 shows a taxonomical strategy for modelling data center energy consumption at hardware and software level. To realize the objectives of green cloud computing, it is essential to implement various techniques at the individual program level. Apart from optimizing the software and hardware components, external factors like the utilization of renewable energy sources, adoption of sustainable organizational initiatives, and compliance with government regulations are crucial in achieving the goals of green cloud computing.

1.2 Research Problems and Objectives

Containerization technology introduced lately as a support of Virtualization has led to better optimization of the resources. Still optimal placement of containers onto the physical machines and server consolidation remains an active research challenge where abstract level details of the resources considered are not enough. Component level details of the resources should be considered for offering better solutions.

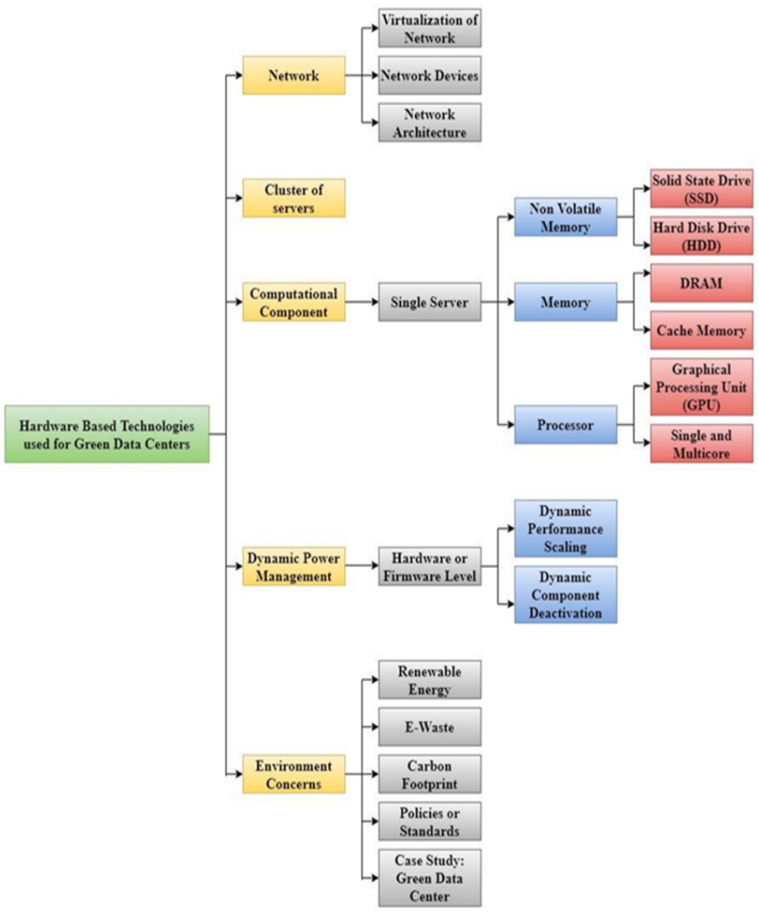


Figure 1.3: Breakdown of data center energy demand modeling at the hardware level based on taxonomies.

1.2.1 Objectives

Design and development of an energy efficient multicore-aware container placement and container consolidation technique in cloud data centre that incorporates all component level factors such as multiple cores, memory, storage, and network while placing the containers on physical machines as well as container migration to minimize energy usage within the cloud data centre.

1.2.2 Sub Objectives

1. Identify and implement best-suited workload categorization technique.
2. Develop energy efficient techniques for placement of containers onto physical machines considering CPU multi cores, memory, storage and network together.
3. Container migration strategies are being developed in order to min-

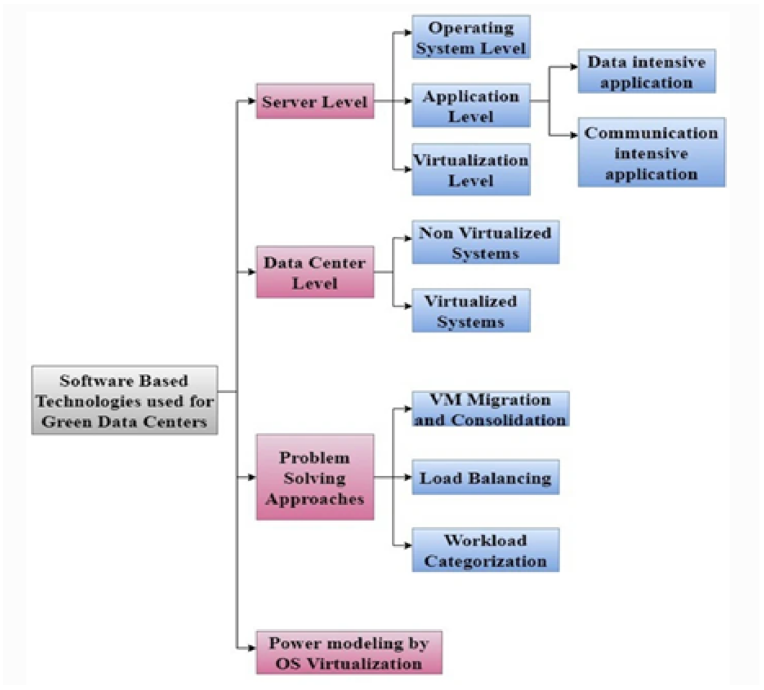

Software Based Technologies used for Green Data Centers
Server Level
Operating System Level
Application Level
Virtualization Level
Data intensive application
Communication intensive application
Data Center Level
Non Virtualized Systems
Virtualized Systems
Problem Solving Approaches
VM Migration and Consolidation
Load Balancing
Workload Categorization
Power modeling by OS Virtualization

Figure 1.4: An overview of data center energy consumption modeling at the software level based on taxonomy.

imize energy usage while preserving the requisite Quality of Service (QoS).

1.2.3 Research contribution

This research work focuses on identification and implementation of best-suited workload categorization technique followed by development of energy efficient techniques for containers placement and consolidation onto physical machines while maintaining QoS in cloud data centers (DC). The research contribution of this work is discussed as below:
Workload characterization and categorization using the Bit Brains Trace workload traces:

- Analyzed the BBT traces to gain insights into different types of workloads in cloud data centers.
- Characterized and categorized workloads by examining their resource usage patterns (CPU usage, memory requirements, I/O operations, and network traffic patterns).
- The workload characterization and categorization data has been used to determine the most suitable VM size for each workload based on

its resource requirements and performance expectations.

- Implemented and compared different classification techniques for BBT traces to find the most suitable model that can be used by researchers help to build a prediction model for future workload.

Implementation of a proposed metaheuristic container placement algorithm in CloudSim 4.0 for both homogenous and heterogenous environments.

- Designed and implemented container placement algorithms that optimize energy consumption by leveraging metaheuristic techniques.
- Integrated the algorithm into CloudSim 4.0 environment, a popular cloud simulation framework, to evaluate its performance and energy efficiency in a realistic environment and compare the proposed algorithms with pre-existing algorithms.

Implementation of a proposed metaheuristic container consolidation algorithm in CloudSim 4.0 for both homogenous and heterogenous environments.

- Designed and implemented container consolidation algorithms in cloud data centers utilizing metaheuristic approaches that dynamically reorganizes containers across VMs and PMs aiming to minimize energy consumption and cause least SLA violations.
- Integrated the algorithms within the CloudSim 4.0 framework to evaluate its effectiveness in achieving energy-efficient container consolidation.

These detailed research objectives aim to provide a comprehensive understanding of workloads in cloud data centers, optimize VM sizing decisions, and develop innovative container placement and consolidation algorithms. By addressing these objectives, the study aims to contribute to the body of knowledge on energy-efficient resource management in cloud computing, leading to improved sustainability and cost-effectiveness in data center operations.

1.2.4 Thesis Organization

The thesis organization for the study on energy-efficient container placement and consolidation in cloud data centers is structured as follows:

Chapter 1: Introduction

- Provides an introduction to cloud computing, virtualization, paradigm shift to containerization and why energy efficiency is required in cloud data centers.
- Presents an outline of the study subject, aims, and research questions.
- Outlines the significance and relevance of energy-efficient container placement and consolidation in cloud data centers.
- Presents the structure and organization of the thesis.

Chapter 2: Background and Literature Review

- Provides a comprehensive review of the background and related literature on energy efficiency in data centers.
- Discusses the existing approaches, techniques, and algorithms for resource management, workload characterization, container placement, and consolidation.
- Highlights the gaps and limitations in the current research and identifies the research opportunities in the field.

Chapter 3: Workload Characterization and Categorization

- Introduces the methodology used for workload categorization and characterization.
- Describes the BBT dataset and its relevance to the research objectives.
- Presents the characterization and categorization techniques applied to the workloads in the dataset.
- Analyzes the findings and provides insights into the different types of workloads and their resource requirements.

Chapter 4: Container Placement in Cloud Data Centers

- Explores the theoretical foundation and principles of container placement in cloud data centers.
- Discusses the proposed metaheuristic container placement algorithms.
- Describes the implementation details and integration of the algorithms into the CloudSim 4.0 simulation framework.
- Evaluates the algorithm's performance in terms of average energy consumption, average active VMs used, average active PMs used and average overall SLA violations through simulation experiments.

Chapter 5: Container Consolidation in Cloud Data Centers

- Discusses the theoretical aspects and challenges of container consolidation in cloud data centers.
- Presents the proposed metaheuristic container consolidation algorithms.
- Describes the implementation details and integration of the algorithm into the CloudSim 4.0 simulation framework.
- Conducts simulation experiments to evaluate the algorithm's efficiency in achieving energy efficient resource consolidation, average active VMs used, average active PMs used and average overall SLA violations.

Chapter 6: Statistical Analysis

- Discusses the theoretical aspects of statistical analysis.
- Presents the results obtained by the applying tukey HSD test to the output of container placement and consolidation algorithms.

Chapter 7: Discussion

- Discusses the theoretical aspects of container placement and consolidation algorithms.
- Discusses the results of container placement and consolidation algorithms.

Chapter 8: Conclusion and Future Work

- Summarizes the main findings, contributions, and implications of the research.
- Describes the study's weaknesses and identifies topics for future investigation and improvement.
- Concludes the thesis with a reflection on the significance of energy-efficient container placement and consolidation in cloud data centers and its potential impact on the sustainability and efficiency of cloud computing environments.

The organization of the thesis follows a logical flow, starting with the background and literature review, progressing through the different research aspects (workload characterization and categorization, container placement, and container consolidation), and concluding with the evaluation of the proposed algorithms and future research directions.

Chapter 2

Background and Literature Review

In the realm of cloud data centers, achieving optimal energy efficiency is paramount. This literature review chapter embarks on a comprehensive exploration of the multifaceted approaches adopted to enhance energy efficiency. It traverses the landscape of hardware and software techniques, providing a nuanced understanding of the myriad strategies employed to minimize energy consumption in the dynamic and ever-evolving domain of cloud data centers.

2.1 Background and Literature Review

Data centers are crucial infrastructure with high energy consumption that deliver Internet-based services on a large scale. Power utilisation models are critical in developing and upgrading energy-efficient operations in data centres with the goal of reducing excessive energy use. In recent years, the importance of energy efficiency in data centers has significantly grown and has become more complex. To ensure uninterrupted data availability, every component of the data center design must effectively perform its designated tasks to minimize downtime and necessitate appropriate energy support. The infrastructure, including power supply, technical cooling, and technical security, forms the foundation of all information technology (IT) infrastructures. Any disruption in the physical infrastructure, no matter how small, has a significant impact on the functionality of IT services. The key attributes of a green data center lies in its energy efficiency and low global environmental impact. A green or sustainable data center is a facility for storing, managing, and distributing data, where all systems, including the hardware and software components, are optimized for energy efficiency. This approach leads to reduced carbon emissions, cost savings, and improved operational efficiency. These eco-friendly data centers contribute to the efforts of modern enterprises in conserving power and minimizing their carbon footprint.

The literature review is structured into two primary categories: Hardware-based techniques and Software-based techniques. 1. Hardware-based techniques: This category encompasses methods and approaches that operate at the physical hardware level of a data center. This includes considerations and optimizations related to various hardware components such as CPUs (Central Processing Units), GPUs (Graphics Processing Units), memory (RAM), storage devices, and networking infrastructure. Techniques falling under this category are concerned with how to enhance the performance, efficiency, and utilization of these physical resources within a data center.

2. Software-based techniques: In contrast, this category comprises methods that operate at higher levels of abstraction within the data center environment. These techniques are applied in the realm of software, virtualization, and operating systems. They focus on optimizing the utilization and management of resources in a more abstract or virtualized manner. Examples include strategies for efficient virtual machine management, containerization, orchestration, and operating system-level optimizations.

Hardware and software form a symbiotic relationship crucial for energy-efficient systems. Focusing solely on one aspect may limit efficiency, as demonstrated by well-optimized software hindered by inefficient hardware, and vice versa. Researchers advocate a comprehensive approach, considering both software-based techniques, like workload scheduling, and hardware-based strategies for infrastructure enhancement. This combined optimization reduces costs, energy consumption, and environmental impact, aligning with sustainability goals. In the ever-evolving landscape of cloud computing, acknowledging advancements in both hardware and

software ensures a holistic understanding of energy-efficient practices, imperative for compliance with regulatory energy efficiency guidelines for data centers. In conclusion, acknowledging both hardware and software-based techniques in the context of reducing energy demand in cloud data centers is essential for achieving a well-rounded, effective, and sustainable approach to energy efficiency. It allows for a more nuanced understanding of the challenges and opportunities in this complex field.
In essence, hardware-based techniques deal with the physical components of the data center, while software-based techniques address the software and virtualization layers that interact with and control these hardware resources. Each category of techniques offers different ways to improve the performance, efficiency, and overall effectiveness of data center operations.

2.1.1 Hardware based techniques

2.1.1.1 Dynamic Voltage and Frequency Scaling of CPU

In the current market, there are several technologies available for dynamically adjusting CPU frequency and voltage based on workload. In their work, Kim et al. [Kim et al., 2011] incorporated the Dynamic Voltage and Frequency Scaling (DVFS) capability of the CPU into their scheduling algorithm. This scheduling approach took into account the deadlines of Bag-of-Tasks applications as constraints and dynamically adjusted the CPU frequency to ensure the timely completion of sub-tasks. A Bag-of-Tasks application typically comprises a set of independent and identical tasks.

Similarly, Pietri et al. [Pietri et al., 2014] also introduced an algorithm for scheduling that capitalizes on the DVFS capabilities of the CPU. Their algorithm's major goal was to fine-tune the CPU rate in order to minimize overall power usage while still achieving user-specified task completion constraints. The algorithms aimed to decrease overall energy consumption, as DVFS was not always inherently energy-efficient due to the potential increase in execution time when scaling the CPU frequency. This could result in increased idle time for processors. As a result, the suggested method only adjusted the frequency if it could reduce total energy use.

A multi-core processor consists of two or more cores that execute and interpret program instructions. This enhances the overall program speed and enables parallel computing, making multi-core processors suitable for CPU-intensive tasks. Operating systems designed for multi-core systems can handle multiple processes and threads, which leads to fundamental concerns such as resource management and performance. The authors of (Attia et al., 2017) conducted a study on multi-core processors, focusing on power management issues in multi-core architectures. Their proposed technique involves using clustered Dynamic Voltage and Frequency Scaling (DVFS) in asymmetric multi-core processors. This technique utilized scoring to schedule critical section threads. An adaptive controlled strategy controls the proposed technique, making decisions on how to efficiently use the technique based on various parameters such as workload style and current core usage. Dinakarrao [Dinakarrao, 2021] introduced a self-aware power management scheme tailored for multi-core microprocessors. To implement DVFS, the power management unit utilized a linear predictor to anticipate the workload.

Dorronsoro et al. [Dorronsoro et al., 2014] described a multi-level hierarchical strategy for planning large workloads of parallel computations on multi-core systems with distributed processing. The primary objective was to concurrently minimize computation time and energy utilization. The authors implemented this technique by combining schedulers at the distributed system and data center levels. Johari and Kumar [Johari and Kumar, 2015] focused on the performance aspect of multi-core systems, specifically task migration and load balancing. They proposed four algorithms: random search, sequential search, average method, and random search with spatial locality. The average method was a load-balancing algorithm implemented at the core level, considering the global and local view of the cores and using a threshold to prevent core overutilization. However, this load balancing algorithm was not suitable for dynamic workloads, and the threshold formula was not robust. Task migration had overhead due to state transfer and cache locality issues when data needed to be loaded again into the cache of the new core. Mann [Mann, 2016] addressed the task migration problem by pinning virtual machine (VM) CPU cores to specific physical machine (PM) CPU cores. A cost problem was formulated, and constraint programming was used to find a solution. The first stage involved VM-to-PM mapping to minimize the cost function, calculating the number of overloaded CPUs, and reducing the count of active servers by allocating VMs from lightly used and overloaded PMs using the Modified Best First Decreasing (MBFD) Heuristic. In the second stage, optimal core mapping was performed using exhaustive backtrack search and first fail heuristic. However, the search space could be further reduced using advanced heuristics. Additionally, the author did not consider memory resources when scheduling VMs. To efficiently utilize memory resources, modern operating systems employ Non-Uniform Memory Access (NUMA) hardware, which automatically optimizes application performance on the hardware. In NUMA, software threads of an application are scheduled on cores where the corresponding memory contents reside, resulting in faster memory access. It is also beneficial for a process being swapped-in to run on a previously used core to take advantage of the cache contents. In NUMA, task migration should be infrequent, and associated memory contents should remain on the same core. With the introduction of Kernel Virtual Machine (KVM), the traditional Linux Kernel has been transformed into a hypervisor that schedules each virtual machine as a running process. However, the memory requirements of these virtual machines are not taken into account. This can lead to performance degradation on NUMA hardware, particularly with long-running tasks like virtual machines. Zijlstra [Zijlstra, 2011] proposed a dynamic NUMA binding approach to address this issue. The kernel ensured that the process of the virtual machine ran on the CPU of the corresponding NUMA node and properly allocates memory resources. Each process is held by the scheduler within the home node until load balancing begins. Arcangeli [Arcangeli, 2012] presented AutoNUMA, a resource management approach that focused on memory access by processes. It suggested that the kernel should monitor the memory access of each process and migrate frequently accessed pages without the application being aware of the memory migration. However, due to the reliance on extensive per-process metadata, scalability issues might arise in very large memory systems. Chen et al [Chen et al., 2018] proposed an innovative approach for DVFS called "lightweight learning-directed DVFS." This approach leveraged counter propagation networks to sense, classify, and predict task behavior, ultimately determining the optimal voltage and frequency settings for the system. Moreover, the authors provided an efficient method for assessing performance requirements for users. The study's results illustrated that the learning-directed DVFS approach can accurately forecast the optimal CPU frequency with remarkable precision, achieving efficiency gains of up to 42 percent.

2.1.1.2 Energy efficiency in GPU

A graphical processing unit (GPU) is a fast computer chip designed primarily for image rendering. However, its parallel processing architecture has made it suitable for High-Performance Computing (HPC) in cloud and server infrastructures. Companies like Nvidia, Intel, and ATI are developing more general-purpose components for GPUs, allowing software developers to utilize the extra processing power on video cards for non-graphic operations. The availability of general-purpose programming languages and Application Programming Interfaces (APIs) like RapidMind and Brook has made it easier for software developers to target GPUs. Nevertheless, delivering HPC through the cloud remains challenging. Even with hardware-level virtualization for GPUs, latency becomes a significant concern as the kernel of an application may not fully utilize the GPU's capabilities. This necessitates scheduling algorithms that can enhance the co-execution of multiple applications and maximize GPU utilization. Ukidave et al [Ukidave et al., 2016] introduced Mystic, a framework that scheduled workloads on GPUs while minimizing interference between applications. Before execution, a profile was created based on six compute resources that applications contended for, including stream multi-processors, memory resources, and the interconnect network. Mystic used collaborative filtering techniques based on Single-Value Decomposition to measure and predict these resources. Applications were scheduled on an ideal GPU if available, or dispatched to the GPU with the lowest similarity score compared to other applications. Mystic improved throughput, GPU utilization, and maintained Quality of Service (QoS). However, the prediction of application characteristics based on previous executions before the current application's execution might have lead to false predictions.

Xu et al [Xu et al., 2017 presented GScheduler, which considered function calls and resource usage during application execution to reduce and detect interference between co-executing applications. The similarity score for key function calls impacting GPU resource utilization was obtained using a GPU resource usage vector and graph. GScheduler outperformed Least Loaded and Round Robin schedulers. However, creating a profile for each application could potentially cause resource exhaustion. The authors did not compare GScheduler with other machine learning frameworks like Mystic, so its efficiency relative to those frameworks was unknown.

Jararweh et al. [Jararweh et al., 2012] proposed a comprehensive power and performance control architecture that reduced energy usage in GPU-based clusters while maintaining system performance at an appropriate level. The system actively adjusted the GPU cluster to respond to variations in workload demands and enhanced GPU device inactivity.

Tang et al [Tang et al., 2019] explored the impact of GPU DVFS on power usage and performance in deep learning. Their tests covered various GPU models, DVFS settings, and Deep Neural Network (DNN) setups. They found that different convolutional algorithms (Winograd, FFT, and GEMM) exhibited varying degrees of power conservation with GPU DVFS. Optimizing the core frequency could lead to energy savings, amounting to 14.5% in GEMM, 12.6% in FFT, and 15.8% in Winograd. These findings highlighted that finding the ideal core frequency not only enhances DNN performance by as much as 33% but also resulted in substantial reductions in energy consumption during training and inference. This underscores the potential of GPU DVFS for energy-efficient DNN processing while maintaining minimal impact on performance.

2.1.1.3 Dynamic Voltage and Frequency Scaling of Memory

Memory in servers and computing systems is a physical device that stores information in the form of bits. The cost of memory decreases as its size increases, but this comes at the expense of speed. To ensure maximum CPU utilization, memory needs to respond to data requests at the same speed as the processor executes instructions. This led to the classification of memory as volatile and non-volatile. Volatile memory, such as SRAM and DRAM, can respond faster to CPU requests but loses its contents when power is lost. On the other hand, non-volatile memory, often referred to as storage, retains data even when power is off. This section focuses on volatile memory (referred to as memory), while non-volatile memory (storage) is discussed later.

In their study, David et al. [David et al., 2011] introduced an approach that employs Memory DVFS to dynamically adjust memory frequencies based on workload, effectively minimizing energy consumption. They also introduced a power model that quantified the power dependency on frequency and demonstrated substantial power reductions through the implementation of memory DVFS. Additionally, they proposed a control algorithm aimed at fine-tuning the memory frequency and voltage. This approach was assessed on real hardware using SPEC CPU2006 workloads and holded the potential for extension to various workload types and DVFS applications, applicable to both memory and CPU components.

Deng et al. [Deng et al., 2018] proposed an approach involving active lower-power modes, termed "MemScale," designed to enhance energy proportionality in the main memory. The primary goal was to enhance the energy efficiency of the memory subsystem, which could account for up to 40% of the overall system's energy consumption. MemScale achieved a performance degradation of less than 10%.

SRAM, composed of flip-flops, stores each bit using transistors but loses data when power is lost. Its power consumption depends on access frequency and can be comparable to DRAM at high frequencies. SRAM also suffers from power leakage. To overcome these challenges, Fujita et al (Fujita et al., 2017) suggested using embedded spin torque transfer MRAM (e-STT-MRAM), which offered high write access speed and endurance due to its technology's smaller Magnetic Tunnel Junction (MTJ) size. In cloud servers, accessing main memory is a known bottleneck, and using short retention MTJs in e-STT-MRAM-based Last Level Cache (LLC) can significantly increase operation speed and reduce power consumption.

Optimizing virtual machine memory sizes can enhance application performance. Sakamoto et al. [Sakamoto et al., 2016] proposed an optimized memory ballooning technique, a memory reclamation technique used by the hypervisor to reclaim unused memory from virtual machines. The technique dynamically adjusted the invocation time based on cache hit ratio, resulting in increased throughput and avoidance of low-performance virtual machines.

Chen et al. [Chen et al., 2017] addressed latency reduction and throughput improvement for read operations by minimizing distortion and bias caused by Least Frequently Used (LFU) and Least Recently Used (LRU) algorithms. They proposed a greedy approach that combined LRU and LFU, where the data cache called the greedy algorithm to free a portion of memory when reaching a predefined limit. The approach reduced latency and increased throughput, although the improvement for writing operations was minimal. In a related work, Blankstein et al. [Blankstein et al., 2017] proposed hyperbolic caching for web applications, emphasizing the efficient use of eviction data structures.

To handle the dynamic nature of workloads in cloud computing, Yang et al. [Yang et al., 2015] modified the architecture of existing CPU/FPGA acceleration systems. They created a CPU-Cache-FPGA (Filed Programmable Gate Array) architecture that utilized its own data cache to minimize data fetch latency. Bus snooping logic is employed to further reduce data access latency. Simulations demonstrated a performance improvement of up to 2.6 times.

Bazzi et al. [Bazzi et al., 2018] provided a comprehensive overview of Resistive RAM-based Non-Volatile Static Random Access Memory (NVSRAM) architectures, focusing on data storage and restoration. Traditional DRAM is a non-volatile, read-write memory that stores data using capacitors. The charged and discharged states of the capacitor represent the binary bits 1 and 0, respectively. However, DRAM requires constant refreshing to prevent data loss, which consumes significant energy and limits

its capacity. To address these issues, non-volatile memory is being incorporated into various aspects of computing.

Venkatesan et al. [Venkatesan et al., 2015] proposed a 3-level cache miss model based on Miss Ratio Curves (MRC). It considered both system and user perspectives to allocate cache resources appropriately, with DRAM at level 1, non-volatile memory at level 2, and disk/magnetic tapes at level 3. Different types of non-volatile memory, such as Phase Change Memory (PCM), Ferromagnetic RAM (FRAM), and Magnetic RAM (MRAM), have emerged. PCM detects resistance changes between amorphous and crystalline states to represent data. The Multi-Level Cell (MLC) architecture in PCM allows for multiple states, but it has drawbacks in terms of performance, endurance, and power consumption. Single-Level Cell (SLC) PCM, with one bit per cell, offers lower power consumption, higher performance, and longer lifespan. Qiu et al. [Qiu et al., 2015] proposed a task scheduling algorithm and MLC/SLC PCM mode configuration using morphable PCM cells. The algorithm optimized the allocation of tasks to PCM cells based on a fitness function, resulting in improved execution time compared to a heuristic approach. Endurance and write performance are critical for PCM, so [Wang et al., 2016] suggested a hybrid architecture that combined PCM and DRAM. DRAM is divided into Metadata cache, which stores metadata in PCM, and data cache. When a DRAM cache miss occurs, a request is sent to PCM, and data deduplication is performed during read/write transactions, leading to significant data reduction. Another approach proposed in [He et al., 2018] is a tri-regional hybrid cache utilized both Spin-Transfer Torque RAM (STT-RAM) and DRAM. The hybrid cache consists of a Non-volatile STT-RAM (NSR) region, a DRAM area, and a Volatile STT-RAM (VSR) region. The asymmetric data access approach optimizes energy consumption by utilizing different cache regions for static and dynamic energy reduction.

Overall, these studies explored various architectural designs and techniques to enhance the performance, energy efficiency, and capacity of non-volatile memory technologies in the context of data storage, cache management, and task scheduling.

2.1.1.4 Coordinated CPU and Memory DVFS

Deng et al. [Deng et al., 2012] introduced CoScale, a novel approach that applied DVFS to both the memory and CPU subsystems to minimize total power consumption in the system. Unlike previous works, CoScale considered performance constraints and coordinates DVFS on both memory and CPU. The objective was to maximize energy savings for the entire system.

As noted by Dhimsan et al. [Dhimsan et al., 2015], reducing the frequency can sometimes lead to increased energy consumption. Therefore, CoScale ensured a balance between the power utilization of the system and its components. Efficiently explored the available CPU and memory frequency settings and adjusting component voltage accordingly is a complex task for the CoScale algorithm. This challenge arises because the algorithm must consider a vast number of possibilities, which are determined by the variables m, n and c. Here, m represented the available memory frequency settings, c represented the CPU frequency settings, and n denoted the number of CPU cores.

CoScale conducted evaluations with various workloads, encompassing compute-intensive (ILP), memory-intensive (MEM), compute-memory balanced (MID), and combinations of these workloads as inputs to the system. The results achieved by CoScale were compared to those of four other algorithms: MemScale [Deng et al., 2012], CPU DVFS, a completely uncoordinated method, and a semi-coordinated algorithm were all used. Memory and CPU frequencies were individually chosen by their individual administrators in a totally uncoordinated manner. The CPU management used an overall performance slack metric in the semi-coordinated policy to account for the resultant performance decrease caused by the memory manager's prior choice. CoScale successfully met performance targets and demonstrates robustness across the parameter search space.

2.1.1.5 Energy efficiency in storage

Cloud computing offers Storage as a Service, allowing users to remotely store and access their data as needed. Market leaders like Amazon have introduced services such as Relational Database Service (RDS) and Simple Storage Service (S3) to provide storage solutions to users. To reduce power consumption in servers, it is worth exploring secondary storage devices like Hard Disk Drives (HDD) and Flash-based Storage (SSD).

Hard Disk Drives (HDD) are the most often used secondary storage medium in data centre servers. In HDDs, the power consumption is primarily attributed to three main components: the Voice Coil Motor (VCM), Spindle Motor (SPM), and electronics [Gurumurthi et al., 2012]. Notably, the SPM component is responsible for the majority of electricity usage in HDDs.

Hibernator [Zhu et al., 2005] is a disk array system designed to manage energy while meeting response time performance goals. The authors proposed a disk-speed-setting algorithm called Coarse-grain Response (CR), which leveraged workload observations to optimize the energy usage of optical disks without compromising performance objectives.

Tomes et al. [Tomes et al., 2017] demonstrated that HDD RAID configurations performed better in terms of sequential write performance. Furthermore, Tomes et al. [Tomes et al., 2017] highlighted that HDD RAIDs consume more energy when the storage system is idle.

Mohseni et al. [Mohseni et al., 2019] described a task scheduling method for multi-CPU and multi-Hard Disk Drive (HDD) systems that minimized the number of missed jobs. The technique prioritized jobs based on readiness time, CPU execution time, CPU completion time, and HDD read/write time. Job execution time was adjusted according to changes in CPU frequency and HDD RPM levels to find an optimal trade-off between energy usage and overall execution time.

Solid State Drives (SSDs) utilize flash memory technology, which employs floating gate transistors to store data. However, write operations consume more energy and can damage the oxide layer, limiting the number of write cycles that can be performed.

Tomes et al. [Tomes et al., 2017] demonstrated that traditional power-off-based energy conservation mechanisms used in HDD RAIDs were not effective for SSD RAIDs due to high operational costs and limited energy conservation potential. SSD RAIDs consumed less energy than HDD RAIDs in common server workloads such as mail, file, and web. The authors also showed that RAID 0 performed better in terms of energy consumption for write-intensive and mixed workloads. However, hybrid arrays combining both HDD and SSD were not considered.

To increase the lifespan of SSDs, Wu et al. [Wu et al., 2018] proposed decreasing random access and promoting sequential access patterns. They employed a sequence recognition module to identify sequential requests, dynamically dividing the cache into RAM and SSD. Sequential requests are directed to the RAM's sequential portion, while cache misses retrieved data from HDD arrays or the random portion of the array. Simulation results showed up to a 45% improvement in SSD performance. However, the use of one LRU list per user for read and write requests may led to memory contention at the cache level.

Gao et al. [Gao et al., 2018] offered a Load Aware data transfer strategy for widespread monitoring utilizing a hybrid storage framework. Video chunks were migrated to SSDs to improve node performance. Although a hybrid architecture was employed for secondary storage, the Cluster-Level Data Migration (CLDM) algorithm identified high and low load nodes through a naive approach of comparing them with average load. However, utilizing clustering and classification techniques could yield more accurate and improved results. Tan et al. [Tan et al., 2018] applied the hybrid secondary storage architecture to Big Data services, demonstrating that using SSDs for local temporary data storage, even in I/O intensive jobs like TeraSort, could achieve good utility. Yin et al. [Yin et al., 2018] introduced DuoFS, a dependable and energy-efficient storage system that balanced energy effectiveness, durability, and efficiency in parallel storage systems by combining HDD and SSD-based file systems. DuoFS employed a transformational middleware layer that routed files to one of two parallel file systems based on their access frequency. By duplicating popular data in SSDs, DuoFS significantly reduced energy consumption, mitigated key causes of storage system reliability issues, and leveraged the high I/O performance of SSDs. Notably, DuoFS utilized highly energy-efficient flash-based storage nodes.

2.1.1.6 Energy efficiency in network components

A fundamental aspect of cloud computing is the ability to access infrastructure remotely, which relies on a well-connected networking infrastructure that serves as a foundation for all other components both inside and outside the data center. When considering energy consumption at this high level of abstraction, the energy utilization of communication links and intermediate networking hardware plays a significant role. Achieving energy-efficient cloud computing begins with effective architectural planning of the data center network, followed by reducing energy consumption in network elements such as transmission systems, routers, and network switches. This involves operating data center components at high utilization rates and transitioning underutilized components like switches and servers into lower-energy modes.

Terzi et al. [Terzi et al., 2021] proposed a novel approach for networking in data centers, replacing wired connections with high data-rate, point-to-point wireless links operating in the 60 GHz frequency band. Heller et al. [Heller et al., 2010] introduced a power management manager capable of dynamically adjusting power elements, active switches, and links to handle varying data center loads. The authors compared different mechanisms for achieving minimum power subnets across various traffic patterns. Leveraging Software Defined Networking (SDN), Li et al. [Li et al., 2014] explored a new energy-aware mechanism for flow scheduling, employing a different routing path for each flow in the time dimension. Wang et al. [Wang et al., 2012] presented a technique that efficiently managed traffic by utilizing virtualized routing topologies and employing adaptive traffic control and offline link weight optimization. Kliazovich et al. [Kliazovich et al., 2013] focused on the communication fabric and proposed e-STAB, a scheduling solution that considered the traffic requirements of cloud applications. Guzek et al. [Guzek et al., 2015] designed the HEROS mechanism for load balancing, enabling energy-efficient resource allocation in heterogeneous environments while considering system heterogeneity.

Networking devices such as routers often operate at low average usage levels, consuming 80%-90% of their peak power [Niccolini et al., 2019]. Energy consumption can be reduced by operating network switches and transport systems at higher utilization rates. Mahadevan et al. [Mahadevan et al., 2010] demonstrated that switch power consumption depends on the number of active ports and their operating speed. Ahn et al. [Ahn et al., 2014] examined the power usage of edge routers and established a direct relationship between power consumption, link usage, and packet size. Similarly, [Abts et al., 2010] showed that the energy cost of inter-server communication could be proportional to the amount of data transferred.

Hardware-based approaches for enhancing energy efficiency in cloud data centers have notable limitations. While advancements in energy-efficient hardware components can contribute to overall energy savings, they often entail fixed consumption patterns that may not dynamically adapt to varying workloads. The upfront costs and logistical challenges associated with deploying new hardware or retrofitting existing infrastructure pose barriers for some organizations. Moreover, hardware-based solutions may have limited impact on idle power consumption and struggle to address inefficiencies in virtualized environments, where the abstraction of hardware layers dilutes their effectiveness. Additionally, the pace of innovation in the hardware industry and potential incompatibility with legacy systems can hinder widespread adoption. Furthermore, these approaches may not comprehensively address software-related inefficiencies, emphasizing the need for a holistic approach that considers both hardware and software optimizations, as well as operational strategies, to achieve sustainable energy efficiency in cloud data centers.

Apart from the above-mentioned hardware techniques, there are software-based techniques that applied at the virtualization layer, OS layer etc. The subcategories are as follows:

2.1.2 Software based techniques

2.1.2.1 Virtualization related techniques

Virtualization employs software to establish an abstract layer above computer hardware, enabling the partitioning of physical machine resources such as storage, disks, and more into multiple virtual machines (VMs). Each user's VM can be assigned an individual operating system on a single physical machine, ensuring VM performance and isolation in case of failures. To enable efficient operations, a Virtual Machine Monitor (VMM) or Hypervisor is responsible for multiplexing resources and power management. In recent years, virtualization technology has become prominent in computer system architecture, providing various benefits such as server consolidation, transparent migration, and secure computing while maintaining compatibility with existing operating systems and applications. In contemporary virtualized environments where multiple VMs run on the same core, there is a need for centralized power management controlled by the hypervisor. However, this approach has limitations. First, it does not allow users to specify individual power control schemes for each VM or client. Second, it can impact the energy efficiency of VMs, especially when they require different energy management strategies, leading to competition among them. To address these issues, Kang et al. [Kang et al., 2018] proposed a per-VM power control method that allowed each VM's guest operating system to implement its own chosen energy management strategy, preventing conflicts between VMs. Compared to the default on-demand governor of the Xen hypervisor, their approach called Virtual performance (VIP) reduced power consumption and improved completion time for CPU-intensive applications by up to 27% and 32%, respectively, while still meeting the latency-sensitive implementation's service-level agreement (SLA).

Furthermore, Xiao et al. [Xiao et al., 2021] focused on energy-efficient optimization of VM scheduling models and I/O virtualization paradigms. They introduced a power-fairness credit sequencing approach with a novel I/O offset method to achieve fast I/O performance while maximizing energy conservation. Additionally, Prabhakaran et al. [Prabhakaran et al., 2021] introduced VM resource calibration as a means to reduce energy usage in virtual servers. They developed a system that utilized controlled feedback architecture and power monitoring services to achieve less usage of energy.

Preliminary VM distribution plays a crucial role in resource management policies, exerting a significant influence on efficiacy of DC and usage of energy. A well-planned VM placement strategy can reduce system overhead by minimizing the need for migrations. Kabir et al. [Kabir et al., 2014] delved into this issue within the context of a hierarchical deployment model for cloud service providers, encompassing cloud, cluster, and host levels. This model facilitated efficient management of geographically distributed infrastructure, aiming for scalability. Given this hierarchical structure, it necessitates an intelligent VM placement approach, complete with mechanisms for selecting cloud clusters and nodes, all aimed at minimizing resource fragmentation and enhancing energy efficiency.

There are two types of placement strategies: centralised and hierarchical. Khosravi et al. [Khosravi et al., 2013] proposed a centralised virtual machine deployment technique for dispersed cloud data centres. The main objective was to reduce both electricity usage and the carbon footprint. They created a data system that continually updates utilisation statistics for clouds, clusters, and hosts, allowing for centralised decision-making and resource optimisation. Their strategy took into account dispersed data centres with varied carbon footprints and Power Usage Effectiveness (PUE) values. They carried out an in-depth review of the efficiency of energy using various bin-packing strategies.
Similarly, Forestiero et al. [Forestiero et al., 2014] introduced EcoMultiCloud. This hierarchical approach attained energy efficiency similar to the centralized ECE solution but offered greater flexibility. Since it was hierarchical in structure, it enabled individual data centers to pick their own interior VM placement methods.
Cagar et al. [Cagar et al., 2013] presented an online placement technique for VM integrated into the hALT (harmony of Living Together) middleware. This technique took into consideration the workload characteristics of VMs and accounted for performance interference when determining suitable placements for each VM.
Meng et al. [Meng et al., 2010] presented the notion of Statistical Multiplexing to VM placement, which is an approach for efficiently accommodat-

ing additional VMs by multiplexed server resources. Their method, known as joint-VM provisioning, entailed consolidating numerous VMs depending on workload characteristics. Statistical multiplexing allows VMs to share resources with co-located VMs during workload spikes. To meet Quality of Service (QoS) requirements, they defined performance constraints for each VM, ensuring sufficient capacity to meet specific application performance levels. Three strategies were provided by the authors for creating performance limits, choosing co-located VMs, and evaluating the overall resource consumption of integrated VMs with complimentary workloads. Joint-VM provisioning considerably increased the consumption of energy efficiency when tested utilizing VM workloads from an industry-standard data centre.

Chen et al. [Chen et al., 2011] conducted a study on VM placement, with a specific focus on consolidating a larger number of VMs on servers. ES took into account Statistical Multiplexing principles, considering factors influenced the aggregated resource demand on the server where the VM was placed. The effective size concept extended from effective bandwidth and also considered the correlation between VM workloads. Taking the correlation coefficient into account, the effective size of a VM was determined by its own demand as well as the demand of co-located VMs. The suggested VM placement technique, which has a temporal complexity of O(1), found the best location for VMs. The study took into account Poisson and standard distributions for VM workloads and tested the method using simulations on a real cloud workload trace. When compared to a generic consolidation method, the effective size technique revealed energy savings ranging from 10% to 23%. The optimisation technique concentrated solely on one dimension: the CPU consumption of the VMs.

Hypervisor technology plays a pivotal role in consolidating virtual machines on physical servers, and extensive research has been conducted to investigate VM consolidation strategies that might improve data centre energy usage efficiency.

Gmach et al. [Gmach et al., 2009] suggested an efficient technique in terms of energy reactive migration regulator that recognised overloaded or under-

loaded hosts. Overload and underload detection were based on the server's CPU and memory utilisation exceeding or falling below a predefined threshold, respectively. Beloglazov et al. [Beloglazov et al., 2010] took a similar method when investigating the influence of these two criteria on overall data centre energy usage and SLA breaches. Underload and overload thresholds were judged to be 30% and 70% efficient, correspondingly, when the overall energy use and average SLA breaches were considered. Unlike Gmach et al. [Gmach et al., 2009], the approach proposed in [Beloglazov et al., 2010] was not dependent on the workload type.
Beloglazov et al. [Beloglazov et al., 2010] enhanced the previously mentioned approach [Beloglazov et al., 2010] by introducing an automatic adjustment mechanism for the underload and overload thresholds. The previous method, which relied on fixed threshold values, was found to be inadequate for addressing the dynamic and unpredictable workload behaviors commonly encountered in cloud environments. The automation is achieved by conducting statistical analysis on historical data derived from virtual machine workloads. Host CPU utilization is assumed to follow a t-distribution. This adaptive approach demonstrates significant improvements in terms of Quality of Service (QoS) contrast to set criteria while still attaining energy savings.
Cloud providers are tasked with maintaining the promised Quality of Service (QoS) to their customers throughout the consolidation process. However, QoS can be compromised due to host overloading. To solve this issue, Beloglazov et al. [Beloglazov et al., 2010] developed an approach for identifying host overloads that maintains QoS while conserving energy. In the case of any known fixed workload, this technique can determine the ideal option for overload identification. The primary goal is to maximize the time between migrations while adhering to a specified QoS target, based on a Markov chain model.

2.1.2.2 Virtualization at the Operating System (OS) Level (Containers)

The Platform as a Service (PaaS) model has brought about a transformative shift in application development by eliminating the need for infrastructure management and expediting the development process. Application isolation is achieved in this paradigm by the use of containers, which may run on both physical machines PMs, and VMs. Containers create isolated virtual environments that do not require the use of intermediary monitoring tools such as hypervisors. Containers have garnered considerable popularity and are emerging as a predominant deployment strategy in cloud computing. Within the cloud environment, consolidation techniques are widely adopted to optimize resource utilization and reduce power consumption. Piraghaj et al. [Piraghaj et al., 2015] directed their attention to container consolidation and conducted a comparative analysis of various algorithms aimed at minimizing power consumption. They assessed these algorithms using parameters such as SLA violations, energy consumption, container transfer rates, and the number of generated VMs. The consolidation of container-based services presents challenges, as it can lead to substantial power consumption within cloud data centers due to limited management control over data center systems. Shi et al. [Shi et al., 2018] devised the TMPSO algorithm to facilitate energy-aware consolidation of containers. The proposed algorithm combined heuristic and greedy optimization mechanisms to strike a balance between computational demands and performance costs.

Additionally, Chen et al. [Chen et al., 2019] introduced a stable matching method called many-to-one and a container placement technique named MLSM. They implemented an early container hosting technology to reduce migration durations through a reliable matching mechanism. The algorithm employed similarity algorithms as a decision-making strategy for matching containers with VMs, with the resource usage rate used to prioritize the virtual machine preference list. The simulation results showed that the technique can save an average of 12.8% more energy than the First Fit approach.

Furthermore, Al-Moalmi et al. [Al-Moalmi et al., 2021] tackled issues related to container and VM placement within a Container as a Service (CaaS) environment, with a focus on optimizing power consumption and resource utilization. They proposed an algorithm based on the Whale Optimization Algorithm (WOA) to address container and VM placement challenges. In essence, each type of container can be hosted by a single VM, and each type of VM can be hosted by a single PM.

Research in energy-efficient resource-utilization strategies for OS container systems are mostly focused on algorithms for OS container placement. Dong et al. [Dong et al., 2014] proposed the Most Efficient Server First (MESF) container placement system, which prioritised allocating containers to the most energy-effective computers first. The machine with the least increase in energy use when hosting the container was the most energy-efficient equipment for each container. The MESF strategy dramatically decreased energy usage in comparison to the Least Allocated Server First (LASF) and random scheduling plans, according to simulation findings utilising actual Google cluster data as job input and machine configuration. In addition, the study provided a new viewpoint on assessing a cloud data center's utilisation of energy.

Pandit et al. [Pandit et al., 2014] also delved into the challenge of efficient resource allocation. Unlike the normal n-dimensional bin packing issue, which considers a bin full when all sub-bins achieve capacity, the resource distribution problem considers a bin full when any sub-bin (e.g., CPU) hits capacity. Pandit et al. [Pandit et al., 2014] utilised Simulated Annealing (SA) to build an effective resource allocation method. SA is a method often used for discrete search space issues, such as bin packing problems. The suggested resource allocation algorithm outperformed the frequently used First Come First Serve (FCFS) allocation strategy in terms of resource utilisation.
Mesos [Hindman et al., 2011] also makes use of OS containers to offer the necessary workload segregation. The Mesos platform allowed cluster computing systems with multiple programming styles to share standard clusters. Mesos utilized a two-level scheduling approach called "resource offers." The studied workloads were developed using Hadoop and MPI programming models. The results demonstrated that Mesos is highly scalable and fault-tolerant, improving resource utilization with less than 4% overhead.

2.1.2.3 Workload Characterization

The study of resource management approaches in cloud data centres has grown significantly, with the objective of reducing energy use. However, because to competitive and security concerns, cloud providers frequently do not publish their workloads, resulting in a scarcity of open-source cloud backend traces. As a result, a great deal of studies are lacking in understanding of the ever-evolving nature of consumer preferences and workload fluctuations.
The readily available nature of cloud backend traces allows academics to mimic real-world cloud data centre workloads, allowing them to assess the practicality of suggested heuristics. Around 2009, Google made its early traces public, triggering a flurry of research activities centred on capacity planning, scheduling, and optimisation.
The workload handled by a system influences its performance in addition to its hardware and software components. Knowing the workload, according to Feitelson [Feitelson, 2015], was more important than inventing new scheduling methods. If the tested systems did not correctly replicate the input workload, the recommended rules or algorithms' outputs may not correspond to real-world conditions.
The quantity of work allocated to a computer system that must be done within a specified timeframe is referred to as its workload. A typical system workload consists of activities and user groups submitting requests to the data centre. In Google's workload, for instance, tasks are the essential elements of a job, with a job generally consisting of one or more tasks. In this sense, users refer to Google's personnel or services.
It is critical to study the input workload that drives the researched system in order to characterise the burden. The input workload must closely reflect the real-world workload for reliable performance evaluation of a computer system. According to Ferrari [Ferrari, 1972], there are three methods for determining the input workload:

1) Natural Technique:

The most basic approach entailed using real weights received directly from the system's log file. Urgaonkar et al. [Urgaonkar et al., 2010] investigated the topic of optimum allocation of resources and energy use in cloud data centres using real traces from a variety of uses. Anselmi et al. [Anselmi et al., 2008] validated their suggested method for the Service Consolidation Problem (SCP) using real workloads from 41 servers. In various research, PlanetLab VM records were used as input workload to evaluate the consolidation approach.

2) Artificial Technique:
The artificial approach entails creating and deploying a workload that is unrelated to the real one. To simulate web server workloads, Mohan Raj and Shriran [Mohan Raj and Shriran, 2011] used synthetic demands based on the Poisson dispersion.

3) Hybrid Technique:
The hybrid approach merges collecting a real workload with the creation of a test workload from portions of the real task. Hindman et al. [Hindman et al., 2011] tested Mesos using CPU and IO-intensive workloads drawn from Facebook cloud backend logs as well as Hadoop and MPI services.

According to Calzarossa and Swerazzi [Calzarossa et al., 1993], the workload modelling process may be separated into three major parts. The initial stage was to build the model by choosing fundamental components like user contribution rates and characteristics. A set of requirements was also included in order to evaluate the suggested model. The needed parameters for modelling were gathered in the second stage while the workload was running in the system. Lastly, a statistical analysis of the acquired data was done in the final stage.
Software-based approaches for enhancing energy efficiency in cloud data centers face several limitations. These include dependency on well-designed applications, as software optimizations rely on efficient resource utilization within applications. Legacy systems present a challenge, as retrofitting or redesigning older software to incorporate energy-efficient practices can

be complex and time-consuming. The dynamic variability of workloads in cloud environments poses difficulties for software-based optimizations to adapt to sudden changes in demand, impacting resource utilization. Additionally, in multi-tenant cloud environments, coordinating fair resource allocation among diverse workloads can be challenging for software-only solutions. Furthermore, software-based approaches may struggle to address idle power consumption, and their effectiveness in virtualized environments can be hindered by suboptimal virtualization practices and potential computational overhead. A holistic strategy that integrates both hardware and software optimizations is crucial to overcoming these limitations and achieving comprehensive energy efficiency in cloud data centers.

2.2 Summary

In this chapter, the focus was on conducting an extensive literature review concerning the enhancement of energy efficiency within data centers. The review encompassed a broad spectrum of techniques, both hardware and software-based, aimed at mitigating power consumption while maintaining or enhancing operational performance. Hardware-related strategies explored methods for optimizing various components, including CPUs, GPUs, memory, storage, and networking, with the ultimate goal of maximizing resource efficiency. Conversely, software-based approaches spanned multiple layers such as virtualization, operating systems, middleware, and applications, all geared toward streamlining resource allocation, improving process management, enhancing security, and optimizing application performance. Collectively, these techniques serve as a comprehensive resource for achieving energy efficiency in contemporary data centers, a critical endeavor for reducing environmental impact and operational expenses while meeting the escalating demands of the digital age.

Chapter 3

Workload Characterization and Categorization

This chapter delves into the critical topic of workload characterization and categorization within cloud data centers. It explores various clustering techniques employed for in-depth analysis of the Bit Brains Trace dataset, offering valuable insights into how these methods contribute to a better understanding of workload patterns and resource utilization in cloud environments.

3.1 Introduction

Cloud platforms are in great demand for hosting a wide range of workloads, particularly online applications that need strict SLAs between the Cloud Service Provider (CSP) and the client. These services involve a wide set of QoS standards in terms of accessibility, reliability, and efficiency. Workloads frequently migrated to cloud systems require memory, CPU, network bandwidth, and storage. These workloads are classified as distinct resource intense workloads based on the resources they utilise more than others. These workloads' actual resource usage is frequently lower than the resources they have requested. Service providers profit from this practise by promising more resources at lower costs than the actual number of resources they have, relying on the knowledge that the vast majority of customers' applications will not run at full capacity. CSP uses the cloud's dynamic provisioning feature to deliver on-demand performance. Recognizing workload behaviour in a cloud Data Center (DC) is crucial because it allows for the elastic scaling up and down of supplied services, which are critical to the DC's capabilities. Workload characterisation is being utilised to estimate resource requirements, allowing for more effective capacity management, allocation, and resource deployment. The workload is commonly characterised using one of two methods: trace-based (Characterizing Application Workloads on CPU Utilization for Utility Computing, n.d.) or model-based (S. Huang and Feng, 2009), (Moro et al., 2009) (Delimitrou and Kozyrakis, 2011). Because it is indifferent with the operating platform on which the trace was documented, the model-based process is preferred over the trace-based procedure. Since trace-based strategies have a limited number of production and quality traces, they require continuous tinkering with workload characteristics to ensure consistency with a new data centre environment, making them less efficient than model-based strategies. The vast majority of workloads in cloud DC are a mash-up of many applications. It is incredibly difficult to develop a completely unified strategy to estimate the future utilisation of these various application areas' resources. These tasks exhibit a wide range of behaviour in terms of periodicity, co-relation, and recurrent tendencies. Workload classification necessitates a more in-depth knowledge of workload behaviour and attributes. Nevertheless, there have been less research on workload characterisation due to a lack of open-source workload traces. Workloads are categorised based on computational paradigms, technology stack, resources, and applications, as seen in Figure 3.1. Workloads are categorized into two types, namely batch workloads and interactive workloads, based on their processing methodology. Their classification based on resource requirements includes memory, CPU, I/O, and database. These requirements encompass scalability, flexibility, extensibility, and administration. Additionally, the cloud infrastructure should offer capabilities that fulfil the organization's top-tier demands, such as privacy, reliable performance, and cost-effectiveness. Computer equipment is organized differently across various computing environments, with data being shared among them to analyze and solve problems. Workloads can be further classified into three types based on generation: Synthetic, Real, and Cloud. Furthermore, workloads are classified into four categories, including Web, Social Network, Video Service, and others, on the basis of application. Figure 3.1 shows the classification of workload.

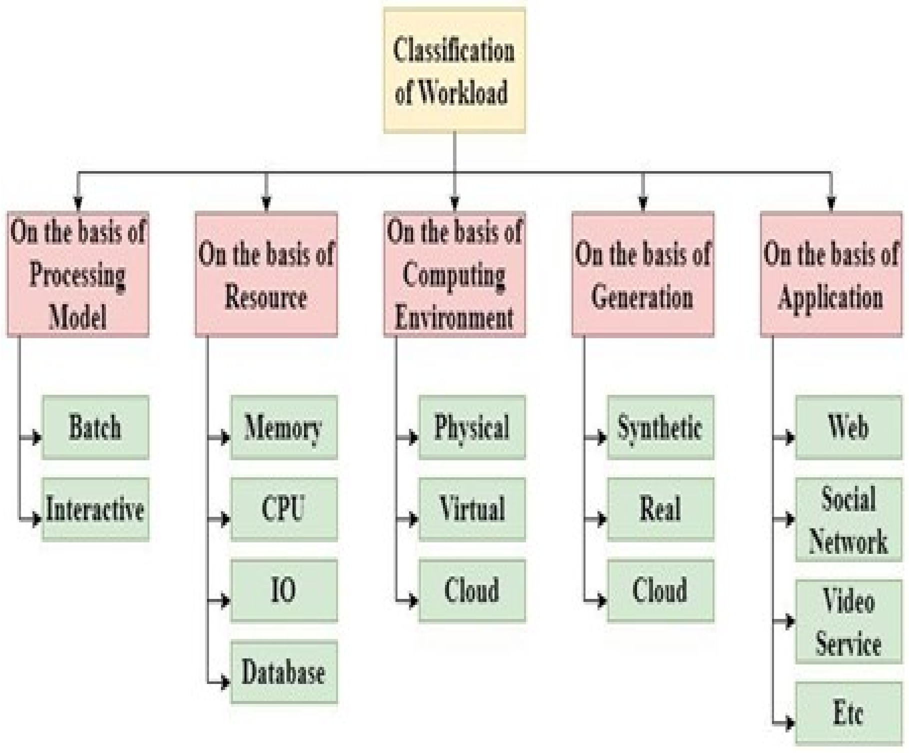

Classification of Workload
On the basis of Processing Model
Batch
Interactive
On the basis of Resource
Memory
CPU
IO
Database
On the basis of Computing Environment
Physical
Virtual
Cloud
On the basis of Generation
Synthetic
Real
Cloud
On the basis of Application
Web
Social Network
Video Service
Etc

Figure 3.1: Categories of Workload

3.2 Literature Review

Cloud workload characterisation and categorization is an essential research subject for better understanding of workloads and efficiently managing cloud resources. The four most common workload traces available in this domain are Google Cluster Trace (GCT) (Charles Reiss et al., 2012), Bit Brains Trace (BBT) (Shen et al., 2015), Alibaba (Alibaba/Clusterdata: Cluster Data Collected from Production Clusters in Alibaba for Cluster Management Research, n.d.), Yahoo Trace (Webscope — Yahoo Labs, n.d.), and Wikipedia (Wikipedia Access Traces — WikiBench, n.d.). Workload characteristics for DC must be statistically aggregated and analysed in order to forecast the data center's future resource requirements. Many studies have used statistical approaches such as Pearson Coefficient of Correlation (PCC), standard deviation, mean, and other related and current methodologies (Birke et al., 2014). Calzarossa and other authors (Carla et al., 2016) created a list of standard internet workloads, workloads from social networks, streaming platforms, mobile applications, and cloud computing infrastructure facilities. They examined the unique characteristics of these workloads and described the methodologies and modeling techniques employed to characterize them. With time-series analysis, Ali-Eldin et al. (Ali-Eldin et al., 2014) explored the time series of Wikipedia's workload and discovered that it is completely predictable and has strong seasonal variation. Self-similarity and burstiness are two of the main workload characteristics according to Yin et al. (Yin et al., 2015). They created a workload generator for cloud computing that exhibits burstiness and self-similarity. They have used NetEase traces for analysis and comparison. Wang et al. (Wang et al., 2015) demonstrated that the effectiveness of dynamic resizing is heavily influenced by workload process statistics. Specifically, both the long-term non-stationarities of the workload (such as the peak-to-mean ratio) and the short-term stochastic behavior (such as the burstiness of arrivals) are significant factors. To show case the influence of these factors, the researchers combined optimization-based modelling of long-term trends with stochastic modelling of short-term dynamics. Workloads taken from real-world data centre traces were used in the trials. Zhang et al. (H. Zhang et al., 2014) created a workload factoring service in the context of proactive workload management. This technology allows on-premise and off-premise architectures to work together to host web-based applications. The important feature of this technique is that it successfully separates the base workload from the flash crowd task, which are intrinsically independent components of the application workload. A speedy frequent data item identification algorithm is at the heart of the intelligent workload factoring service. This algorithm enables the categorization of incoming requests based not only on their volume but also on their data content, taking into account the evolving popularity of application data. They used Yahoo video stream data for their analysis.

Recognizing and forecasting patterns in cloud workloads is a challenging problem addressed by Patel et al. (J. Patel et al., 2015). They introduced an innovative resource estimation approach based on clustering that groups tasks with similar characteristics into the same cluster. They utilized the Google cluster dataset for their analysis.

Panneerselvam et al. (Panneerselvam et al., 2014) outlined the cloud workload classification and characterization metrics needed for training and modeling prediction algorithms. They examined two commonly used prediction algorithms and evaluated their prediction accuracy in forecasting memory-heavy and CPU-intensive workloads within the MATLAB environment.

Due to the scarcity of open-source workload traces, only a few attempts at characterization of cloud DC workloads have been made so far. The following are some of the most well-known research projects: The BBT dataset that represents business critical workloads was analyzed by the authors in (Shen et al., 2015). To characterize the workload, statistical methods such as standard deviation and mean, Pearson Correlation Coefficient (PCC), Autocorrelation Function, Peak to Mean Ratio and Coefficient of Variation were used. The analysis was carried out using basic statistical and time pattern analysis. The findings from the study are as: (a) there is a strong correlation among demanded memory and CPU utilization; (b) Memory and CPU utilizations are easy to predict over short time periods; and (c) disk and network utilization follow patterns, implying that prediction granularity is measured in days. (d) Peak workloads can vary from 10 to 10,000 times greater than the average workloads, depending on the type of resources involved. The authors in (Characterizing Task Usage Shapes in Google Compute Clusters – Google Research, n.d.) presented a task usage shape classification that precisely reproduces the technical specifications of historical data on average job wait time and machine resource utilization. They utilized real time data from Google and found that merely simulating the job mean usage can gain considerable precision in trying to replicate resource utilization and task wait time. One major drawback is that the results are very complex and produce complex characterization of task shape classification. The authors in (Rasheduzzaman et al., 2014) examined production workload trace (version 2) by Google and utilized

K-means clustering to group similar jobs together. They demonstrated a simple method for establishing workload attributes as well as knowledge and insights for workload performance on cluster machines. The authors did not use the complete trace to perform the analysis, which led to the discrepancy in the results. Moro (Moro et al., 2009) introduced an innovative method to precisely assess the execution workload performed by a computer. Their proposed method involved direct utilization of the memory reference sequence generated during program execution. The memory reference sequences were treated as sequences of floating-point numbers and subjected to analysis using signal processing techniques. Spectral analysis was employed during the feature extraction phase, while Ergodic Continuous Hidden Markov Models (ECHMMs) were used in the pattern matching phase. The proposed algorithms' effectiveness was assessed through trace-driven simulations using SPEC 2000 workloads. Cheng et al. (2018) conducted a comprehensive case study characterizing Alibaba's co-located long-running and batch task workloads across various dimensions. They investigated patterns of resource demand and reservation, resource utilisation, workload dynamics, straggler issues, and the interaction and influence of co-located workloads. A notable strength of their work is the use of Alibaba DC traces and workload categorization based on resource utilization.

Mishra et al. (2010) presented a workload categorization technique and applied it to the Google Cloud Backend, one of the biggest cloud backends. They employed established statistical clustering techniques, assessed workload aspects, used methods like k-means for task clustering, evaluated qualitative cartesian coordinates within workload elements, and combined adjacent task clusters to reduce prediction variables. Their methodology resulted in eight workloads across several Google compute clusters. They demonstrated that workload features related to the number of tasks and resource consumption remain consistent for the same compute cluster but may vary among clusters.

Ismaeel et al. (2019) presented a novel method for selecting the appropriate task clustering approach in data centers based on validation indices and result correlation. They devised an efficient pre-processing strategy, reducing the big data challenge to a compact 2D matrix involving independent jobs with CPU and memory requirements, using the Google Cloud trace for analysis.

Shekhawat et al. (2018) provided a method for categorising and characterising data centre workloads on the basis of resource use. They applied K-Means clustering to group workloads into clusters and explored seven distinct machine-learning techniques for workload classification and distribution approximation. They also presented an approach for assessing the relevance of various categorization attributes, although they did not compare their results with other clustering algorithms.

E. Patel and Kushwaha (2020) discussed the cluster effectiveness of K-Means and Gaussian Mixture Models (GMM) for characterizing cloud workloads based on CPU and memory utilization. They utilized the Google cluster trace and Bit Brains dataset but did not consider other parameters like disk and network usage or evaluate the performance of the two clustering approaches.

Table 3.1 provides a comparison of the work conducted in the field of workload characterization.

Table 3.1: Comparison of work done in the domain of workload characterization

Work	**Advantages**	**Disadvantages**
Work (Carla et al., 2016) is focused on conventional web workloads, and the workloads associated with social network, mobile and video services are considered. It discusses different techniques for their characterization.	Many features have been explored from several viewpoints, such as qualitative and quantitative elements relating to technology foundations, user engagements with services and apps.	The main disadvantage is that the authors did not consider using different machine learning techniques for the classification purpose.

Studied (Ali-Eldin et al., 2014) the trend of workload by time-series analysis, descriptive statistics and polynomial splines.	The short-term prediction algorithm can anticipate workload with a mean absolute percentage error of less than 2 percent.	The main disadvantage is that they did not consider using the advanced technologies like machine learning for trend prediction and also, they did not build the workload generator.
Developed (Yin et al., 2015) a workload generator for cloud computing based on a superposition of 2-state Markov Modulated Possion Processes (MMPP2s).	The suggested generator may generate workloads with both the necessary burstiness and self-similarity. A thorough empirical examination proves BURSE's correctness, durability, and usefulness.	They did not consider comparing the proposed approach with already existing techniques.
Showed (Wang et al., 2015) that the value of dynamic resizing is highly dependent on statistics of the workload process	They provided a novel model that incorporates SLA features.	They did not consider comparing the proposed approach with already existing techniques.

Created a service for workload factoring (H. Zhang et al., 2014)	A viable and cost-effective way for maximising the utilisation of public cloud services in conjunction with their privately-owned (legacy) data centres.	The main disadvantage is that they did not consider using the advanced technologies like machine learning for workload prediction rather rely on basic statistical methods.
Presented (J. Patel et al., 2015) an innovative resource estimation approach based on clustering that have similar characteristics into the same cluster.	They created a workload model that estimates the workload behaviour of randomly sampled processes from the trace log.	The resource management is difficult due to the proposed workload estimation problem.
They (Panneerselvam et al., 2014) specified the Cloud workload classification and characterisation metrics that are required for training and modelling prediction algorithms.	The studies demonstrate that CPU-intensive workloads have a higher estimation error rate than memory-intensive workloads.	They did not investigate the efficiencies of the prediction techniques.
Statistical methods such as standard deviation and mean, PCC, Autocorrelation Function, Peak to Mean Ratio and Coefficient of Variation were used (Shen et al., 2015).	The analysis was carried out using basic statistical methods.	They have not used the machine learning approaches for better classification.

Characterization of task use shape that properly replicates historical trace performance features in terms of average task wait time and machine resource utilisation (Zhang et al., 2011).	Utilized real time data from Google and found that merely simulating the job mean usage can gain considerable precision in trying to replicate resource utilization and task wait time.	One major drawback is that the results are very complex and produce complex characterization of task shape classification.
Examined production workload trace (version 2) by Google and utilized K-means clustering to group similar jobs together (Rasheduzzaman et al., 2014).	Demonstrated a simple method for establishing workload attributes as well as knowledge and insights for workload performance on cluster machines.	Did not use the complete trace to perform the analysis, which led to the discrepancy in the results.
Described a method for characterisation of workload using ergodic hidden Markov models (Moro et al., 2009).	The suggested techniques are tested using SPEC 2000 workloads and trace driven models. They demonstrate that ECHMMs obtain an average classification accuracy of 76 percent across eight different workloads.	They did not consider comparing the proposed approach with already existing techniques.

Characterized the batch instance workloads based on: CPU utilization, memory utilization, and job timeframe into three different categories (Cheng et al., 2018).	The primary strength lies in the fact that the authors have used the traces of Alibaba DC and have done workload categorization based on resource utilization.	They did not consider comparing the proposed approach with already existing techniques and did not use the AI methods.
Mishra et al. (2010) proposed a workload categorization technique and its implementation to the Google Cloud Backend, probably the world's largest cloud backend.	Their methodology yields eight workloads when applied to several Google compute clusters.	They did not consider the job constraints and they did not take the entire dataset for analysis which is the major drawback of their proposed approach.
Introduced (Ismaeel et al., 2019) a new methodological process for selecting the appropriate task clustering approach in DC based on clustering purpose, validation indices, and result correlation.	Developed an effective pre-processing strategy, reducing the big data challenge to a compact 2D matrix of independent jobs.	Used just CPU and memory requirements.

Proposed (Shekhawat et al., 2018) a technique for classifying and characterizing DC workloads based on the resource utilization.	They have used K Means clustering for grouping the workload into clusters. Seven distinct machine-learning techniques have been used and compared.	Not considered and compared the results with any other clustering algorithm.
Compare the cluster representation of the two strategies for heterogeneity in resource utilization of cloud workloads using K Means and Gaussian Mixture Model (E. Patel and Kushwaha, 2020).	K-Means and GMM cluster efficiency for the characterization of Cloud workloads on the basis of CPU and memory utilization were presented.	Not considered the other parameters like disk and network usage and also did not evaluate the performance of two clustering approaches.

The above-mentioned work overlooked the potential benefits of utilizing various clustering techniques to validate the experimental data. Additionally, they failed to evaluate the clustering methods using diverse performance evaluation parameters, which could aid in effective workload characterization and categorization. This is crucial in predicting resource requirements, enhancing capacity management, allocation, and resource deployment. By incorporating these approaches, the classification methods can be compared and optimized for improved resource management. Workload categorization is a crucial stage in workload analysis. Model correctness is critical for workload analysis, resource use prediction, and provisioning. The classification accuracy study helps us decide which method is optimal for a particular data centre workload.

3.3 Methodology

3.3.1 Dataset Characteristics

The distributed DC at Bitbrains is a managed hosting and business-computing powerhouse. The Bits Brain Trace [Shen et al., 2015] dataset contains performance statistics for 1,750 VMs with size of around 1.16GB. Large banks, credit card firms, insurers, and others are among the company's clientele. Towers Watson and Algorithmics are two application developers whose solvency applications are hosted on Bitbrains. These applications are often used to finalise accounting information at the end of a fiscal quarter. The performance measurements for a particular VM are included in each file of the dataset. These files are divided into two categories: fastStorage and Rnd. The first trail is FastStorage, which has 1,250 virtual machines (VMs) attached to Storage Area Network (SAN) storage devices. The second trace, Rnd, has 500 virtual machines (VMs) that are either linked to a fast SAN or a considerably slower Network Attached Storage (NAS). Because storage attached to fastStorage devices is more efficient, the fastStorage trace comprises more server side and compute units than the Rnd trace. In contrast, the Rnd trace features a larger number of management units, necessitating less storage and fewer frequent access requirements.
The Bit Brains Trace dataset stands out for its versatility and robustness, offering diverse data that enhances algorithm performance across various scenarios. Inclusive of different data types and contexts, it fosters adaptable and generalized analytical models. Notably, the dataset excels in addressing edge cases and outlier scenarios, providing a realistic simulation environment for algorithm testing. This proves crucial in assessing algorithm resilience and efficacy, ensuring robust real-world performance. Researchers benefit from a holistic testing ground, enabling the development of more reliable computational models. The Bit Brains Trace dataset emerges as a valuable resource for advancing algorithmic understanding and application.

3.3.2 Clustering Algorithms

Clustering is the procedure of categorizing data elements according to their similarity score. Clustering can be comprehensive or partial, overlapping or distinct, and may also involve fuzzy distinctions. K-Means partitional clustering splits data items into non-overlapping groups (Onan, 2019). K-Means clusters are prototype-based when the cluster is represented by a prototype and all nodes in the cluster are close to it. Centroid and medoid are two popular prototypes. Clusters in Gaussian Mixture Models (GMM) are based on density. Data clustering facilitates the process of discovering and summarising aspects of interest. Clustering cloud workload traces is essential in cloud data centers because it enables effective workload characterization and categorization, which are the first steps in predicting resource requirements. By grouping similar workloads together, administrators can optimize resource allocation, capacity management, and deployment, leading to better performance and cost-effectiveness.
The selection of clustering algorithms for the Bit Brains Trace is grounded in considerations of execution time and overhead. Notably, the GMM and K Means algorithms exhibit shorter execution times compared to alternatives, leading to a mitigation of SLA violations and reduced simulation overhead. This strategic choice ensures efficient clustering in the Bit Brains Trace, emphasizing a balance between prompt execution and minimal computational burden to enhance the overall performance and reliability of simulations (Ikotun et al., 2023).

3.3.3 Classification Algorithms

The classification method is a supervised learning strategy that uses training data to determine the type of fresh observations. It is a kind of software that learns from a dataset of observations and then classifies new data. As a supervised learning method, the classification method employs labeled input data that comprises both input and output. In workload characterization and categorization, Random Forest (RF), Logistic Regression (LR), K Nearest Neighbor (KNN), Support Vector Machine (SVM) and Decision Tree (DT), Multi-Layer Perceptron (MLP), and Back Propagation Neural Network classification algorithms were utilised.

3.3.4 Process Flow

Generally, the workload in the DC consists of different types of attributes. Some of the attributes are more important when it comes to characterizing the workload. Considering the relevance distribution of qualities is crucial in figuring out the type of task. It is important as during classification stage, significance of the attribute determines how much weight has to be given to it or to a group of attributes. The main purpose of performing significance assessment is to order the attributes in terms of predictive power. Decision Tree Classifier algorithm has been used to perform the significance analysis. The top four attributes having highest significance value have been taken for further analysis. In the BBT dataset memory usage [KB], disk read throughput [KB/s], CPU usage [MHz], network transmitted throughput [KB/s], disk write throughput [KB/s] and network received throughput [KB/s] have high percentages in terms of attribute significance analysis. The disk write throughput and disk read throughput are combined to form a single attribute named disk usage. Similarly, network transmitted throughput and network received throughput are combined to form a single attribute named network usage. Normalization of the BBT dataset is done using min-max normalization. The K means technique was applied to the combined data set consisting of attributes having high significance. The technique was further applied to each attribute to calculate K. The value of number of clusters, K is determined using elbow criterion. If c represents the clusters obtained for CPU usage [MHz], m for Memory usage [KB], d for Disk usage [KB/s] and n represents clusters acquired for Network usage [KB/s]. The product of c, m, d, and n yields the number of possible workloads in the dataset. As a result, the frequency of various workloads is calculated. This analysis yielded the dataset's workload distribution. GMM clustering is applied to all attributes first, followed by individual attributes. The outcomes of both algorithms are compared using parameters such as the Calinski Harabasz index (CHI) and the Davies-Bouldin Index (DBI). CHI is also known as the Variance Ratio Criterion.

A higher CHI score denotes a model with more defined clusters. The index is the ratio of all clusters' total between-cluster and within-cluster variance. When clusters are large and well-spaced, the score is greater, which correlates to a classic cluster idea. When DBI is used to evaluate the model, a lower DBI indicates a model with greater cluster separation. This index represents the average similarity of clusters, wherein similarity is defined as a criterion that relates cluster distance to number of clusters. It distinguishes between clusters that are both remote and small. The Davies-Bouldin criterion is based on a weighted average of distances within-cluster and between-cluster. The lowest possible score is zero. A value closer to zero indicates a better partition. Davies-Bouldin scores are easier to calculate because they only use point-wise distances and the index is solely based on amounts and characteristics inherent in the dataset. After characterizing and categorizing the workload, an experiment was conducted to create the appropriate sized virtual machines. The best clustering technique was used to obtain samples for workload distribution on virtual machines. The first step of this experiment involved collecting sample output data, or workload clusters, from the best clustering technique, which were then used as input for VM sizing. Depending on the characteristics of the workload, VMs of different sizes were created and organized into clusters. The workload was mapped to different VM clusters, and as a comparison, the workload was also mapped to randomly sized VMs. The experimental results were then compared in terms of the number of VMs used. This experiment aimed to determine whether creating VMs based on workload distribution could result in more efficient resource utilization and workload management. An experimental assessment to approximate classification accuracy for various machine-learning algorithms to explore how workload distribution affects each model has been done. The main purpose of doing classification is to check which algorithm should be used when predicting the workload in the future. This is done to ensure that correct prediction algorithm would be applied in the future such that prediction of workload should be done in an efficient manner. The analysis was carried out using BBT fastStorage dataset [Shen et al., 2015]. The ratio of training and testing data is kept as 70 and 30 respectively. Figure 3.2 shows the methodology followed to characterize and categorize the workload.

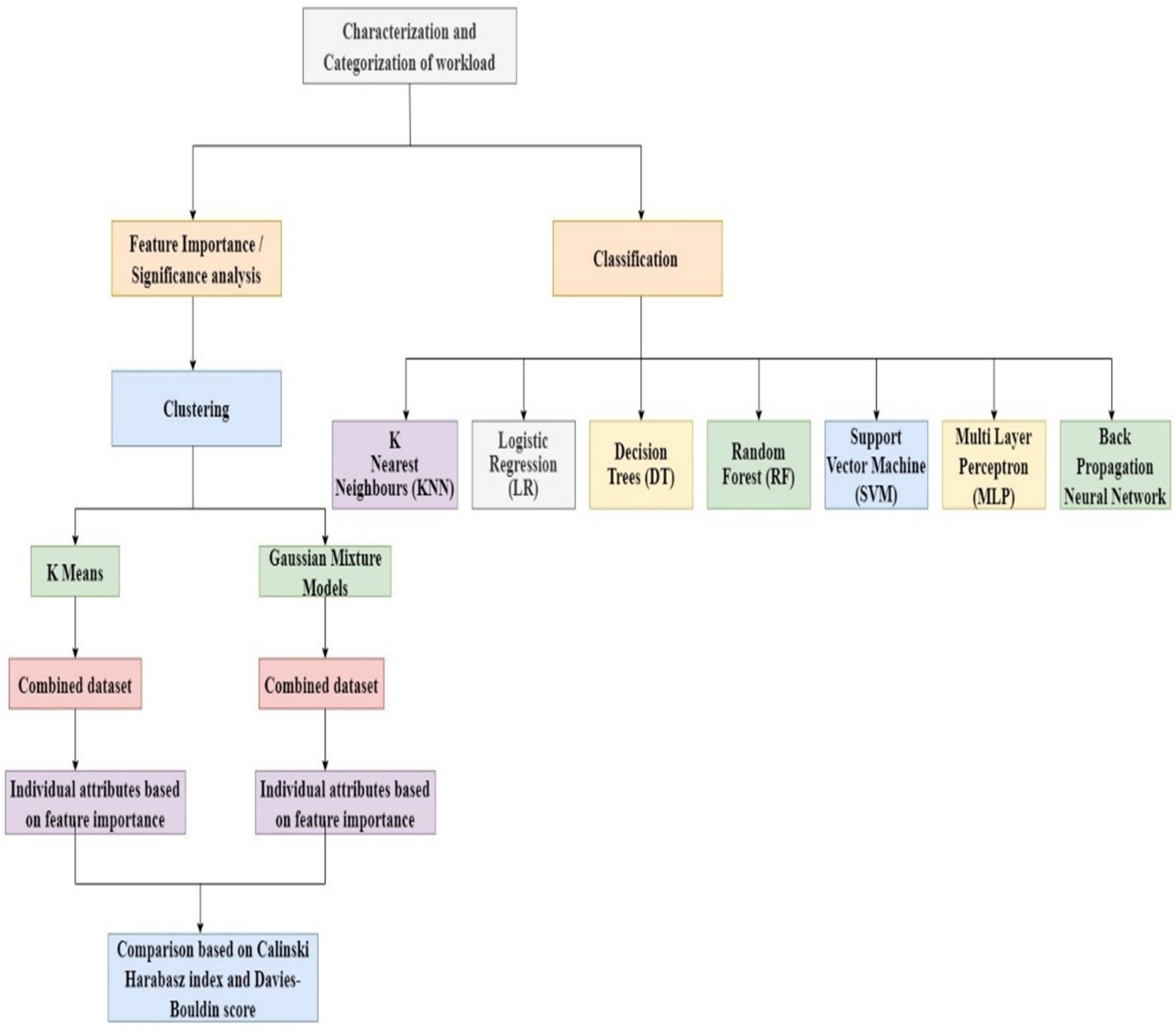

Characterization and Categorization of workload
Feature Importance / Significance analysis
Classification
Clustering
K Nearest Neighbours (KNN)
Logistic Regression (LR)
Decision Trees (DT)
Random Forest (RF)
Support Vector Machine (SVM)
Multi Layer Perceptron (MLP)
Back Propagation Neural Network
K Means
Gaussian Mixture Models
Combined dataset
Combined dataset
Individual attributes based on feature importance
Individual attributes based on feature importance
Comparison based on Calinski Harabasz index and Davies-Bouldin score

Figure 3.2: Methodology for categorization and characterization for workload

3.4 Experimental Setup

Normalization of the BBT dataset is done using min-max normalization. The K means technique was applied to the combined data set consisting of attributes having high significance. The technique was further applied to each attribute to calculate K. The value of number of clusters, K is determined using elbow criterion. If c represents the clusters obtained for CPU usage [MHz], m for Memory usage [KB], d for Disk usage [KB/s] and n represents clusters acquired for Network usage [KB/s]. The product of c, m, d, and n yields the number of possible workloads in the dataset. As a result, the frequency of various workloads is calculated. This analysis yielded the dataset's workload distribution. The experiments have been conducted on M1-3.2GHz processor and 16 GB RAM. GMM clustering is applied to all attributes first, followed by individual attributes. The outcomes of both algorithms are compared using parameters such as the Calinski Harabasz index(CHI) and the Davies-Bouldin Index (DBI). CHI is also known as the Variance Ratio Criterion. After characterizing and categorizing the workload, an experiment was conducted to create the appropriate sized virtual machines. This experiment aimed to determine whether creating VMs based on workload distribution could result in more efficient resource utilization and workload management. Following the characterization of the dataset, classification is done using different classification algorithms K Nearest Neighbours (KNN), Logistic Regression (LR), Decision Trees (DT), Random Forest (RF), Support Vector Machine (SVM), Multi-Layer Perceptron and Back Propagation Neural Network. The classification of workloads is an important step in workload analysis. The accuracy of the models built is crucial for workload analysis, resource usage prediction and provisioning. The classification accuracy analysis aids us in determining which algorithm is best for a given data center workload.

3.5 Results

The findings of experiments conducted are reported in this section. Firstly, the feature importance or attribute significance analysis is done using a decision tree algorithm. The results are as shown in Figure 3.3. The attributes are represented on the x-axis, while the value of coefficients is depicted on the y axis. It is clear that the importance of CPU usage [MHz] is highest for the BBT dataset. The network and disk usage dominates the second and third position in terms of significance analysis. Memory usage has low significance among all the attributes.

Elbow method to determine the value of K is applied and the graph is plotted between K and inertia. WCSS (Within-Cluster Sum of Squares) or inertia signifies the summation of squared distances between each data

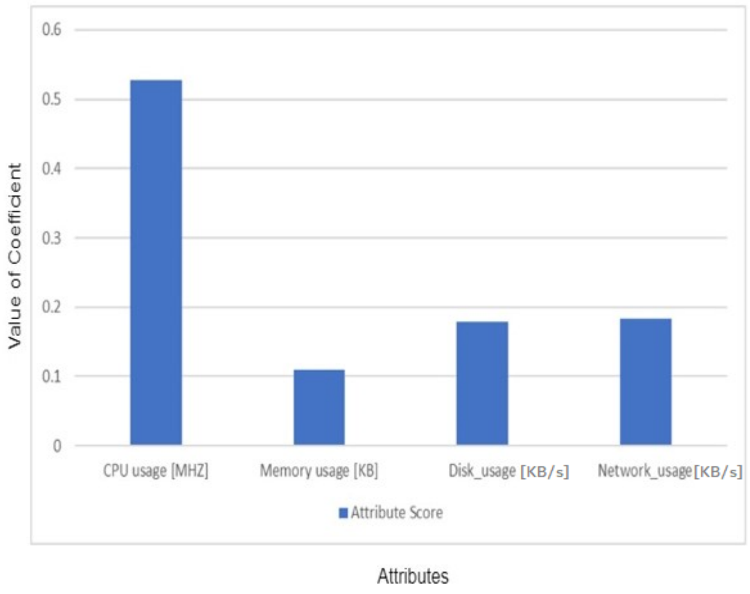


Figure 3.3: Attribute score of different attributes

point and the centroid of its respective cluster. It gauges how tightly the clusters are formed or how compact they are. A lower WCSS value indicates more uniform and homogeneous clusters, reflecting superior clustering performance. The graph is shown in Figure 3.4. The value of x-axis depicts the K values and the value of the Y-axis depicts the value of inertia. At K = 4, it produces an elbow, indicating that the BBT fastStorage dataset has four different types of workloads. The execution time for K Means on all attributes combined is 4.1s.

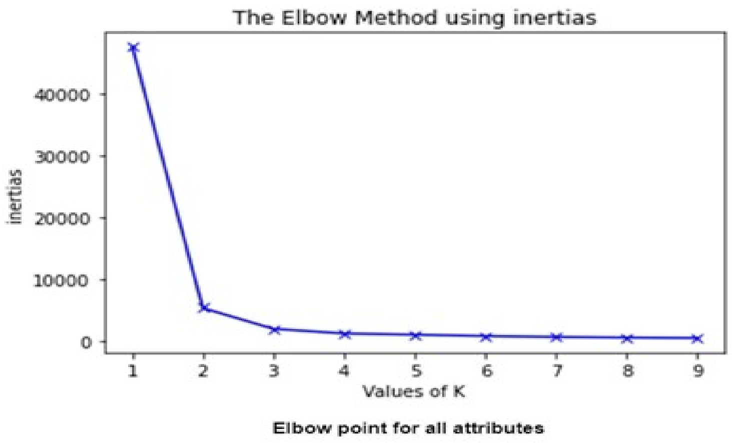


Figure 3.4: Value of K vs inertias on high significant attributes

The plots of the clustering results based on each attribute individually, such as CPU usage, memory usage, disk usage, and network usage are depicted in Figure 3.5, Figure 3.6, Figure 3.7 and Figure 3.8 respectively. It can be concluded from these graphs that for CPU usage two different workloads have been identified that is CPU LOW (C_L) and CPU HIGH (C_H); for memory, three different workloads have been observed that is Memory LOW(M_L) , Memory MEDIUM (M_M) and Memory HIGH (M_H); 2 workloads have been identified for disk usage that is Disk LOW (D_L) and Disk HIGH (D_H) and 2 workloads have been identified for network usage that is Network LOW (N_L) and Network HIGH (N_H).

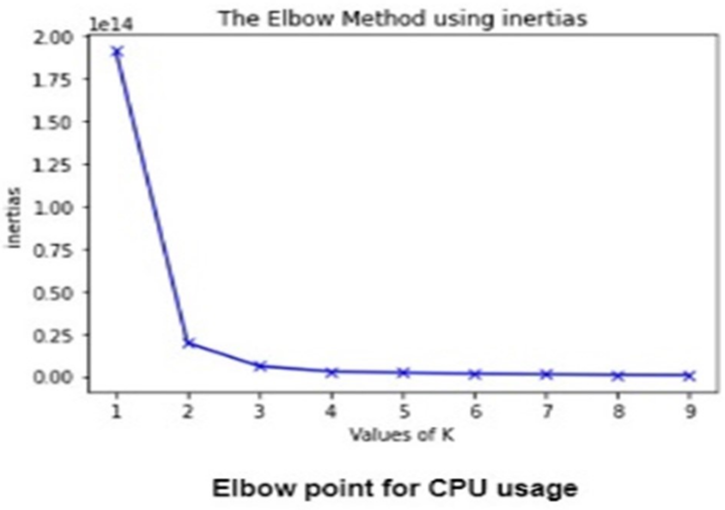


Figure 3.5: K vs Inertia (CPU usage)

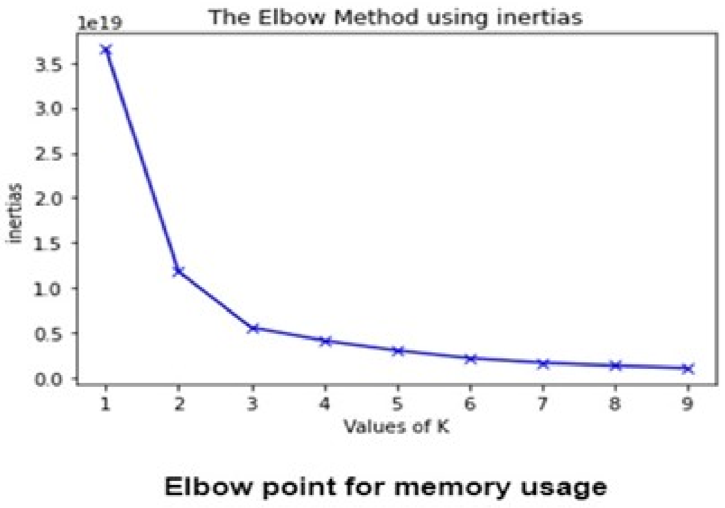


Figure 3.6: K vs Inertia (Memory usage)

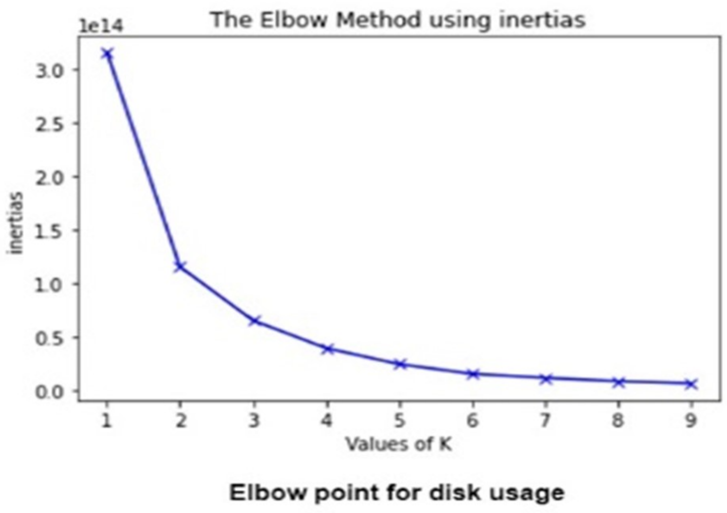


Figure 3.7: K vs Inertia (Disk usage)

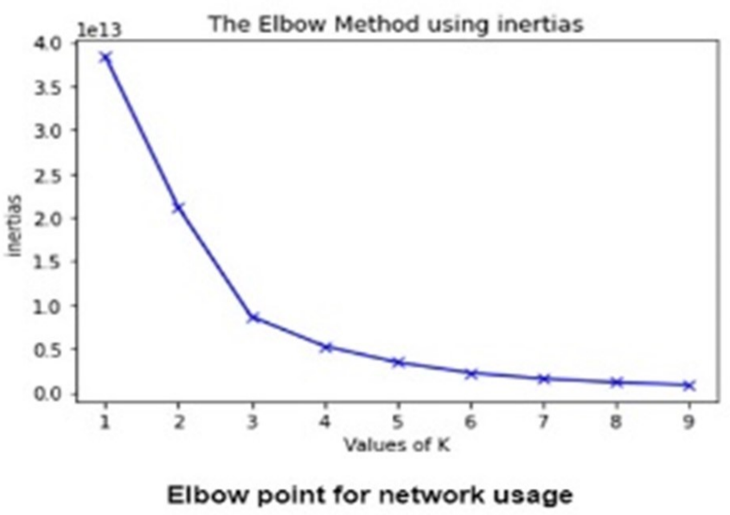


Figure 3.8: K vs Inertia (Network usage)

The 24 distinct workload combinations are determined once the elbow point is calculated using K Means clustering. The percentage of tasks can be identified by calculating the number of tasks in each combination. Depending on the usage of the resources, the workloads have been characterized as High, Medium and Low as per the resource used. Table 3.2 is formatted as follows:[CPU usage][Memory usage][Disk usage][Network usage]. Depending on the usage of the resources(R), the workloads have been characterized as Low (R_L), Medium (R_M), High (R_H). R represents resources like CPU (C), Memory (M), Network (N) and Disk (D). For instance, [C_L] [M_L] [D_H] [N_L] represents CPU low, memory low, Disk medium and network low resource consumption.

Table 3.2 shows that the majority of tasks (93.38 percent) have modest resource utilization. These processes used less CPU, memory, storage space, and network bandwidth. These virtual machines are made up of short administrative chores and application inquiries. The workload then consists of 3.41 percent of jobs that employ a high CPU, medium memory, and low disk and network use. Typically, these virtual machines are utilized to run CPU-intensive consumer applications.

Similarly, GMM clustering is applied on the BBT dataset. The graphs of Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) are plotted initially on the entire dataset. It can be concluded from the graph that there are 7 different types of workload in the entire dataset.

Table 3.2: Percentage of tasks in each cluster (K Means)

Type of workload	**Number of tasks**	**Percentage of tasks**
$[C_L]$ $[M_L]$ $[D_L]$ $[N_L]$	10479078	93.38
$[C_L]$ $[M_L]$ $[D_L]$ $[N_H]$	6242	0.05
$[C_L]$ $[M_L]$ $[D_H]$ $[N_L]$	10823	0.09
$[C_L]$ $[M_L]$ $[D_H]$ $[N_H]$	3150	0.02
$[C_L]$ $[M_M]$ $[D_L]$ $[N_L]$	127939	1.14
$[C_L]$ $[M_M]$ $[D_L]$ $[N_H]$	1492	0.01
$[C_L]$ $[M_M]$ $[D_H]$ $[N_L]$	1823	0.01
$[C_L]$ $[M_M]$ $[D_H]$ $[N_H]$	401	0
$[C_L]$ $[M_H]$ $[D_L]$ $[N_L]$	28650	0.25
$[C_L]$ $[M_H]$ $[D_L]$ $[N_H]$	147	0
$[C_L]$ $[M_H]$ $[D_H]$ $[N_L]$	3873	0.03
$[C_L]$ $[M_H]$ $[D_H]$ $[N_H]$	9	0.00008
$[C_H]$ $[M_L]$ $[D_L]$ $[N_L]$	88450	0.78
$[C_H]$ $[M_L]$ $[D_L]$ $[N_H]$	74	0.0006
$[C_H]$ $[M_L]$ $[D_H]$ $[N_L]$	4	0.00003
$[C_H]$ $[M_L]$ $[D_H]$ $[N_H]$	59	0.0005
$[C_H]$ $[M_M]$ $[D_L]$ $[N_L]$	382850	3.41
$[C_H]$ $[M_M]$ $[D_L]$ $[N_H]$	50	0.0004
$[C_H]$ $[M_M]$ $[D_H]$ $[N_L]$	90	0.0008
$[C_H]$ $[M_M]$ $[D_H]$ $[N_H]$	6	0.00005
$[C_H]$ $[M_H]$ $[D_L]$ $[N_L]$	83166	0.74
$[C_H]$ $[M_H]$ $[D_L]$ $[N_H]$	14	0.0001
$[C_H]$ $[M_H]$ $[D_H]$ $[N_L]$	3406	0.03
$[C_H]$ $[M_H]$ $[D_H]$ $[N_H]$	3	0.00002

Figure 3.9 shows the AIC and BIC plot. On the x-axis, the number of clusters or components are depicted whereas on Y-axis the score of AIC and BIC is depicted. The value of AIC and BIC is same for each analysis therefore, both the lines overlap each other that results in depiction of one line in all graphs. The execution time for GMM on all attributes combined is 8.1s.

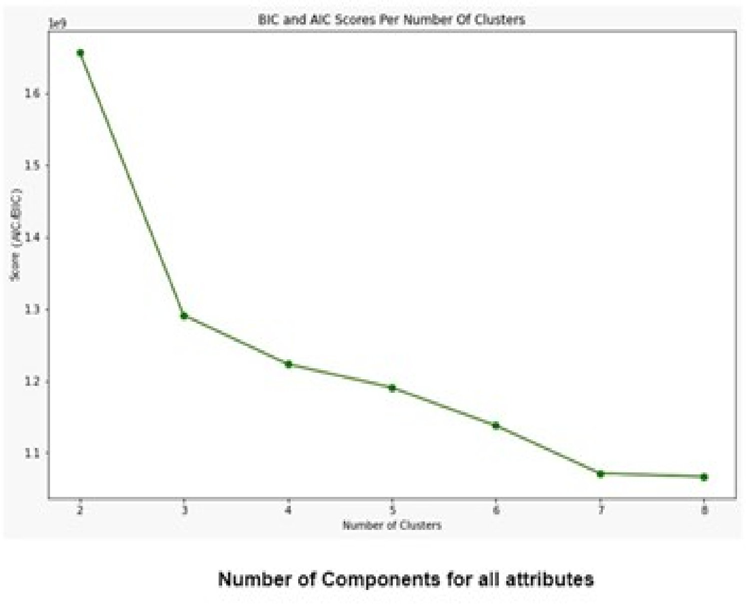


Figure 3.9: AIC and BIC plot for all attributes

Once the clustering is done on the combined dataset, the GMM is applied on the individual attributes. Clustering based on individual attributes, CPU usage, memory usage, disk usage, and network usage yielded the results as shown in figure 3.10, figure 3.11, figure 3.12 and figure 3.13 respectively. From the given graphs, it can be concluded that CPU and memory usage have five different types of workloads followed by disk and network usage having 4 different types of workloads.

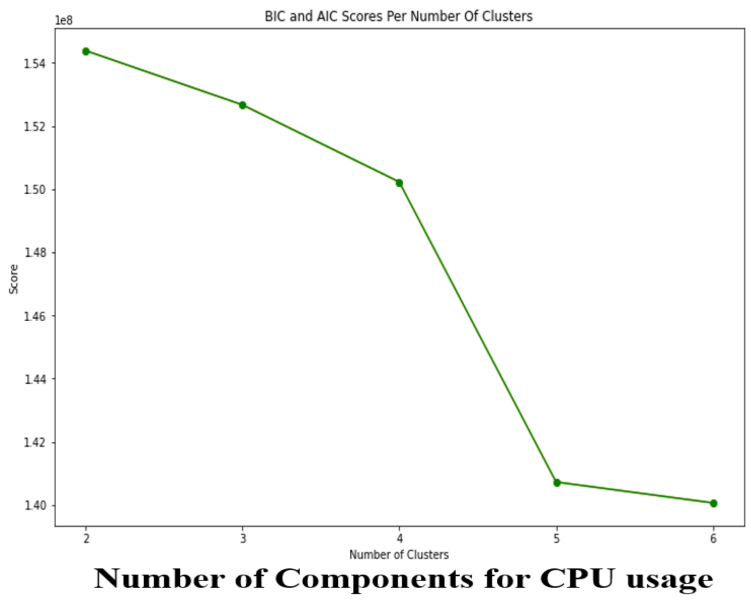


Figure 3.10: AIC and BIC plot (CPU usage)

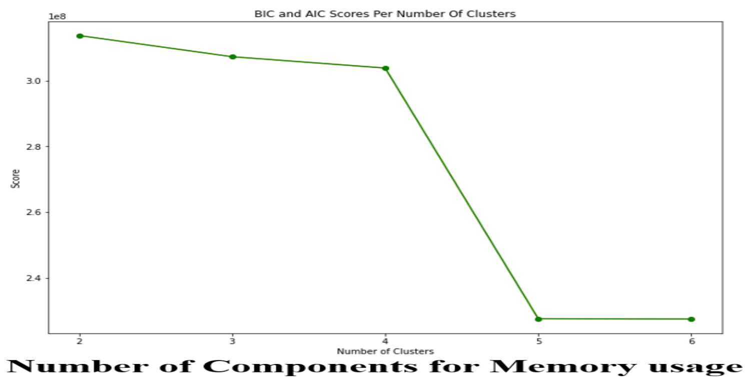


Figure 3.11: AIC and BIC plot (Memory usage)

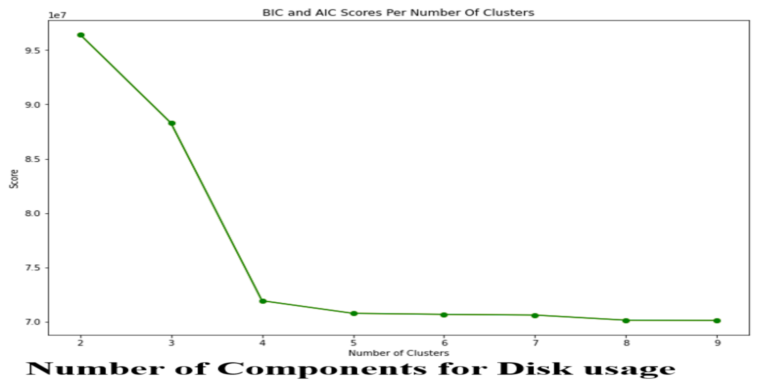


Figure 3.12: AIC and BIC plot (Disk usage)

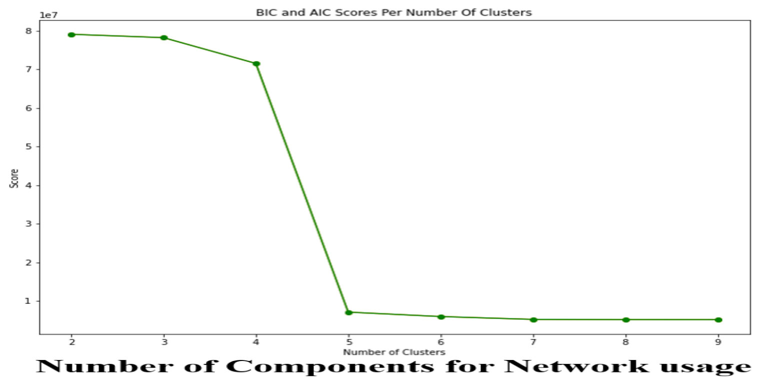


Figure 3.13: AIC and BIC plot (Network usage)

Once the number of components are identified for each attribute, 400 different workload combinations are formed. There are some combinations in all clusters that are not identified due to thir sparse points, therefore the 336 combinations are taken, which consists of workloads from all four attributes. Table 3.3 depicts the various clusters formed by the GMM. Depending on the usage of the resources(R), the workloads have been characterized as Very Low (R_{VL}), Low (R_L), Medium (R_M), High (R_H) and Very

High (R_{VH}). R represents CPU (C), Memory (M), Network (N) and Disk (D). For instance, [C_{VL}] [D_{VL}] [N_H] [M_{VL}] represents CPU very low, disk very low, network high and memory very low resource consumption.

Table 3.3: Percentage of tasks in each cluster (GMM)

Types of workloads (CPU, DISK, NETWORK, MEMORY)	**Number of tasks**	**Percentage of tasks**
[C_{VL}] [D_{VL}] [N_{VL}] [M_{VL}]	1694756	15.10235435
[C_{VL}] [D_{VL}] [N_{VL}] [M_{VH}]	1864489	16.61488353
[C_{VL}] [D_{VL}] [N_L] [M_{VH}]	144998	1.29211
[C_{VL}] [D_{VL}] [N_H] [M_{VL}]	90361	0.805227325
[C_{VL}] [D_{VL}] [N_H] [M_{VH}]	777464	6.928157693
[C_{VL}] [D_{VH}] [N_{VL}] [M_{VH}]	309871	2.761330624
[C_{VL}] [D_{VH}] [N_H] [M_M]	61191	0.545286852
[C_{VL}] [D_{VH}] [N_H] [M_{VH}]	477556	4.255609617
[C_L] [D_{VL}] [N_H] [M_H]	140988	1.256375982
[C_M] [D_L] [N_L] [M_M]	69007	0.614936998
[C_M] [D_L] [N_L] [M_H]	70439	0.627697874
[C_M] [D_{VH}] [N_H] [M_M]	55656	0.495963214
[C_H] [D_{VL}] [N_{VL}] [M_{VH}]	150892	1.344632768
[C_H] [D_{VL}] [N_H] [M_{VL}]	194811	1.736004919
[C_H] [D_{VL}] [N_H] [M_M]	72132	0.64278458
[C_H] [D_{VL}] [N_H] [M_{VH}]	214474	1.911226363
[C_H] [D_L] [N_L] [M_M]	102409	0.912589781
[C_H] [D_L] [N_H] [M_M]	184342	1.64271329
[C_H] [D_L] [N_H] [M_{VH}]	73562	0.655527634
[C_H] [D_{VH}] [N_{VL}] [M_{VL}]	67194	0.598780944
[C_H] [D_{VH}] [N_{VL}] [M_M]	116116	1.03473596
[C_H] [D_{VH}] [N_{VL}] [M_{VH}]	395369	3.523222656
[C_H] [D_{VH}] [N_L] [M_M]	92185	0.821481402
[C_H] [D_{VH}] [N_H] [M_{VL}]	151433	1.349453742
[C_H] [D_{VH}] [N_H] [M_M]	633867	5.648532321
[C_H] [D_{VH}] [N_H] [M_{VH}]	1114687	9.933228181
[C_{VH}] [D_{VL}] [N_H] [M_H]	71513	0.637268531

Both K means and GMM performance is evaluated based on Calinski Harabasz index, Davies-Bouldin score and execution time. Table 3.4 shows the values of different parameters by applying K Means and GMM.

From Table 3.4, it can be concluded that K means outperforms the

Table 3.4: Scores for performance evaluation parameters

Parameter	**K Means**	**Gaussian Mixture Models**
Execution Time (in seconds)	4.1s	8.1s
Calinski Harabasz Index	54500063	12027922.5
Davies-Bouldin score	0.37	3.77

GMM on all mentioned parameters. The Davies-Bouldin score of K Means is 0.37 that is much better than that of 3.77 of GMM. Closer the value to zero, better the cluster separation. The Calinski Harabasz Index of K means is 54500063 whereas for GMM it is 12027922.5. Greater the value of Calinski Harabasz Index score better is the density and cluster separation. After characterizing and categorizing the workload traces, an experiment was conducted that aimed to investigate whether workload distribution could be used to create virtual machines of appropriate sizes. The sample data from K Means clusters was utilized as input for VM sizing. VMs of different sizes were created based on the characteristics of the workload and arranged into clusters. The workload was mapped to different VM clusters, and the results were compared to those obtained by mapping the workload to random sized VMs created. To evaluate the efficiency of this approach for workload management and resource utilization, the number of VMs were calculated. According to the experimental results, there was a notable difference in the number of virtual machines (VMs) used in VM clusters created using K-means clustered workload data i.e.19 in comparison to the number of randomly sized VMs created i.e. 26. An experimental assessment to approximate classification accuracy for various machine-learning algorithms to explore how workload distribution affects each model has been done. The analysis was carried out using BBT fast-Storage dataset. The ratio of training and testing data is kept as 70 and 30 respectively. Table 3.5 displays the accuracy results, AUC ROC score, Precision and execution time for different algorithms.

Table 3.5: Accuracy percentage of classification algorithms

Algorithms	Accuracy in %	AUC ROC Score	Precision	Execution Time (s)
K Nearest Neighbours (KNN)	98.79	0.963	0.96	448
Logistic Regression (LR)	79.19	0.941	0.93	182
Decision Trees (DT)	99.18	0.976	0.97	165
Random Forest (RF)	97.8	0.958	0.95	190
Support Vector Machine (SVM)	84.34	0.922	0.91	172844
Multi-Layer Perceptron	79.72	0.825	0.82	200
Back Propagation Neural Network	80	0.847	0.84	250

3.6 Summary

In the realm of cloud data centers, workload characterization and categorization emerge as crucial practices that facilitate effective resource management and optimization. Workload characterization involves a meticulous examination of the diverse workloads present within the data center environment. This analysis encompasses factors such as resource consumption patterns, performance metrics, and utilization behaviors, shedding light on how different workloads interact with available resources. Simultaneously, workload categorization entails the classification of these heterogeneous workloads into coherent groups based on shared attributes and behaviors. By clustering similar workloads together, data center administrators can tailor resource allocation strategies to meet the specific demands of each category. This alignment between workload types and resource provisioning leads to enhanced operational efficiency and performance. The combined effects of workload characterization and categorization are far-reaching. They empower data centers to make informed decisions about resource allocation, capacity planning, and load balancing. Furthermore, such insights enable the implementation of dynamic scaling mechanisms to accommodate fluctuating workloads effectively. Ultimately, these practices contribute to optimized resource utilization, reduced operational costs, improved energy efficiency, and an overall elevated quality of service in the realm of cloud data centers.

The research delves into the pivotal role of categorizing and characterizing workloads in cloud data centers. It encompasses the clustering of diverse workload types employing two distinct clustering techniques. The allocation of workloads is achieved through a fusion of unique workload combinations in both clustering modes. Following the completion of clustering, a performance evaluation is conducted to ascertain the superior clustering approach. In accordance with the K Means algorithm, a significant proportion (93.38%) of tasks exhibit low resource utilization across CPU, memory, storage, and network bandwidth. These encompass virtual machines engaged in brief administrative tasks and application inquiries. Conversely, the Gaussian Mixture Model (GMM) identifies a maximum of 16.61% of tasks that exhibit very low utilization of CPU, Disk, Network, and very high utilization of Memory resources. The outcomes underscore K Means' superiority in terms of Calinski Harabasz Index, Davies-Bouldin Index and execution time. Subsequent to the clustering process, classification is implemented using diverse classification techniques. Notably, the decision tree technique achieves the highest accuracy, reaching an impressive 99.18%.

Chapter 4

Container Placement in Cloud Data Center

In this chapter, various container placement algorithms are explored, emphasizing their role in enhancing energy efficiency within cloud data centers. It highlights the implementation and impact of these algorithms while shedding light on the efficacy of proposed and implemented metaheuristic solutions in effectively reducing data center energy consumption.

4.1 Introduction

Better resource management is required to improve resource utilisation in Data Center (DCs). Energy efficient management techniques utilize resources efficiently while increasing energy efficiency (Dayarathna et al., 2016). VM placement aims to improve resource utilisation by deploying many VMs on a single Physical Machine (PM) (Corradi et al., 2014) (Ammar et al., 2019). Similarly, container placement techniques entail assigning several containers to a single VM. These VM and container placement approaches have the potential to enhance resource utilization for both physical machines (PMs) and virtual machines (VMs), thereby reducing energy consumption in cloud data centers. A Container Placement (CP) technique aims to distribute containers in the most efficient manner possible, utilizing the fewest number of VMs and active PMs, thereby increasing resource efficiency and lowering energy consumption in data centers. The container placement work is divided into two segments. The cloud provider must first assess new container licensing applications and install the containers on the hosted VMs before hosting the VMs on the PMs. Second, during the container/VM migration process, the cloud infrastructure optimises container and VM mobility to minimize energy usage, enhance resource usage, and so on. Better initial container and VM placement improves the work required for second stage optimization (Masdari et al., 2016). Some studies have reduced CP to a single level, mostly by doing container assignment to PM (Kaur et al., 2017). The disadvantage is that VM placement does not take into account VM-container arrangements, which limits the circumstances in which all containers may be co-located. As a result, the problem of container mapping to VMs is approached differently from the problem of mapping VMs to PMs. Container placement is similar to bin packing in that virtual machines and containers function as a multidimensional vector, with each column representing a distinct type of resource such as CPU, memory capacity, and network bandwidth. Because there is no difference in bin packing whether objects are organised opposite or on top of one another; nevertheless, this strategy is not accurate and ideal for container placement challenges because each item is a container that must be put in a VM. When a container utilises a resource, no other containers can use it at the same time. A good solution is especially beneficial to cloud providers because it enables them to take greater control of their computing resources while maintaining customer QoS as long as the algorithm distributes a group of containers over several physical servers. It is an NP-hard combinatorial issue to allocate containers to VMs and VMs to PMs (Al-Moalmi et al., 2021). The problem can be resolved in order to minimise energy consumption in a cloud DC. There are numerous approaches to address this issue, one of which is Evolutionary Computing (EC), which is utilised to lower energy consumption while increasing resource utilisation. Efficient container placement and consolidation pose significant challenges in optimizing resource utilization. Incorrect placement may lead to imbalances in workload distribution across servers, resulting in either underutilized or overutilized resources. The dynamic nature of cloud workloads

further complicates the scenario, requiring adaptive placement strategies to address fluctuations in demand and ensure effective resource allocation and scaling. Achieving energy efficiency is paramount, as suboptimal container placement may trigger unnecessary server activations and inefficient resource utilization, contributing to heightened energy consumption. Ensuring fault tolerance and high availability introduces complexities in container placement. Strategies must consider redundancy and isolation to minimize the impact of failures, contributing to a resilient system. Balancing these considerations is crucial for effective container placement and consolidation in cloud environments, emphasizing the need for adaptive strategies to navigate the dynamic nature of workloads and optimize resource utilization.
Metaheuristic algorithms, including genetic algorithms, simulated annealing, and particle swarm optimization, offer effective solutions for container placement and consolidation challenges in resource utilization (Kalra and Singh, 2005). These algorithms efficiently navigate the solution space, identifying near-optimal container placements to address workload distribution issues. Their inherent adaptability enables dynamic adjustments based on changing workloads, effectively tackling the unpredictability of resource demands. Additionally, metaheuristics can incorporate energy efficiency objectives, optimizing container placements to minimize energy consumption and contribute to sustainability in cloud data centers. Furthermore, metaheuristic algorithms can be tailored to consider fault tolerance requirements. They explore redundant placement strategies to ensure high availability and resilience to failures, enhancing the robustness of container placement and consolidation. In summary, metaheuristic algorithms provide adaptive and efficient solutions for optimizing container placement and consolidation in the dynamic and complex environment of cloud computing.

4.2 Literature Review

Despite the growing popularity of container technology as a cloud service paradigm, there have been few attempts to minimize usage of energy in Container as a Service (CaaS) DCs. Many attempts have been made to resolve the container placement problems. In a study by Nardelli et al. (2017), they approached this task as an Integer Linear Programming problem, considering the diversity of resource types among containers and VMs. Their technique not only aimed at enhancing various Quality of Service (QoS) measures but also had the ability to reallocate containers during runtime if it could lead to improved QoS. To establish a baseline, they compared their proposed formulation with two widely used heuristics: the greedy first-fit and round-robin methods, typically employed to address the container placement problem.

Similarly, Boukadi et al. (2017) tackled the resource allocation problem from an enterprise perspective. They introduced a linear program (LP) designed to determine the optimal deployment of a business process on cloud containers. Their method was evaluated against both a VM-based scheduling scheme and the First-Fit method.

In another approach, Smimite and Afdel (2020) presented a Hybrid solution for resource and workload management. They combined ant colony optimization (ACO) with the first-fit decreasing (FFD) algorithm to mitigate unnecessary use of energy.

Furthermore, some efforts have explored the consolidation of containers as an integrated optimization problem, considering both container placement and VM placement simultaneously. Mann (Mann, 2018), for example, gave a solution to the joint optimization issue of dynamic container aggregation for a multi-task learning tracing. They took into account size issues, colocation restrictions, license fees, and hardware affinity relationships. Based on an actual review of a real-world workload trace, the integrated solution delivered much significantly better outcomes than studying the two concerns independently. Following a similar approach, Shi et al. (Shi et al., 2018a) introduced a Two-stage Multi-type Particle Swarm Optimization technique, referred to as TMPSO, for energy-efficient container consolidation in cloud data centers. Through experimental evaluations conducted on benchmark datasets, they were able to demonstrate that their proposed algorithm achieved energy savings when compared to certain existing methods. Tan, Ma, and Mei (Tan et al., 2019) approached the problem in two steps. To solve the container placement problem, the authors combined two approaches to develop a genetic programming hyper-heuristic with customized rules. Experiments demonstrated that their hybrid method considerably cut energy usage when compared to utilizing only human-designed rules. Apart from the above-mentioned research papers, nowadays researchers have increased the use of different metaheuristic algorithms for the placement of containers and VMs. Meta-heuristic algorithms, in general, have been used to identify optimum solutions to large-scale computing issues and have grown in popularity because they are less difficult and more efficient than other older procedures (M. K. Patra et al., 2022). Meta-heuristic algorithms are crafted by drawing inspiration from animal and insect behaviors, natural phenomena, evolutionary principles, and other naturally occurring concepts. These algorithms can be categorized into five distinct types: bio-stimulated, nature-inspired, physics-based, evolutionary, and swarm-based. Bio-stimulated meta-heuristic optimization is based on wild animal and marine creature hunting and foraging habits. Nature-inspired optimization algorithms are the second type of meta-heuristic that are based on natural systems. Physical laws usually motivate physics-based meta-heuristic techniques. The fourth category, evolutionary type is built on the notions of the natural selection process. In this approach, the community tries to build on the fitness function measurements in their environment and puts forth every effort to locate the ideal outcome in the search spaces. The last category of meta-heuristic algorithm i.e., swarm-based is based on the mutual behaviour of insects, nanoparticles, and other social animals. Meta-heuristics are well suited to combinatorial optimization issues. While they are not always guaranteed to discover the best global solution, they may frequently find an excellent solution in a reasonable period. They are an alternative to exhaustive search, which would take exponential time. In order to avoid local minima, meta-heuristics frequently integrate some type of randomization (Saber et al., 2018). Hussein and colleagues (Hussein et al., 2019) proposed an architecture for placing containers on virtual machines (VMs). Their design attempts to maximise VM and physical machine (PM) use while minimising the number of active VMs and PMs. They investigated placement algorithms like the Best Fit and Max Fit heuristics, as well as a fitness function that assesses resource use. In addition, they combined their fitness function with a meta-heuristic termed Ant Colony Optimisation based on Best Fit (ACO-BF).

Similarly, Farzai (Farzai et al., 2020) framed the VM placement problem as a multi-objective optimization challenge, with the objectives of reducing resource wastage, minimizing power consumption, and reducing bandwidth usage within the network. They presented an ant colony optimization technique as a solution to this defined problem.

Furthermore, Shabeera and colleagues (Shabeera et al., 2017) introduced an ACO-based Virtual Machine Placement (VMP) approach for locating data and VMs using constrained PMs. This method selects PMs in close proximity, and the workloads are distributed over the VMs allocated by this scheme, which outperforms other allocation strategies.

In a similar vein, Bouaouda (Bouaouda et al., 2022) conducted a comparison of container placement strategies, specifically the First Fit Decreasing algorithm and the Ant Colony Optimization algorithm. They compared these two approaches based on energy consumption. Zhang (W. Zhang et al., 2022) provided a two-stage container management approach for minimising migration costs and load balancing. To handle optimization issues at diverse time scales, they devised adaptive and greedy algorithms. Their suggested methods are tested using real-world trace data from Alibaba, demonstrating that they beat the alternative approach in terms of load balancing and migration cost. The authors in (Akindele et al., 2022) proposed a new fixed-length crossover operator for Grouping Genetic Algorithm (GGA) container resource allocation optimization to promote population variety and exploration. They also suggested problem-specific Best-Fit and Largest VM heuristic operators to enhance local search by reordering containers from the chromosomal tail's lower fitness PMs into existing VMs and PMs with higher usage where possible. They showed that the suggested GGA may dramatically cut energy usage in large-scale test situations using the newly built operators. Bouaouda (Bouaouda et al.,

2023) offered a method for predicting cloud container location and power usage in data centres utilising heuristics and meta-heuristics like Genetic Algorithm (GA) and First Fit Decreasing (FFD). They conducted evaluations of their algorithms using CloudSim, and the results consistently showed that their approaches, particularly the Genetic Algorithm, outperformed Ant Colony Optimization (ACO) and Simulated Annealing (SA), providing more effective and efficient solutions.

Additionally, in their study, they introduced an algorithm based on the Whale Optimization Algorithm (WOA) (Al-Moalmi et al., 2021) to tackle both stages of placement as a unified optimization problem. The algorithm's primary objective was to identify the optimal configuration of Virtual Machines (VMs) and Physical Machines (PMs) within a single search space. Extensive testing was conducted across diverse heterogeneous environments, with comparisons made to recent approaches. The experimental results consistently demonstrated the superiority of their proposed method over the alternatives in a variety of test environments.
Table 4.1 shows the comparison of work done in the domain of container placement in CDC.

Table 4.1: Comparison of work done in the domain of container placement in cloud data centers

Work	**Advantages**	**Disadvantages**
An Integer Linear Programming issue is used to generalise the elastic provisioning of virtual machines for container deployment (Nardelli et al., 2017).	Their technique took into consideration the diversity of container and VM resource types.	They did not consider using AI based metaheuristics algorithms for efficient container placement.

Focused on the resource distribution issue from an enterprise standpoint and proposes a linear program (LP) for determining the best installation of a business processes on cloud containers (Boukadi et al., 2017).	The authors compared their suggested technique to the First-Fit method for containers and a VM-based deployment.	They did not consider testing their proposed technique with large number of VMs and PMs.
To reduce needless power usage, (Smimite and Afdel, 2020) proposed a Hybrid strategy for managing resources and workload based on Ant Colony Optimisation (ACO) and the first-fit decreasing (FFD) algorithm.	The experiment findings showed that employing the FFD for container placement outperformed ant colony optimization, especially in homogenous environments. In the context of workload management, however, ACO produced quite satisfying solutions.	They did not track under-utilized hosts and tested the proposed approaches for a smaller number of containers and hosts.
Gave a solution to the joint optimization issue of dynamic container aggregation for a multi-task learning tracing (Mann, 2018).	Took into account size issues, colocation restrictions, license fees, and hardware affinity relationships.	The proposed approaches were not compared with other pre-existing approaches.

Proposed (Shi et al., 2018a) a Two-stage Multi-type Particle Swarm Optimization approach, named TMPSO, to energy-aware container consolidation in Cloud data centers.	When compared to certain current techniques, the proposed algorithm saved much more energy.	They did not consider container migrations to realize the dynamic optimization of container consolidation.
To answer the challenge of container placement, the authors created a hybrid solution that combined genetic programming hyper-heuristics with human-designed criteria (Tan et al., 2019).	The hybrid technique could dramatically reduce energy usage when compared to employing only human-designed rules.	They did not consider testing proposed algorithm in the heterogenous environment.
Suggested a container placement architecture for VMs. The suggested design seeked to maximize VM and PM usage while keeping the number of deployed VMs and active PMs to a minimum (Hussein et al., 2019).	They investigated the Best Fit and Max Fit heuristics, as well as a fitness function that assesses resource consumption.	They did not mention the details about the QoS.

Framed the problem of VMP as a multi-objective optimization problem (Farzai et al., 2020).	Multi-objective optimization model not only focused on energy savings; however, it was also concerned with reducing resource waste and lowering the high rate of data transmission over the shared network bandwidth of DCs.	They did not focus on the real-world problem of container placement.
Presented (Shabeera et al., 2017) an ACO-dependent Virtual Machine Placement (VMP) method for locating data and VMs using restricted PMs	Selects PMs in close vicinity, and the workloads were distributed over the VMs allocated by such a scheme, outperforming other allocation strategies.	They did not consider implementing the proposed algorithm in the heterogenous environment. They did not focus on the real-world problem of container placement also.
Presented (Bouaouda et al., 2022) the comparison of container placement strategies like First Fit Decreasing algorithm and Ant Colony Optimization algorithm	They compared the proposed strategy on the basis of the energy consumption.	The major drawback is that they did not consider SLA violations and also tested the approaches for smaller number of hosts.

Provided a two-stage container management approach for minimising migration costs and load balancing (W. Zhang et al., 2022).	To handle the optimisation difficulties for container placement and container migration at different time scales, they introduced the Balance Aware Container Placement (BACP) method and the Adaptive Threshold Container Migration (ATCM) algorithm.	They tested the algorithms on a small number of servers and containers.
Proposed (Akindele et al., 2022) a new fixed-length crossover operator for GGA container resource allocation optimization.	They showed that their algorithm reduced the energy consumption in large scale test cases.	Not Mentioned
Predicted cloud container location and power usage in data centres utilising heuristics and meta-heuristics using a Genetic Algorithm (GA) and First Fit Decreasing (FFD) (Bouaouda et al., 2023).	Their proposed strategies provide better and efficient solutions.	They tested the strategies with a smaller number of hosts.

Addresses (Al-Moalmi et al. 2021) the problem of container and VM placement in CaaS systems while optimising both power consumption and resource utilisation.	In one search space, the suggested method looked for the optimal number of VMs and PMs. The suggested approach is compared to current methods in various degrees of diverse situations.	Not mentioned.

The techniques mentioned above suffer from various limitations that render them incomplete or unsuitable for the current scenario as the container placement problem is NP hard.
To demonstrate the NP-hardness of the container placement problem using 3SAT, it is necessary to show a reduction from 3SAT to the container placement on virtual machine. The task is to prove that an instance of 3SAT can be transformed into an instance of the container placement on virtual machine in polynomial time.

The Container Placement on Virtual Machine:
The container placement on virtual machine problem involves placing a set of containers of different sizes onto a set of virtual machines with fixed capacities, such that the number of virtual machines used is minimized, and the containers' total sizes do not exceed the capacities of the assigned virtual machines.
Formally, given a set of containers with sizes C = $c_1, c_2, ..., c_n$ and a set of virtual machines with capacities V = $v_1, v_2, ..., v_m$, the task is to find a placement of containers onto virtual machines such that:
1. The total size of containers assigned to each virtual machine does not exceed its capacity.
2. The number of used virtual machines is minimized.
Proof
Given an instance of 3SAT with n variables and m clauses, it is necessary

to construct an instance of the container placement on virtual machine.

1. For each variable x_i in the 3SAT instance, create two containers with sizes $ci_1 = 2$ and $ci_2 = 1$. These two containers represent the truth values "true" and "false" for the variable x_i.

2. For each clause C_j = (a, b, c) in the 3SAT instance, create a virtual machine with capacity $v_j = 3$.

3. Now, for each literal in a clause, assign the corresponding container to the virtual machine representing that clause. For example:

If the literal is xi, assign container ci_1 to the virtual machine corresponding to the clause C_j.

If the literal is negation xi (negation of xi), assign container ci_2 to the virtual machine corresponding to the clause C_j.

Now, let's analyze the reduction:

If the 3SAT instance is satisfiable:

If the 3SAT instance is satisfiable, there is a truth value assignment to the variables that fulfils all of the clauses. In the corresponding container placement on virtual machine instance, it is necessary to assign assign containers representing "true" for the variables set to true and containers representing "false" for the variables set to false. Since each virtual machine's capacity is 3, and each container has a size of 1 or 2, the assignment of containers to virtual machines will be feasible.

If the container placement on virtual machine instance has a feasible solution:

If the container placement on virtual machine instance has a feasible solution, it means there exists an assignment of containers to virtual machines such that each virtual machine's capacity is not exceeded. This implies that for each clause, at least one literal in that clause evaluates to true. Therefore, the corresponding 3SAT instance is satisfiable.

Since the reduction is valid and 3SAT is known to be NP-hard, the container placement on VM is also NP-hard. Therefore, the container placement on VM is NP-hard when using 3SAT as the reference problem.

For instance, most of the approaches utilized a limited number of containers, virtual machines (VMs), and servers in their experiments, thereby limiting the validity and generalizability of their results. Additionally, they primarily focused on testing their proposed methods in either homogenous or heterogeneous scenarios, without considering their applicability in both settings. Another significant issue with these techniques is that they neglected critical optimization metrics such as energy consumption and service level agreements (SLAs), which are essential for achieving efficient resource utilization and meeting performance targets.

4.3 Methodology

This section presents the workflow for designing and developing an optimal containers placement technique in order to reduce the number of instantiated VMs and active PMs that in turn reduces energy consumption of DCs. A Discrete Firefly algorithm (DFF) and Discrete Firefly algorithm with Local Search Mechanism (DFFLSM), which are metaheuristic algorithms that are based on the flashing behavior of fireflies were used for placing containers on VMs are proposed. These algorithms were compared with pre-existing algorithms like First Fit (FF), First Fit Decreasing (FFD), Random and Ant Colony Optimization (ACO). The results were compared in terms of reduction in average energy consumption, average active VMs, average active PMs and average overall SLA violations. Figure 4.1 shows the workflow for the implementation of container placement algorithms.

4.3.1 Problem formulation Container Placement in Cloud Data Center

This section presents a precise numerical illustration of the proposed work's aim, which is to decrease the use of energy in data centers by minimizing the number of active virtual machines (VMs) and physical machines (PMs). The power function in a cloud data center is a crucial component that determines the amount of energy used, and it is represented mathematically as follows:

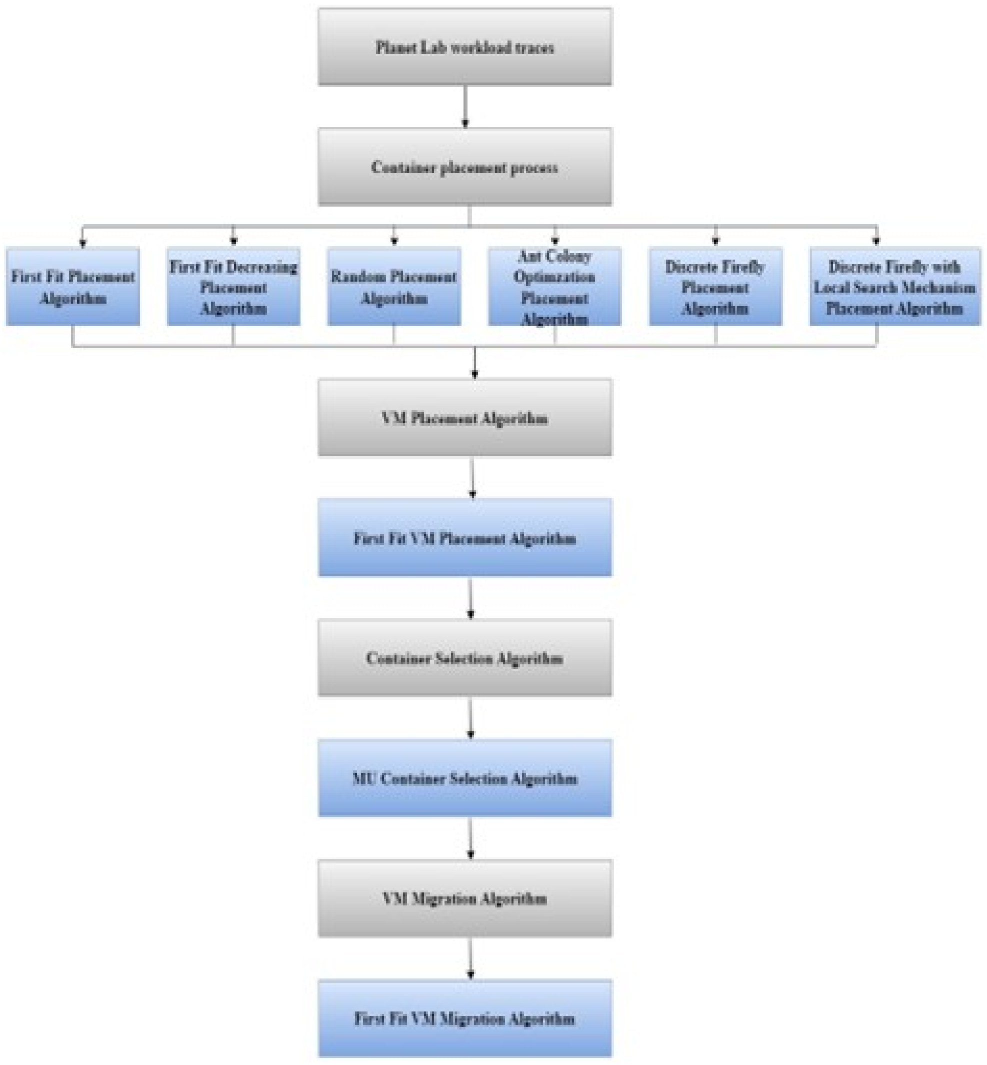

Planet Lab workload traces
Container placement process
First Fit Placement Algorithm
First Fit Decreasing Placement Algorithm
Random Placement Algorithm
Ant Colony Optimization Placement Algorithm
Discrete Firefly Placement Algorithm
Discrete Firefly with Local Search Mechanism Placement Algorithm
VM Placement Algorithm
First Fit VM Placement Algorithm
Container Selection Algorithm
MU Container Selection Algorithm
VM Migration Algorithm
First Fit VM Migration Algorithm

Figure 4.1: Methodology for container placement in cloud data center

$$P_k\,(t) = P_k^{idle} + (P_k^{max} - P_k^{idle}\,) \times U_{(k,t)}, \qquad N_{vm} > 0 \quad (4.1)$$

$$0, \qquad N_{VM} = 0$$

where:

N_{vm} = Number of virtual machines

P_k^{idle} = idle power consumption of physical machine k

P_k^{max} = maximum power consumption of physical machine k

$U_{(k,t)}$ = CPU Utilization percentage of server k at time t

To put it another way, consumption of power may be reduced by minimizing the number of active PMs, which is expressed by:

$$minimize\ N_{pm} = \sum_{k} N_{vmpm_k} \tag{4.2}$$

N_pm stands for the number of PM, N_{vmpm_k} stands for number of VM running on k^{th} PM. Each VM_i $(1 < j < N)$ has CPU capacity VC_i , memory capacity $VMem_i$ and bandwidth capacity VB_i. Each Container j $(1 < j < M)$ has its CPU demand CC_j, memory demand $CMem_j$ and bandwidth demand CB_j. The following constraints are considered to reduce energy consumption or reduce the active number of PMs.

$$y_i = \sum_{j=1}^{M} x_{i,j} \forall\ i\ \in I \tag{4.3}$$

$$\sum_{i=0}^{N} x_{i,j} = 1 \quad \forall\ j \in J \tag{4.4}$$

$$\sum_{j=1}^{M} CC_J.x_{i,j} \leq\ VC_i \tag{4.5}$$

$$\sum_{j=1}^{M} CMem_J.x_{i,j} \leq\ VMem_i \tag{4.6}$$

$$\sum_{j=1}^{M} CB_J.x_{i,j} \leq\ VB_i \tag{4.7}$$

i represents the set of VMs, j represents the set of containers and $x_{i,j}$ represents the assignment of j^{th} container on i^{th} VM. Eq. (4.3) determines whether a VM is used $(yi = 1)$ or not $(yi = 0)$ that is means whether the container has been assigned to the VM or not. Constraint (4.4) demonstrates that a container is exclusively assigned to one of the virtual machines. Constraints (4.5), (4.6), and (4.7) specify the virtual machine's capacity constraints for the CPU, memory, and bandwidth respectively. The constraints in equation (4.5), (4.6) and (4.7) describes the resource capacity requested by container (i.e., requested CPU, requested memory, requested bandwidth) is less than or equal to that of VM.

4.3.2 Process Flow

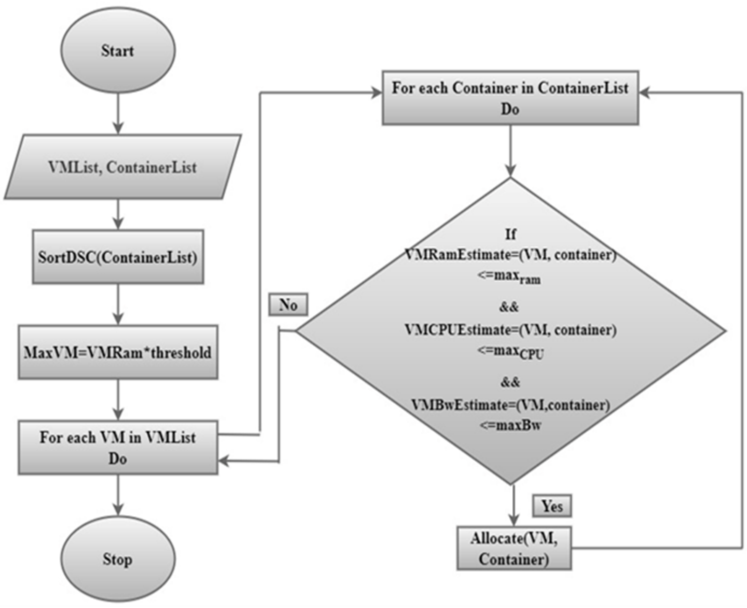
The container placement algorithms have been designed to optimize energy consumption in cloud data centers by optimally placing the containers on VMs. This section focuses on the process flow followed for designing, developing and comparing the different container placement algorithms. Two approaches that utilize metaheuristic algorithms, Discrete Firefly algorithm (DFF) and Discrete Firefly with local search mechanism (DFFLSM), based on the flashing behavior of fireflies, to place containers on virtual machines (VMs) are proposed and implemented. The performance of the proposed algorithms was compared with pre-existing algorithms such as First Fit Decreasing (FFD), First Fit (FF), Random, and Ant Colony Optimization(ACO) in terms of several metrics, including average energy consumption, average active VMs, average active PMs, and average overall SLA violations. Bin packing is an algorithmic technique used for efficient storage of items using the minimum number of boxes. In the field of information technology, this approach can be utilized for tasks such as file storage, IT assistance, and addressing efficiency issues, among others. Specifically, the First-Fit Decreasing (FFD) algorithm (SMIMITE and AFDEL, 2020) is commonly used for container placement in which the containers are sorted in descending order of their RAM capacity. This algorithm allocates the containers to the virtual machines while ensuring that the capacity restrictions of the virtual machines are not violated. In case, the first virtual machine lacks sufficient space, the algorithm moves on to the second one and continues this process until all containers are accommodated. Containers are assigned at random to VMs, which are then hosted at random in hosts servers in Random container placement algorithm (SMIMITE and AFDEL, 2020). The FF container placement algorithm (SMIMITE and AFDEL, 2020) is one of the oldest memory management strategies. This method arranges the bins in the order of first in, first served. The best-suited VM available can be used for the allocation of containers on VM. The Ant Colony Optimization (ACO) is a meta-heuristic algorithm that mimics the natural food-seeking behavior of ants (SMIMITE and AFDEL, 2020). Despite their limited memory, ants have evolved a messaging network that relies on a chemical substance called pheromone, which each ant uses to mark its path. By detecting the amount of pheromone, other ants can plan their movements based on the likelihood of finding food.

4.3.3 Algorithms

Algorithm 1 First Fit Decreasing Conntainer Placement Algorithm

```
Input: ContainerList, VMList
Output: ContainerPlacement
Arrange containers in descending order on the basis of the RAM required
$Max_{ram}$ = VMRam*RAMThreshold
$Max_{bw}$ = total bandwidth capacity of VMs
$Max_{cpu}$ = total CPU capacity of VMs
for container in ContainerList do
    for VM in VMList do
        VMRam estimate (VM, Container) $\leq Max_{ram}$
        VMCPU estimate (VM, Container) $\leq Max_{cpu}$
        VMbw estimate (VM, Container) $\leq Max_{bw}$
        Allocate (VM, Container)
    end for
end for
```

Figure 4.2 shows the flow chart of FFD Container Placement algorithm.



Start
VMList, ContainerList
SortDSC(ContainerList)
MaxVM=VMRam*threshold
For each VM in VMList Do
Stop
For each Container in ContainerList Do
If VMRamEstimate=(VM, container) <=max_ram && VMCPUEstimate=(VM, container) <=max_CPU && VMBwEstimate=(VM,container) <=maxBw
No
Yes
Allocate(VM, Container)

Figure 4.2: Flowchart of FFD Container Placement algorithm

RAM threshold was changed from 100 percent to 50 percent decreasing 10 percent each time. The maximum amount of RAM utilized by can be calculated by the equation below:

$$Max_{ram} = VMRam * RAMThreshold \tag{4.8}$$

Algorithm 2 Random Container Placement Algorithm

```
1: Input: ContainerList, VMList
2: Output: ContainerPlacement
3: $Max_{ram}$ = VMRam*RAMThreshold
4: $Max_{bw}$ = total bandwidth capacity of VMs
5: $Max_{cpu}$ = total CPU capacity of VMs
6: for container in ContainerList do
7:     for VM in VMList do
8:         VMRam estimate (VM, Container) $\leq Max_{ram}$
9:         VMCPU estimate (VM, Container) $\leq Max_{cpu}$
10:        VMbw estimate (VM, Container) $\leq Max_{bw}$
11:        Allocate (VM, Container)
12:    end for
13: end for
```

Figure 4.3 depicts the Random container placement algorithm's flowchart. RAM threshold was changed from 100 percent to 70 percent decreasing 10 percent each time. The maximum amount of RAM utilized by VM can be calculated by the equation 4.8.

Algorithm 3 First Fit Container Placement Algorithm

```
1: Input: ContainerList, VMList
2: Output: ContainerPlacement
3: $Max_{ram}$ = VMRam*RAMThreshold
4: $Max_{bw}$ = total bandwidth capacity of VMs
5: $Max_{cpu}$ = total CPU capacity of VMs
6: for container in ContainerList do
7:     for VM in VMList do
8:         VMRam estimate (VM, Container) $\leq Max_{ram}$
9:         VMCPU estimate (VM, Container) $\leq Max_{cpu}$
10:        VMbw estimate (VM, Container) $\leq Max_{bw}$
11:        Allocate (VM, Container)
12:    end for
13: end for
```

Figure 4.4 depicts the First Fit container placement algorithm's flowchart. RAM threshold was changed from 100 percent to 70 percent decreasing 10 percent each time. The maximum amount of RAM utilized by can be calculated by the equation 4.8.

Figure 4.5 depicts the ACO container placement algorithm's flowchart. Each ant obtains all of the containers and tries to allocate them to the host using the probability choice (ProbabilityVM) algorithm from equation 4.9.

Algorithm 4 Ant Colony Optimization-based Container Placement Algorithm

```
Input: ContainerList, VMList
Output: ContainerPlacement
Initialize parameters, set pheromone value
for container in ContainerList do
    for VM in VMList do
        R = container.getRam()/VM.getRam()
    end for
    initializeRamAllocation(R)
end for
for VM in VMList do
    Cr = VM.getRam()
    initializePheromone(Cr)
end for
for t = 1 to tmax do
    for container in ContainerList do
        Calculate Heuristic function in terms of allocation of each container according
        to equation 4.11
        for VM in VMList do
            Calculate probability according to equation 4.9
            initializeProbability(P)
        end for
        for i = 1 to k do
            vote(VMs, probability)
        end for
        for VM in VMList do
            if MaxVote then
                Allocate(Container, BestVM)
                Update Pheromones according to equation 4.12
            end if
        end for
    end for
end for
```

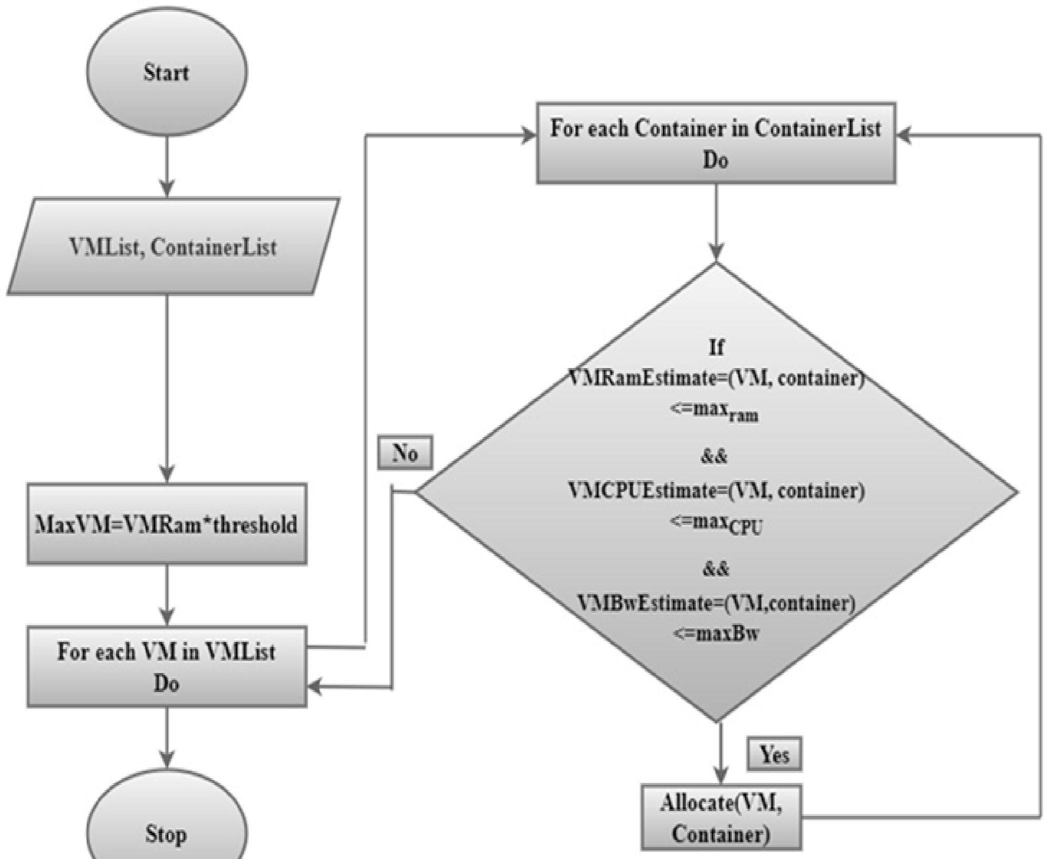

Start
VMList, ContainerList
MaxVM=VMRam*threshold
For each VM in VMList Do
Stop
For each Container in ContainerList Do
If VMRamEstimate=(VM, container) <=max_ram && VMCPUEstimate=(VM, container) <=max_CPU && VMBwEstimate=(VM,container) <=maxBw
No
Yes
Allocate(VM, Container)

Figure 4.3: Flowchart of Random Container Placement algorithm

$$\text{Probability}_{vm} = PH(VM)^{alpha} \times H(VM)^{beta} / \sum PH(VM)^{alpha} \times H(VM)^{beta} \tag{4.9}$$

The probabilistic computation ($Probability_{VM}$) is based on the present pheromone concentration (PH) and a heuristic (H) that helps ants choose the best hosts.

Using equation 4.10, the probable RAM allocation (RA) for each container (CNT) on VM was estimated (SMIMITE and AFDEL, 2020).

$$RA_{CNT}^{VM} = CNT_{RAM}/VM_{RAM} \tag{4.10}$$

As seen in equation 4.11, the heuristic (H) is derived based on 4.10.

$$H_{CNT}^{VM} + = RA(CNT)/RA(CNT)_{VM} \tag{4.11}$$

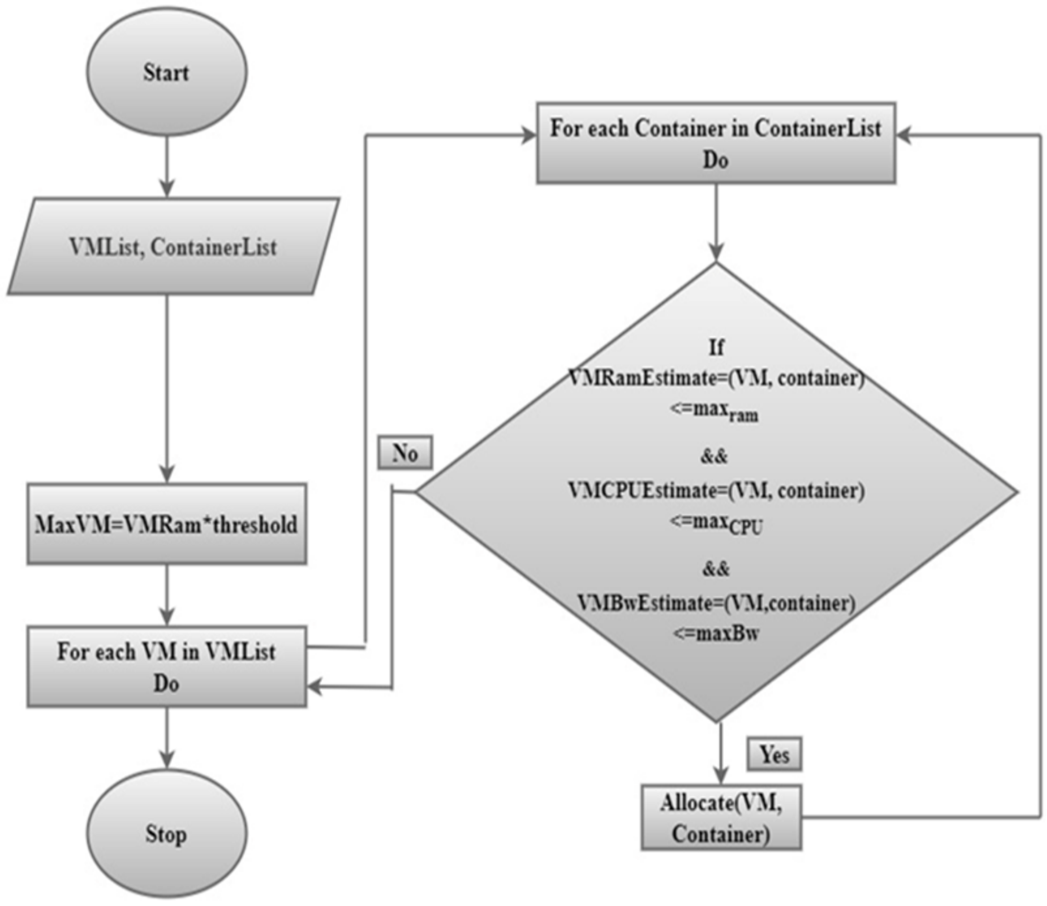

Start
VMList, ContainerList
MaxVM=VMRam*threshold
For each VM in VMList Do
Stop
For each Container in ContainerList Do
If VMRamEstimate=(VM, container) <=max$_{ram}$ && VMCPUEstimate=(VM, container) <=max$_{CPU}$ && VMBwEstimate=(VM,container) <=maxBw
No
Yes
Allocate(VM, Container)

Figure 4.4: Flowchart of FF Container Placement algorithm

The capacity of each host in terms of RAM was used to initialize the pheromone (PH) concentration. After an ant had selected a host, the pheromone concentration was updated using equation 12.

$$PH^{BestVM} = PH^{(BestVM)} * (1 - \rho) + Q/RA^{(BestVM)} \tag{4.12}$$

Where Q is an adaptive parameter and ρ is the evaporation factor.

The original Firefly Algorithm (FA) is a continuous optimization algorithm that is designed to handle continuous variables. In the container-to-VM mapping problem, however, the number of containers mapped to each VM is typically a discrete variable. As such, FA may not be well-suited for the problem, as it may not be able to handle the discrete nature of the variable effectively. DFF can be used for container-to-VM mapping, particularly when the number of containers mapped to each VM is a discrete variable. DFF is a variation of the original Firefly Algorithm that

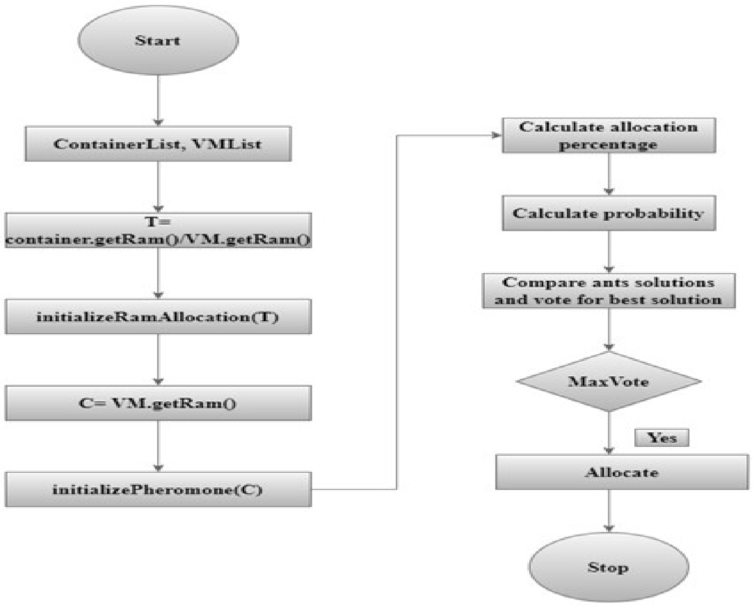

Start
ContainerList, VMList
T= container.getRam()/VM.getRam()
initializeRamAllocation(T)
C= VM.getRam()
initializePheromone(C)
Calculate allocation percentage
Calculate probability
Compare ants solutions and vote for best solution
MaxVote
Yes
Allocate
Stop

Figure 4.5: Flowchart of ACO Container Placement algorithm

can handle discrete variables and is particularly suitable for combinatorial optimization problems (Balaji et al., 2023).

Algorithm 5 Discrete Firefly (DFF) Container Placement Algorithm

```
1: Input: n, $\alpha$, $\beta_o$, $\gamma$, MaxGeneration
2: Output: optimum placement of container on VM
3: Define the objective function $f(\boldsymbol{x}i)$ according to equation 4.15.
4: Initialize $n$ fireflies, $\boldsymbol{x}i = (xi1, xi2, \ldots, x_{id})$.
5: while $t <$ MaxGeneration do
6:     Identify each firefly's absolute brightness according to equation 4.17.
7:     for $i = 1$ to $n$ do
8:         for $j = 1$ to $n$ do
9:             if $I_i > I_j$ then
10:                Evaluate the distance among firefly $i$ and firefly $j$ using equation 4.18.
11:                Evaluate the attractiveness of firefly $j$ attracting $i$ using equation 4.17.
12:                Move firefly $i$ to $j$ using equation 4.19.
13:            end if
14:        end for
15:        Disperse position after moving by discretization strategy given in equation 4.20.
16:    end for
17: end while
18: Obtain the optimum placement of container on VM.
```

Time and Space Complexity

Step 1: Defining the objective function $f(x_i)$ takes constant time and can be considered O(1).

Step 2: Initializing n fireflies, $x_i = (x_{i1}, x_{i2}, \ldots\ldots\ldots, x_{id})$ takes *O(nd)* time, as it requires initializing n fireflies, each with d dimensions.

Step 3: The while loop runs MaxGeneration times, so it has a time complexity of O(MaxGeneration).

Step 4: Identifying each firefly's absolute brightness takes *O(n)* time since it involves evaluating the brightness for each of the n fireflies. The first for loop runs n times. The second for loop also runs n times.

Step 5: The if statement takes constant time and can be considered *O(1)*.

Step 6: Evaluating the distance between firefly i and firefly j takes *O(d)* time, as it requires calculating the distance between two points in d-dimensional space.

Step 7: Evaluating the attractiveness of firefly j attracting i takes constant time and can be considered *O(1)*.

Step 8: Moving firefly i to j takes *O(d)* time, as it involves updating the position of firefly i in each of its d dimensions.

Step 9: The end if statement takes constant time and can be considered *O(1)*.

Step 10: The second for loop runs n times, so its time complexity is *O(n)*.

Step 11: Dispersing position after moving by discretization strategy takes *O(d)* time, as it involves updating the position of firefly i in each of its d dimensions.

Step 12: The end i statement takes constant time and can be considered *O(1)*.

Step 13: The end while statement takes constant time and can be considered *O(1)*.

Step 14: Obtaining the optimum placement of the container on VM takes constant time and can be considered *O(1)*.

Therefore, the overall time complexity of the DFF algorithm is $O(MaxGeneration* n^2 * d)$, where n is the number of fireflies, d is the dimension of each firefly's position vector, and MaxGeneration is the maximum number of genera-

tions (iterations) allowed for the algorithm to run.
Objective function requires memory to store its input values, but its space complexity is negligible compared to the rest of the algorithm. Initialization step involves creating an array of n fireflies, each represented by a d-dimensional vector. The space complexity of this step is *O(nd)*. The main loop of the algorithm involves computing the brightness and attraction of each firefly and updating their positions. The space complexity of this step is also *O(nd)* because it involves updating the same array of n fireflies.
Figure 4.6 shows the flowchart of DFF Container Placement algorithm.

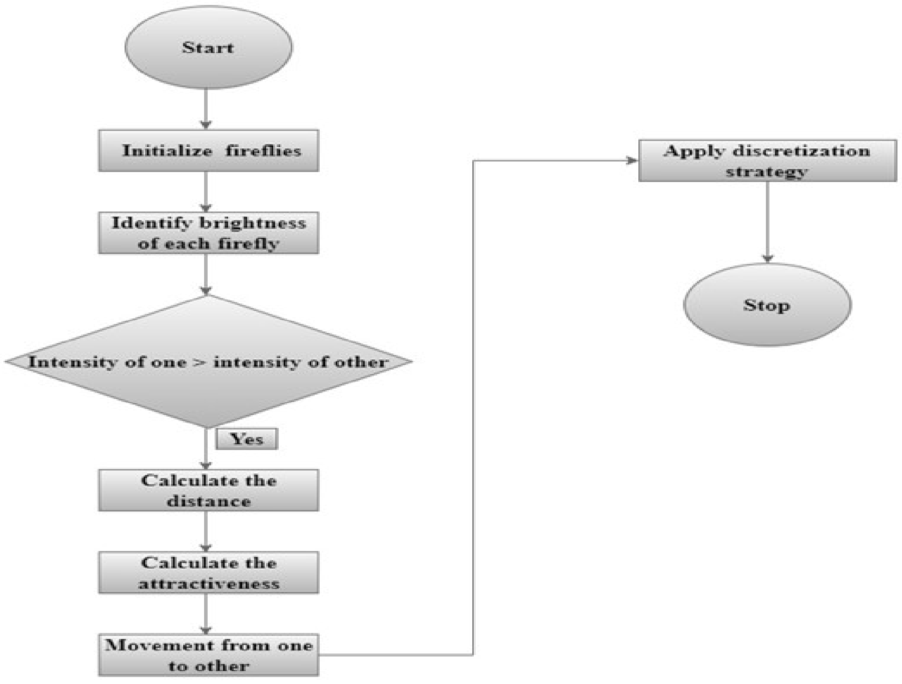

Start
Initialize fireflies
Identify brightness of each firefly
Intensity of one > intensity of other
Yes
Calculate the distance
Calculate the attractiveness
Movement from one to other
Apply discretization strategy
Stop

Figure 4.6: Flowchart of DFF Container Placement algorithm

The resource required by containers to get properly placed on the VMs is calculated using the equation number 4.13.

$$core * MIPS + Bandwidth + RAM \tag{4.13}$$

Here, core refers to the number of processing units in the system, MIPS refers to the processing speed of the CPU, Bandwidth refers to the amount of data that can be transferred between components (e.g., CPU and memory) in a given amount of time and RAM represents main memory size.
Once the containers were placed on the VMs, the resources available to the

VM after placement were calculated using equation 4.14.

$$core * MIPS + Bandwidth + RAM \tag{4.14}$$

Initial populations of fireflies were generated randomly and the $F(x_i)$ is calculated using equation 4.15.

$$F\ (x_i)\ =\ VM_i^{ra}\ -\ Cont_j^{ru} \tag{4.15}$$

where $F(x_i)$ is the resource left of the VM after placement of the container.

The firefly algorithm contains the three sections that are described below:

The absolute luminosity I_j of firefly j is defined as objective function value at x_j

$$I_j = F(x_j) \tag{4.16}$$

In the target minimization issue, the smaller the objective function value, the brighter the absolute luminosity of fireflies.
Attractiveness: The brightness and attractiveness are proportionate, and both diminish as the distance separating among fireflies grows. As a consequence, if there are two flashing fireflies, the less luminous one will move towards the brighter one. It will travel at random if there is no other firefly that is bright than it.

$$\beta_{ji} =\ \beta_0 \times\ e^{-\gamma_{rij}^2} \tag{4.17}$$

Where β_0 is the attractiveness at r = 0, gamma a fixed light absorption coefficient, r_{ij} is the distance between the i_{th} and j_{th} firefly, is listed as below:

$$r_{ij} = \sqrt{\sum_{k=1}^{d} (x_{ik} -\ x_{jk})^2} \tag{4.18}$$

Movement: A firefly i's movement is drawn to another, more appealing (brighter) firefly j, as determined by (Ding et al., 2018):

$x_i(t+1) = x_i(t) + \beta_{ji}(x_j(t) - x_i(t)) + \alpha(rand - 0.5 * A)$

(4.19)

where the second term is due to the attraction, and β_{ji} is the attractiveness. The third term is randomization and α represents the randomization argument.

Discretization Strategy: When firefly i approaches towards firefly j, its location changes from a binary number to a real number. As a result, it must be substituted by a binary number for this actual integer. The following sigmoid function limits S (x_{ik}) to the range of zero to one.

$$S(x_{ik}) \ = \ \frac{1}{1+e^{-x_{ik}}} \tag{4.20}$$

where $S(x_{ik})$ denotes the probability of bit x_{ik} taking 1.

Local search mechanisms are often introduced (Saber et al., 2018) in metaheuristic algorithms to help improve the quality of solutions and avoid being stuck in local optima. Based on the vast number of alternative configurations, the container-to-VM mapping issue is a combinatorial optimization problem, which implies that finding the best solution might be challenging. Local search can help the algorithm explore the search space more effectively and find better solutions. Overall, local search mechanism can be a useful addition to DFF container placement algorithm, as it can help improve the quality of solutions and avoid being stuck in local optima. DFF container placement algorithm is a global search algorithm designed to explore a large search space and find a good solution. However, they can sometimes be stuck in local optima or suboptimal solutions. This can happen when the algorithm converges to a suboptimal solution that appears to be brighter than other nearby solutions, causing the fireflies to move towards it and being stuck in a suboptimal solution. By incorporating a local search mechanism (Saber et al., 2018), the solutions found by the global search algorithm can be further refined and hence can help to potentially escape from suboptimal solutions. In case of DFFLSM container placement algorithm, the local search mechanism can help to refine the solutions found by the algorithm by exploring the neighborhood of each firefly and improving its position based on its attraction to other fireflies. This can potentially lead to better solutions and improve the quality of the final solution obtained by the algorithm.

Algorithm 6 Discrete Firefly (DFF) Container Placement Algorithm with local search mechainsm

```
1: Input: n, $\alpha$, $\beta_o$, $\gamma$, MaxGeneration
2: Output: optimum placement of container on VM
3: Define the objective function $f(\boldsymbol{x}i)$ according to equation 15.
4: Initialize $n$ fireflies, $\boldsymbol{x}i = (xi1, xi2, \ldots, x_{id})$.
5: while $t <$ MaxGeneration do
6:     Identify each firefly's absolute brightness according to equation 4.17.
7:     for $i = 1$ to $n$ do
8:         for $j = 1$ to $n$ do
9:             if $I_i > I_j$ then
10:                Evaluate the distance among firefly $i$ and firefly $j$ using equation 4.18.
11:                Evaluate the attractiveness of firefly $j$ attracting $i$ using equation 4.17.
12:                Move firefly $i$ to $j$ using equation 4.19.
13:            end if
14:        end for
15:        Disperse position after moving by discretization strategy given in equation
   4.20.
16:    end for
17:    if $t > 2$ then
18:        Optimize the local optimal solution using local search mechanism.
19:    end if
20: end while
21: Obtain the optimum placement of container on VM.
```

LocalSearchMechanism()

1. for m = 1:N_{vm}
2. Col_{m+1}= Col_m+Col_{m+1}
3. Col_m=0
4. for i=1:N_{vm}
5. for j=2:N_{vm}
6. if ($I_i > I_j$)
7. Update solution with the new placement scenario generated
8. else

```
9. end for
10. end for
11. end for
12. Repeat steps 2 to 7
```

A local search mechanism is used to improve the quality of the best solution while also speeding up the outcome finding procedure. Considering that the answer provided after iterating once is the optimal solution, the following local search strategy was used to increase the quality of this optimum result and speed up the next iterative search: The first column's values were added to the second column, and these were the new values of the second column. The first column's values were all replaced with 0. As a result, a new type of placement was created. When the relevant value of the objective function was compared to the objective function value of the previous placement result, the optimal solution was updated; otherwise, the value in the second column was added to the third column, resulting in the new value of the third column. The second column's values were all replaced with 0. The procedure continued by comparing the new objective function values to the old objective function values; if the difference was smaller, the optimal solution was updated. If no such enhanced optimal placement scenario existed then the algorithm continued with the existing placement scenario generated at step 14 of algorithm 6.

Time and Space Complexity

Overall, the time complexity of the DFFLSM algorithm can be expressed as $O(MaxGeneration * n^2 * d)$, where n is the number of fireflies, d is the dimension of each firefly's position vector, and MaxGeneration is the maximum number of generations (iterations) allowed for the algorithm to run. If the LocalSearchMechanism function is executed when $(t > 2)$, it involves another nested for loop over all VMs $(O(N_{vm}))$, and within each VM, another nested for loop to compare each pair of fireflies $(O(n^2))$. The updated solution with a new placement scenario is assumed to have a constant time complexity. Therefore, the overall time complexity is $O(n^2 * d * MaxGeneration + N_{vm} * n^2)$ or simply $O(n^2 * d * MaxGeneration)$, if LocalSearchMechanism is not executed.

Objective function $F(x_i)$ requires memory to store its input values, but its space complexity is negligible compared to the rest of the algorithm. Initialization step involves creating an array of n fireflies, each represented by a d-dimensional vector. The space complexity of this step is $O(nd)$. The main loop of the algorithm involves computing the brightness and attraction of each firefly and updating their positions. The space complexity of this step is also O(nd) because it involves updating the same array of n fireflies. Local Search Mechanism function operates on the same array of n fireflies and does not introduce any new data structures. Therefore, its space complexity is also negligible. Having the Local Search Mechanism function does not affect the main loop of the algorithm, which still requires an array of n fireflies, each represented by a d-dimensional vector, and thus the space complexity of $O(n * d)$ is maintained.

Figure 4.6 depicts the DFFLSM container placement algorithm's flowchart.

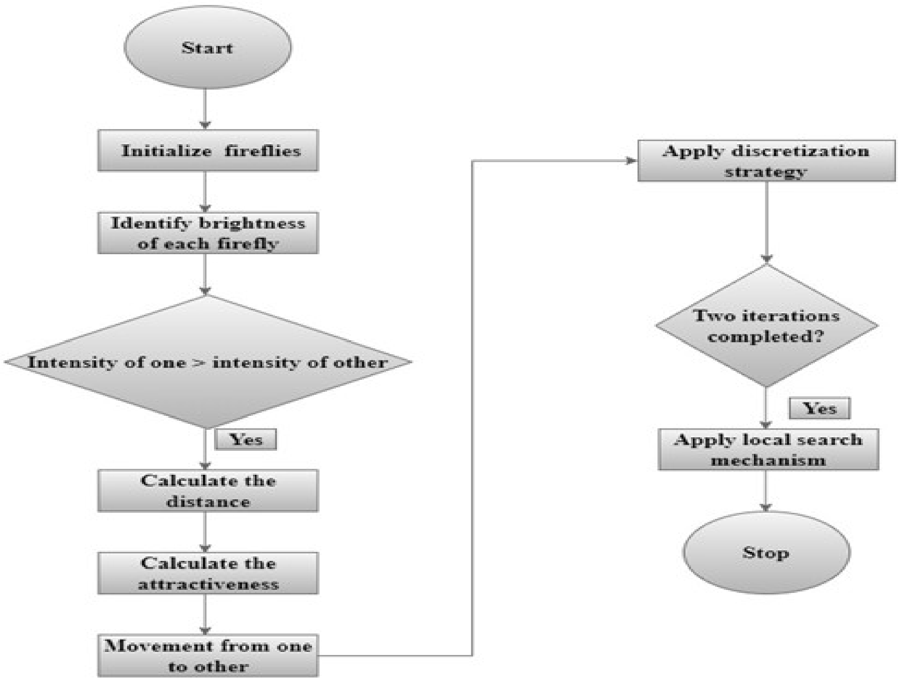

Start
Initialize fireflies
Identify brightness of each firefly
Intensity of one > intensity of other
Yes
Calculate the distance
Calculate the attractiveness
Movement from one to other
Apply discretization strategy
Two iterations completed?
Yes
Apply local search mechanism
Stop

Figure 4.7: Flow chart of DFFLSM Container Placement algorithm

4.4 Experimental Setup

The experiments have been conducted in Cloudsim 4.0 with i5-2.4GHz processor and 8 GB RAM. Table 4.2 shows the objectives and various sets of experiments conducted for different combinations of container placement algorithms.

Table 4.2: Objectives and Experiment sets for Container Placement

Set	**Objective**	**RAM Threshold**	**CPU Threshold**	**Bandwidth**	**Overbooking factor**
1	Examine the influence of CP algorithms on average energy consumption.	[70%,80%,90%,100%]	90%	80%	80%
2	Examine the influence of CP algorithms on average active VM.	[70%,80%,90%,100%]	90%	80%	80%
3	Examine the influence of on CP algorithms on average active PM	[70%,80%,90%,100%]	90%	80%	80%
4	Examine the influence of CP algorithms on average overall SLA.	[70%,80%,90%,100%]	90%	80%	80%
5	Examine the influence of CP algorithms corresponding to different container overbooking factor on average energy consumption, average active VM, average active PM and average overall SLA.	80%	90%	80%	20,40,80%

Table 4.3 shows the values of different arguments/parameters used in

ACO, DFF and DFFLSM container placement algorithms. For each container placement algorithm, the threshold values for RAM, CPU and Bandwidth were fixed as: RAM threshold varied as 70 percent, 80 percent, 90 percent and 100 percent; CPU threshold was fixed as 90 percent and bandwidth threshold is fixed as 80 percent. The results were calculated corresponding each RAM threshold.

Table 4.3: Parameter values used in algorithms (ACO, DFF and DFFLSM)

Parameters	**Values**
ACO CP algorithm	
α	1
β	2
ρ	0.7
Q	100
DFF & DFFLSM CP algorithm	
α	1
β	1
A	1
γ	0.01

4.4.1 Scenario 1: Homogenous Environment

In this case, a DC was simulated with 700 homogenous servers and 1000 VMs. A total of 5000 containers were used which belonged to 3 different types. The configurations of the PMs are shown in table 4.4 and that of VMs and containers are shown in Table 4.5.

Table 4.4: PM configurations in homogenous environment for Container Placement

Physical Machine Configurations and Power Models					
Type of Server	**CPU [3GHz] (37274 MIPS/core)**	**Memory (GB)**	**P idle (Watt)**	**P max (Watt)**	**Number**
1	4 Core	64	93	135	700

4.4.2 Scenario 2: Heterogenous Environment

In this case, a DC was simulated with 700 heterogenous servers and 1000 VMs with the set of 3 different servers and 4 different types of VMs. A total of 5000 containers were used which belonged to 3 different types

Table 4.5: VM and container configurations in homogenous environment for Container Placement

Type of Containers and VMs						
Type of Container	**CPU MIPS (1 core)**	**Memory (MB)**	**Type of VM**	**CPU [1.5GHz] (18636 MIPS/core)**	**Memory (GB)**	**Number**
1	4658	128	1	2	2	1000
2	9320	256				
3	18636	512				

as that of in homogenous environment. The configurations of the PMs are shown in table 4.6 and that of VMs and containers are shown in table 4.7.

Table 4.6: PMs configurations in heterogenous environment for Container Placement

PM Configurations and Power Models					
Type of Server	**CPU [3GHz] (37274 MIPS/core)**	**Memory (GB)**	**P idle (Watt)**	**P max (Watt)**	**Number**
1	4 Core	64	86	117	234
2	8 Core	128	93	135	233
3	16 Core	256	66	247	233

Table 4.7: VMs and container configurations in heterogenous environment for Container Placement

Types of Containers and VMs						
Type of Container	**CPU MIPS (1 core)**	**Memory (MB)**	**Type of VM**	**CPU [1.5GHz] (18636 MIPS /core)**	**Memory (GB)**	**Number**
1	4658	128	1	2	1	250
2	9320	256	2	4	2	250
3	18636	512	3	1	4	250
			4	8	8	250

4.5 Results

In experiments, the following cases have been investigated:

A) Examine the influence of Container Placement algorithms on average energy consumption.

B) Examine the influence of Container Placement algorithms on average active VM.
C) Examine the influence of Container Placement algorithms on average active PM.
D) Examine the influence of Container Placement algorithms on average overall SLA violations.
E) Examine the influence of Container Placement algorithms corresponding to different overbooking factor on average energy consumption, average active VM, average active PM and average overall SLA violations.

All the above cases have been considered for both homogenous and heterogenous environment. PlanetLab workload traces (Park and Pai, 2006) were used to assess the container placement algorithms considering the simulation setup and parameters indicated in section 4.2. Each of these records contained 10 days of experimentation workload from randomly selected providers, which were gathered during March and April 2011 (Beloglazov and Buyya, 2012). The dataset included CPU utilization percentages for over a thousand VMs deployed across more than 500 global locations. The data was structured with each day represented as a separate folder, and within each folder, individual files contained daily CPU utilization readings for each VM, sampled at 5-minute intervals. Each container was allocated to a task that contains statistics on CPU usage for one day that was reported every five minutes.

4.5.1 Homogenous Environment

4.5.1.1 Examine the influence of Container Placement algorithms on average energy consumption:

The average energy usage of a DC was calculated corresponding to different CP algorithms at different RAM thresholds. The comparison of average energy consumed by these CP algorithms at different RAM thresholds is as shown in figure 4.8. DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average energy consumption with energy saving of 10.05%, 8.79%, 15.91%, 7.61% and 4.25% respectively with changing RAM thresholds as shown in table 4.8. DFF outperformed FF, FFD, Random and ACO in terms of average energy consumption with energy saving of 6.05%, 4.74%, 12.18% and 3.51% respectively with changing RAM thresholds as shown in table 4.9.

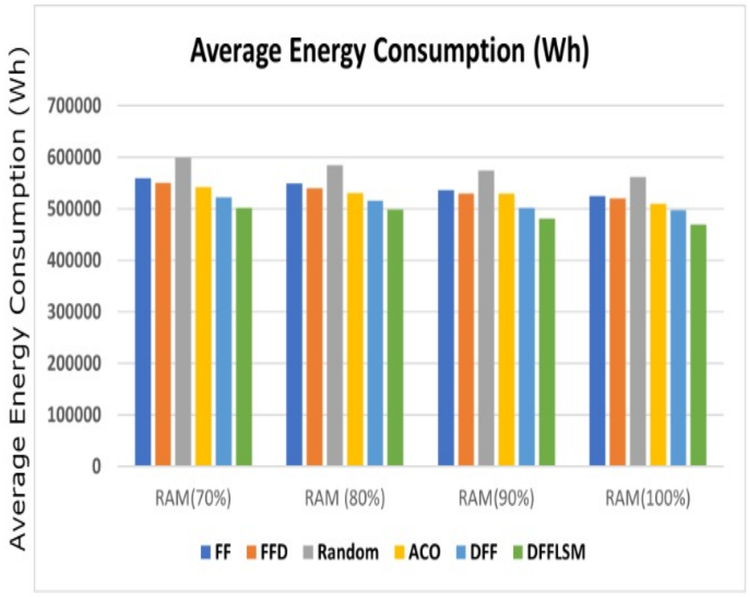


Figure 4.8: Comparison of average energy consumption in homogenous environment for different container placement algorithms

4.5.1.2 Examine the influence of Container Placement algorithms on average active VM:

The comparison of different container placement algorithms for average active VM at different RAM thresholds is as shown in figure 4.9. DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average active VM with a percentage of 48.63%, 24.26%, 51.21%, 35.29% and 20.94% respectively with changing RAM thresholds as shown in table 4.8. DFF outperforms FF, FFD, Random and ACO in terms of average active VM with a percentage of 35.03%, 24.26%, 38.28% and 18.15% respectively with changing RAM thresholds as shown in table 4.9.

4.5.1.3 Examine the influence of Container Placement algorithms on average active PM:

The comparison of different container placement algorithms for average active PM at different RAM thresholds is as shown in figure 4.10.
DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of

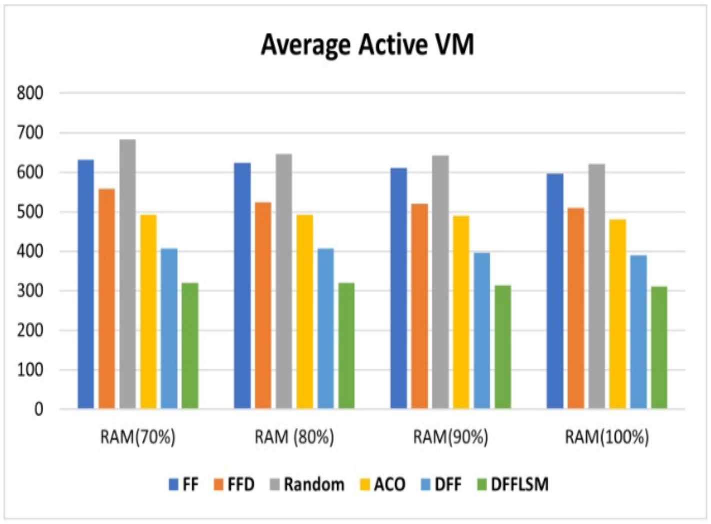


Figure 4.9: Comparison of average active VM in homogenous environment for different container placement algorithms

average active PM with a percentage of 24.50%, 18.59%, 27.62%, 12.40% and 8.36% respectively with changing RAM thresholds as shown in table 4.8.

DFF outperformed FF, FFD, Random and ACO in terms of average active PM with a percentage of 17.60%, 11.16%, 21.02% respectively with changing RAM thresholds is as shown in table 4.9.

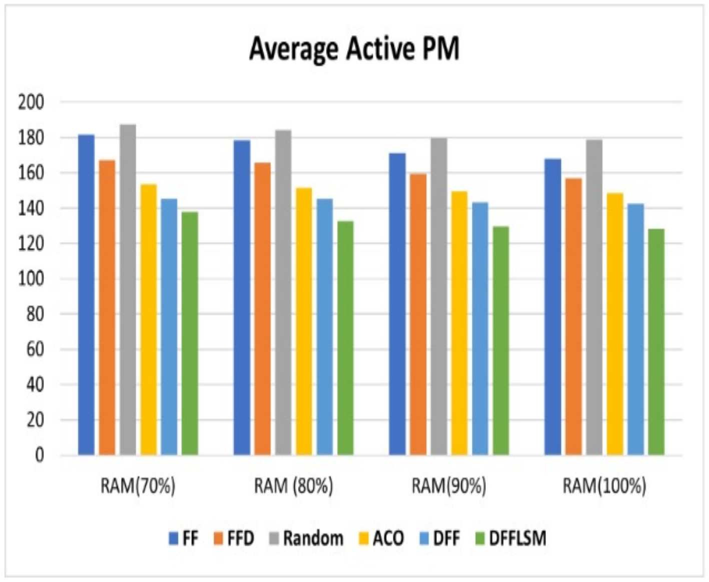


Figure 4.10: Comparison of average active PM in homogenous environment for different container placement algorithms

Table 4.8: Percentage Reduction in average energy consumption, average active VM and average active PM of DFFLSM in comparison to other container placement algorithms in homogenous environment

Homogenous Environment					
	Average reduction in energy consumption (%)				
	FF	**FFD**	**Random**	**ACO**	**DFF**
DFFLSM	10.05	8.79	15.91	7.61	4.25
	Average reduction in active VM (%)				
	48.63	24.26	51.21	35.29	20.94
	Average reduction in active PM (%)				
	24.5	18.59	27.62	12.4	8.36

Table 4.9: Percentage reduction in average energy consumption, average active VM and average active PM of DFF in comparison to other container placement algorithms in homogenous environment

Homogenous Environment				
	Average reduction in energy consumption (%)			
	FF	**FFD**	**Random**	**ACO**
DFF	6.05	4.74	12.18	3.51
	Average reduction in active VM (%)			
	35.03	24.26	38.28	18.15
	Average reduction in active PM (%)			
	17.6	11.16	21.02	4.41

4.5.1.4 Examine the influence of Container Placement algorithms on average overall SLA violations:

According to equation 5.3, the SLA metric is defined as a proportion of the difference between the required and allocated amount of CPU for each VM. It is violated, if the VM that hosts the container does not get the necessary amount of CPU. The container placement algorithms optimizes the initial distribution of containers onto VMs, resulting in fewer SLA breaches in the experiment, ensuring that each VM receives the necessary resources. The impact of different container placement algorithms on average overall SLA violations for different RAM thresholds is shown in figure 4.11.

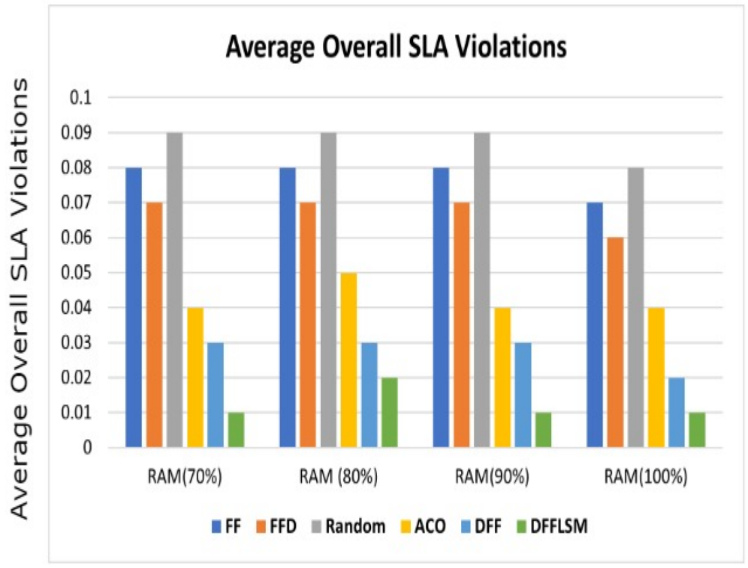
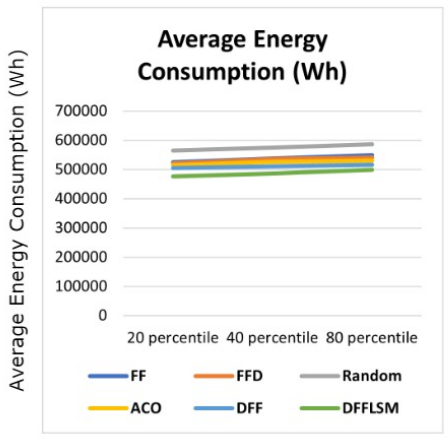


Figure 4.11: Comparison of average overall SLA violations in homogenous environment for different container placement algorithms

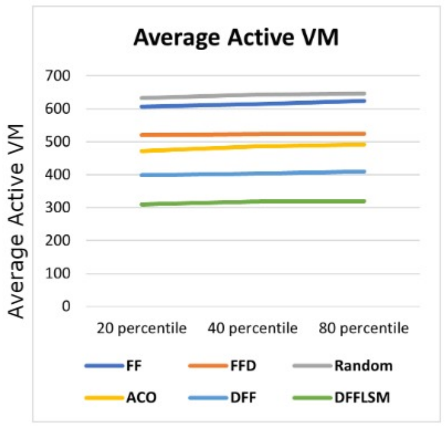
4.5.1.5 Examine the influence of Container Placement algorithms corresponding to different overbooking factor on average energy consumption, average active VM, average active PM and average overall SLA violations:

Overbooking is a significant factor that impacts the efficiency of container placement and consolidation algorithms in terms of energy utilization, average active VM, average active PM and SLA breaches. Containers were assigned to VMs based on a predetermined percentage of application workload. The greater the percentile, the fewer containers could fit on each

VM. For this set of experiment, the overbooking factor was set as 20, 40, or 80 percentile. The impact of different container placement algorithms for different values of overbooking factor on average energy consumption, average active VM, average active PM and overall average SLA violations is as shown in figure 4.12 and figure 4.13 respectively. The RAM threshold in this set of experiment was kept at 80%.

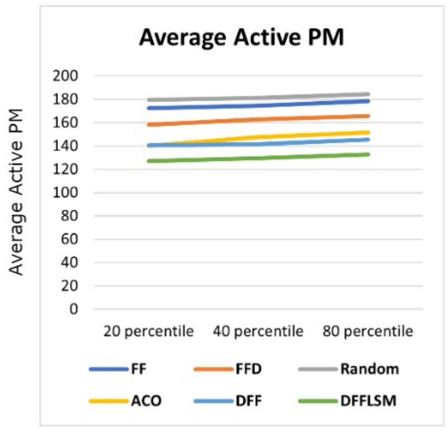


(a)

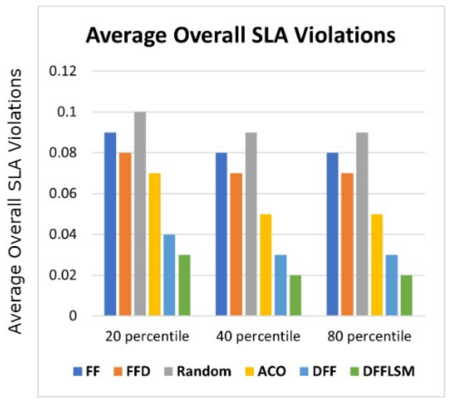


(b)

Figure 4.12: Impact of different container placement algorithms on data center (a) average energy consumption and (b) average active VM for different values of overbooking factor (RAM Threshold= 80 percent) in homogenous environment



(a)



(b)

Figure 4.13: Impact of different container placement algorithms on data center (a) average active PM and (b) average overall SLA violations for different values of overbooking factor (RAM Threshold= 80 percent) in homogenous environment

4.5.2 Heterogenous Environment

4.5.2.1 Examine the influence of Container Placement algorithms on average energy consumption:

The comparison of different container placement algorithms for average energy consumed at different RAM thresholds is as shown in figure 4.14. DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average energy consumption with energy saving of 50.63%, 49.64%, 54.21%, 31.09% and18.70% respectively with changing RAM thresholds as shown in Table 4.10. DFF outperformed FF, FFD, Random and ACO in terms of average energy consumption with energy saving of 39.27%, 38.05%, 43.67% and 15.24% respectively with changing RAM thresholds as shown in table 4.11.

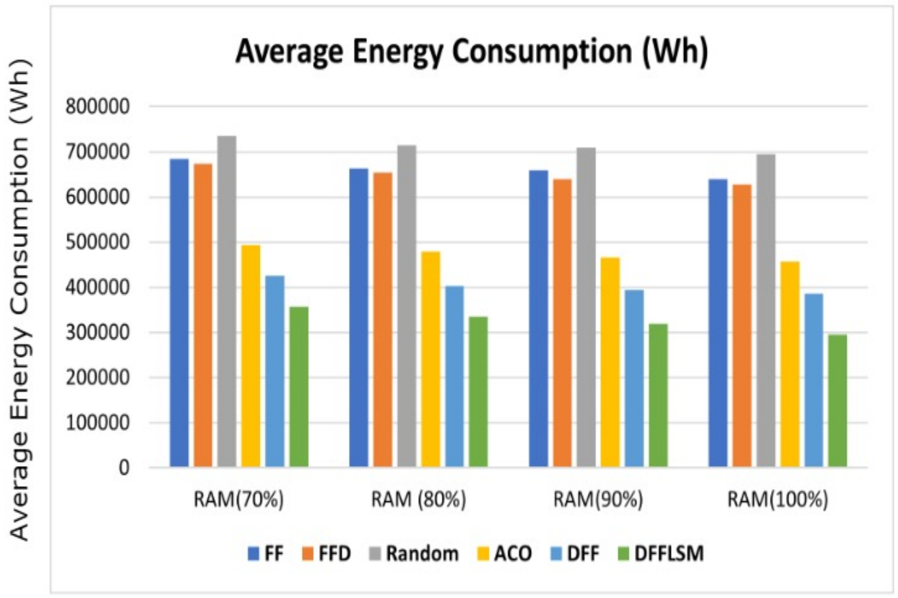


Figure 4.14: Comparison of average energy consumption in heterogenous environment for different container placement algorithms

4.5.2.2 Examine the influence of Container Placement algorithms on average active VM:

The comparison of different container placement algorithms for average active VM at different RAM thresholds is as shown in figure 4.15. DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average active VM with a percentage of 45.54%, 41.89%, 48.23%, 18.79% and 12.95% respectively with changing RAM thresholds as shown in table 4.10.

DFF outperformed FF, FFD, Random and ACO in terms of average active VM with a percentage of 37.43%, 33.24%, 40.53%, 6.69% respectively with changing RAM thresholds as shown in table 4.11.

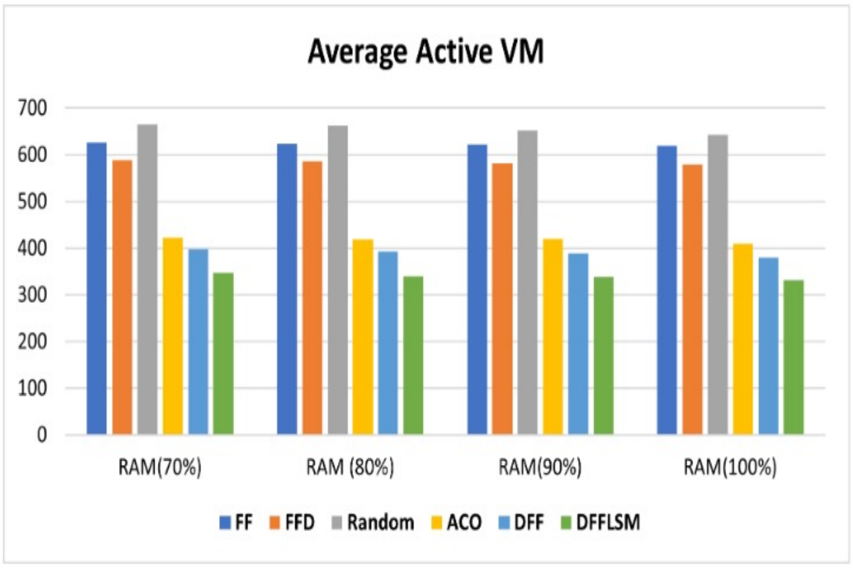


Figure 4.15: Comparison of average active VM in heterogenous environment for different container placement algorithms

4.5.2.3 Examine the influence of Container Placement algorithms on average active PM:

The comparison of different container placement algorithms for average active PM at different RAM thresholds is as shown in figure 4.16. DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average active PM with a percentage of 26.16%, 22.22%, 37.88%, 14.34% and 8.85% respectively with changing RAM thresholds as shown in table 4.10. DFF outperformed FF, FFD, Random and ACO in terms of average active PM with a percentage of 18.99%, 14.67%, 31.84% and 6.02% respectively with changing RAM thresholds as shown in table 4.11.

4.5.2.4 Examine the influence of Container Placement algorithms on average overall SLA violations:

The impact of different container placement algorithms on average overall SLA violations for different RAM thresholds is shown in figure 4.17.

Table 4.10: Percentage reduction in average energy consumption, average active VM and average active PM of DFFLSM in comparison to other container placement algorithms in heterogenous environment

Heterogenous Environment					
	Average Energy Consumption (%)				
	FF	**FFD**	**Random**	**ACO**	**DFF**
	50.63	49.64	54.21	31.09	18.7
DFFLSM	**Average Active VM (%)**				
	45.54	41.89	48.23	18.79	12.95
	Average Active PM (%)				
	26.16	22.22	37.88	14.34	8.85

Table 4.11: Percentage reduction in average energy consumption, average active VM and average active PM of DFF in comparison to other container placement algorithms in heterogenous environment

Heterogenous Environment				
	Average Energy Consumption (%)			
	FF	**FFD**	**Random**	**ACO**
DFF	39.27	38.05	43.68	15.24
	Average Active VM (%)			
	37.43	33.24	40.53	6.69
	Average Active PM (%)			
	18.99	14.67	31.84	6.02

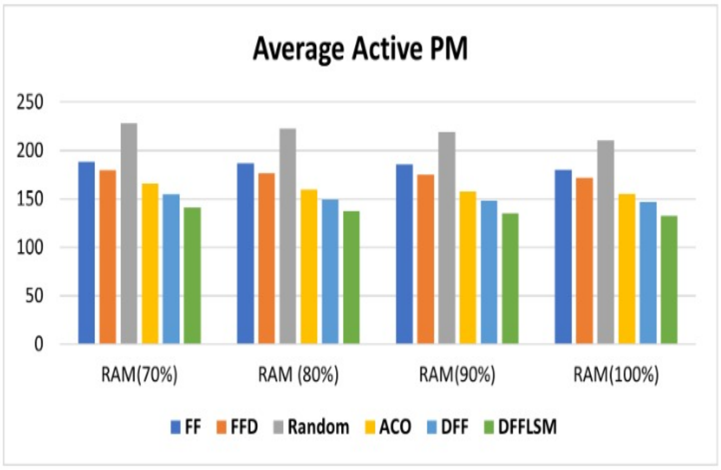


Figure 4.16: Comparison of average active PM in heterogenous environment for different container placement algorithms

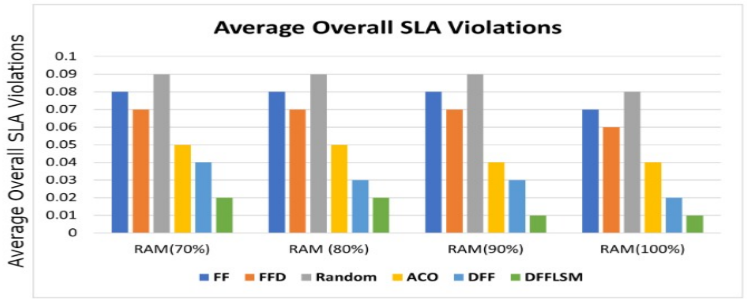


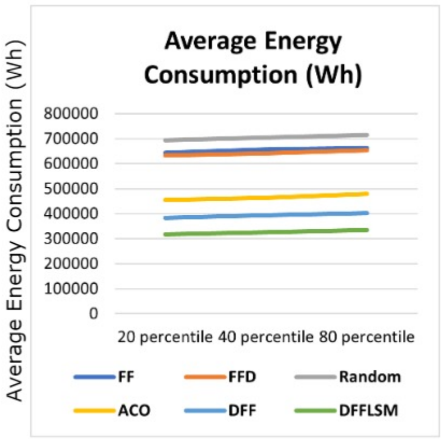
Figure 4.17: Comparison of average overall SLA violations in heterogenous environment for different container placement algorithms

4.5.2.5 Examine the influence of Container Placement algorithms corresponding to different overbooking factor on average energy consumption, average active VM, average active PM and average overall SLA violations:

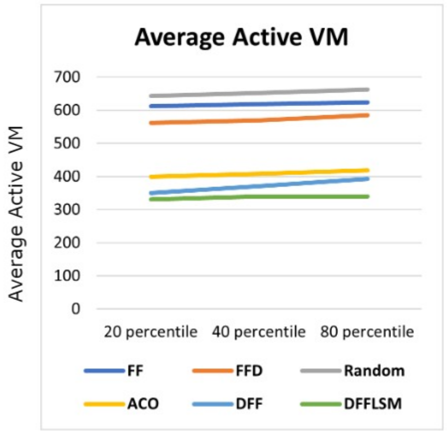
Average energy consumption and average active PM is shown in figure 4.18. Average active PM and overall average SLA violations is as shown in figure 4.19.

4.6 Summary

Container placement stands as a pivotal aspect of optimizing resource utilization and overall performance in cloud data centers. With the rise of containerization technology, which encapsulates applications and their dependencies into isolated units called containers, the manner in which

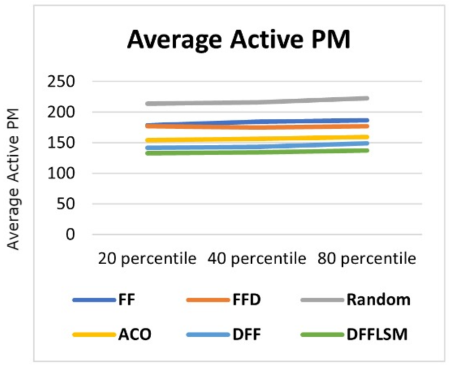


(a)

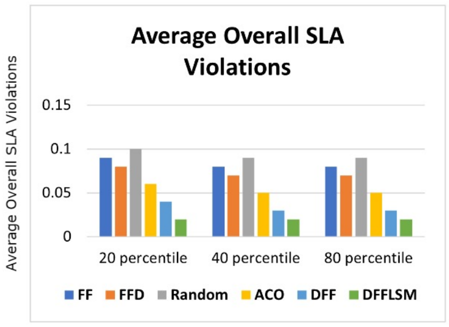


(b) a

Figure 4.18: Impact of different container placement algorithms on data center (a) average energy consumption and (b) average active VM for different values of overbooking factor(RAM Threshold= 80) in heterogenous environment



(a)



(b)

Figure 4.19: Impact of different container placement algorithms on data center average active PM (a) and average overall SLA violations (b) for different values of overbooking factor (RAM Threshold= 80) in heterogenous environment

these containers are distributed across the available physical servers holds significant importance.

Efficient container placement involves intelligent decision-making algorithms that determine the optimal arrangement of containers on servers. This process takes into account factors such as resource availability, workload characteristics, and energy efficiency goals. The aim is to strike a balance between maximizing server utilization, ensuring high application performance, and minimizing energy consumption.

Effective container placement strategies contribute to enhanced data center efficiency by preventing resource wastage and minimizing contention for resources. It helps prevent issues such as performance bottlenecks and ensures that applications coexist harmoniously on the same physical server. Moreover, intelligent placement can lead to energy savings by consolidating containers onto fewer servers, thus reducing the need for unnecessary hardware utilization.

The proposed DFFLSM algorithm for container placement demonstrated superior performance compared to existing algorithms, including FF, FFD, ACO, and DFF, leading to significant optimizations in data center energy utilization. When contrasted with these established methods, DFFLSM achieved remarkable results, exhibiting a reduction of 9.32% and 40.85% in average data center energy consumption within homogeneous and heterogeneous environments, respectively. Furthermore, it achieved a substantial decrease in the average count of active physical machines (PMs), marking an 18.30% and 21.89% reduction in homogeneous and heterogeneous settings when compared to both existing and proposed techniques. In terms of energy efficiency, the DFF algorithm outperformed the FF, FFD, Random, and ACO container placement algorithms. Compared to these methods, DFF brought about noteworthy improvements, achieving energy consumption reductions of 6.62% and 34.06% in homogeneous and heterogeneous environments, respectively. Additionally, it achieved a significant drop in the average active PMs by 13.55% and 17.88% in homogeneous and heterogeneous environments, respectively, when compared to pre-existing approaches. These findings underscored the effectiveness of DFFLSM and DFF in enhancing data center energy efficiency and operational performance across diverse scenarios.

Chapter 5

Container Consolidation in Cloud Data Center

In this chapter, various container consolidation algorithms are explored, emphasizing their role in enhancing energy efficiency within cloud data centers. It highlights the implementation and impact of these algorithms while shedding light on the efficacy of proposed and implemented metaheuristic solutions in effectively reducing data center energy consumption.

5.1 Introduction

Virtualization optimises resource utilisation by allowing many operating systems (Guest OS) to operate on a single physical server's hardware. Containerization, on the other hand, enables several applications to operate within the same OS on a single VM or server (Jain and Choudhary, 2016). VMs are ideal for organisations that demand OS functionality and functionalities while running many applications on a server or managing multiple OSs. Containers are a better option for reducing the number of servers necessary for multiple services (Guan et al., 2017). While previous efforts to lower both energy and task completion time in cloud DC have been made, they have not been progressive, resulting in a high number of container migrations. When evaluating workload adjustments or condensing the workload to a smaller number of servers to conserve energy, the cost of container migrations is ignored (Abdelbaky et al., 2015). Downtime is caused by container migration (for example, CRIU), and repeated migrations can have a detrimental impact on job completion timescales and are likely to result in Service Level Agreement (SLA) breaches. As a result, it is preferable to have a DC placement approach that minimises power, job completion time, and container movements while being expandable to its size (Xu and Fortes, 2010). SLAs are commonly used in the cloud to ensure consistent QoS between cloud providers and clients. There has been an increased desire to reduce energy consumption while meeting SLAs (Carvalho et al., 2017). In the Infrastructure as a Service (IaaS) model, addressing this issue involves optimizing the usage of Physical Machines (PMs) and implementing load balancing measures. However, in light of the recent introduction of Container as a Service (CaaS) offerings by cloud providers, containers are gaining increasing significance and are poised to become the predominant deployment method within the cloud ecosystem, especially in Platform as a Service (PaaS) environments. Consequently, the need to reduce power consumption while still maintaining Service Level Agreements (SLAs) at the Virtual Machine (VM) level has become of paramount importance. Containerization outperforms virtualization in terms of speed and installation density. Container implementation density is frequently 6-12 times that of VM installation density due to their lightweight nature (Liu et al., 2020). When a cloud provider maintains a high installation density, fewer PMs are deployed to serve the same number of customers, resulting in lower capital (buy) and operational expenditures (e.g., energy). Given these considerations, various open-source groups and businesses are increasing their investments in container-based virtualization solutions. It is also worth noting that containers and virtual machines get along well. That is, VM-Docker combinations clearly achieve near native, if not somewhat superior, efficiency. CaaS services are therefore developed on top of VMs (Li et al., 2015) (Affetti et al., 2015). VMs, according to cloud providers, give an added degree of security to untrusted apps. Meanwhile, containers provide a good setting for semi-trusted workloads (Piraghaj et al., 2015a).

5.2 Literature Review

Various container migration techniques have been created to provide load-balancing services and to reduce computing machine energy usage. In a study conducted by Chen et al. (2018), a container migration strategy for container consolidation was devised. The algorithm's performance was assessed based on various metrics, including the number of virtual machines created, the average rate of container migrations, energy consumption and violations of service level agreements. Meanwhile, U-Chupala et al. (2017) introduced a container rebalancing approach that operates concurrently with a scheduling process. This container rebalancing method improved the utilization of LXC clusters while ensuring minimal interference with the scheduling process. Zheng (Zheng et al., 2021) introduced a container resource migration-scheduling algorithm called Utilization Variation Prediction in One Cycle (UVPOC). They also developed a two-level scheduler mechanism that monitors real-time resources on a global scale. The simulator results proved the algorithm's practicality and showed that the algorithm improved the global resource utilization of containers and VMs. Chhikara (Chhikara et al., 2021) presented a container management technique to increase host resource consumption and avoid server Overload or Underload. Ma et al. (Ma et al., 2020) tackled the challenge of container migration in edge computing to achieve load balancing while considering migration costs. In a proposal by Zhong and Buyya (Zhong and Buyya, 2020), a three-pronged approach to heterogeneous task allocation was introduced for cost-effective container orchestration. This approach focused on optimizing resource utilization and elastic instance pricing. The three components included enabling diverse workload configurations for efficient initial container placement, adjusting cluster sizes to accommodate changing workloads using auto-scaling techniques, and implementing a rescheduling feature to shut down unnecessary VM instances, thereby saving costs and reallocating critical tasks without interrupting their progress.

Hanafy et al. (Hanafy et al., 2018) proposed selection criteria for containers and hosts, achieving improved energy efficiency and meeting SLAs compared to other solutions. Tang et al. (Tang et al., 2019) framed the problem of container migration for mobile application jobs in Fog Computing (FC) as a large-scale Markov Decision Process (MDP). They defined a system model with a cost function encompassing factors such as delay, power consumption, and migration cost. Subsequently, they introduced container migration techniques based on deep Q-learning to make rapid decisions. Their improvements included enhancing random action selection during exploration and refining the Deep Neural Network (DNN) training approach for Q-network updates. The results indicated that their proposed algorithm reduced latency, power consumption, and migration costs.

Ghribi (Ghribi, 2014) created a framework for cloud computing workload prediction and VM deployment. The platform comprised an estimating module that predicted the incoming data centre demand, as well as schedulers that determined the best way to allocate VMs to PMs. Spicuglia (Spicuglia et al., 2015) released OptiCA, which facilitated the building of massive data intensive applications suitable for CaaS. The suggested technique's goal was to achieve the specified efficiency under any given energy and core restraints. Dong (Dong et al., 2014) proposed the most efficient server first (MESF) job scheduling technique, which assigned work to the most energy efficient computers first. To optimise data centre energy efficiency, the scheduler analysed resource allocation information and machine energy profiles to determine where jobs should be scheduled. Yaqub (Yaqub et al., 2014) emphasised the distinctions between IaaS and PaaS implementation methodologies. The PaaS deployment approach was built on OS-level containers that is responsible for hosting a variety of software services. Moreover, they noted that PaaS DC frequently underutilized the underlying infrastructure owing to unforeseen software application needs and the fluctuating quantity of containers that are provided and de-provisioned. As a result, their primary contribution was to describe the server consolidation problem as a multidimensional bin-packing problem and solve it using metaheuristics such as Late Simulated Annealing, Late Acceptance, Simulated Annealing, and Tabu Search. Shi (Shi et al., 2018b) employed NSGA-II based algorithm. They explored non-dominated solutions to determine which hosts, not overloaded, would undergo container migration during each reallocation interval. For each of these selected hosts, they employed the First-Fit (FF) policy to determine the destination hosts for their containers. Piraghaj (Piraghaj et al., 2015b) presented a two-stage framework. First, the framework used static Underload and Overload levels to identify Overloaded or Underloaded host containers that should be relocated. The containers to be migrated are then determined by two algorithms, "Maximum Correlation (MCor)" and "Maximum Usage (MU)." The framework then mapped the migrating containers to destination VMs using one of the bin-packing host selection techniques: Random, First Fit (FF), and Least-Full hosts. Table 5.1 shows the comparison of work done in the domain of container consolidation.

Table 5.1: Comparison of work done in the domain of container consolidation

Work	**Advantages**	**Disadvantages**
Developed a container movement strategy for container consolidation (Chen et al., 2018).	The algorithm has a positive impact on energy consumption.	They did not combine the VM and container consolidation algorithms.
Proposed (U-Chupala et al., 2017) a container rebalancing approach with a rebalancing process working alongside a scheduling process.	The container rebalancing method increases LXC cluster utilization while maintaining minimal interference with the scheduling process.	They have not tested the effectiveness of the proposed approach.

Proposed (Zheng et al., 2021) a container resource migration scheduling algorithm called UVPOC.	Created a two-level scheduler system to keep track of global real-time resources, using LTSM to forecast the trend of dominating resource utilisation and decide whether to undertake container migration or VM pre-boot in terms of the provisioning of resources.	They did not consider to test the method with large number of datasets. Their defined dataset consists of small number of VMs and PMs.
Presented a container management technique to increase host resource consumption and avoid server Overload or Underload (Chhikara et al., 2021).	The best-fit container placement methodology is used to handle host overload or underload issues by locating the optimum destination host for container placement.	However, while deciding on the appropriate host for the container, the authors did not account for migration costs.
Addressed the edge computing container migration problem in order to perform load balancing while bringing migration costs into consideration (Ma et al., 2020).	The proposed approach increased total service performance.	They did not consider the mobility.

Suggested (Zhong and Buyya, 2020) a three-pronged heterogeneous scheduling approach for cost-effective orchestration of containerized services through the use of resources optimisation and elastic instance pricing.	The proposed technique reduces the overall significant cost for different types of cloud workload patterns.	They did not consider to compare the proposed algorithm with the pre-existing methods.
Suggested container and host selection criteria (Hanafy et al., 2018).	Outperform the other algorithms energy and service level agreement commitment.	The dataset used in this approach consists of small number of VMs and PMs.
Characterised the container migration problem of mobile application jobs in FC as a large-scale MDP problem (Tang et al., 2019).	They introduced deep Q-learning-based container migration techniques. To accomplish quick decision making, they improve random action selection in exploration and DNN training approach in Q-network update.	Not mentioned

Proposed (Ghribi, 2014) a framework for cloud computing workload prediction and VM deployment.	First, an estimating module was included to forecast the DC's incoming load. Then, schedulers were created to identify the best way to allocate VMs to PMs.	The efficiency of the suggested algorithms was not validated.
Released OptiCA, which facilitates the building of massive data intensive applications suitable for CaaS (Spicuglia et al., 2015).	The suggested technique's goal is to achieve the specified efficiency under any given energy and core restraints.	The used dataset is very small which doesn't validate the results when the number of PMs, VMs and containers are huge in number.
Proposed (Dong et al., 2014) the most efficient server first (MESF) job scheduling technique, which assigns work to the most energy efficient computers first.	Suggested MESF strategy significantly reduces energy consumption	They did not consider the SLA violations metrics.

Emphasised the distinctions between IaaS and PaaS implementation methodologies (Yaqub et al., 2014).	Primary contribution is to describe the server consolidation problem as a multidimensional bin-packing problem and solve it using metaheuristics such as Late Simulated Annealing, Late Acceptance, Simulated Annealing, and Tabu Search.	They did not consider to compare the proposed techniques with the already existing techniques for validation.
Employed NSGA-II based algorithm for container consolidation problem (Shi et al., 2018b).	They look for non-dominant methods that determine which non-overloaded hosts are made to transfer their containers at each relocation.	They did not consider the SLA violations and the dataset used for experimentation is small in number.
Presented (Piraghaj et al., 2015b) a two-stage framework for energy efficient container consolidation in cloud data centers.	The containers to be migrated are then determined by two algorithms, "MCor" and "MU." The framework then maps the migrating containers to destination VMs using one of the bin-packing host selection techniques: Random, FF, and Least-Full hosts.	The major drawback is that they did not consider the use of metaheuristic algorithms.

The techniques that have been mentioned previously are not be suitable

for the present scenario, as they are incomplete or insufficient for solving the complex container placement and consolidation problems in cloud data centers. In order to find the most optimal solutions for these problems, it is necessary to use AI-based meta-heuristic algorithms. These algorithms have proven to be highly effective in solving the dynamic and large-scale nature of these problems, which traditional optimization techniques struggle to handle. Therefore, the research needs to explore and implement these advanced techniques to ensure efficient and effective container placement and consolidation in cloud data centers. The container placement and consolidation problems are NP hard (Al-Moalmi et al., 2021) (Hussein et al., 2019).
To prove that the container consolidation problem is NP-hard using 3SAT, it is necessary to reduce 3SAT to the container consolidation problem. In other words, it is necessary to show that an instance of 3SAT can be transformed into an instance of the container consolidation problem in polynomial time.
The Container Consolidation Problem:
The container consolidation problem involves consolidating a set of containers of different sizes onto a minimum number of servers while ensuring that the total sizes of containers on each server do not exceed their capacities.
Formally, given a set of containers with sizes C = $c_1, c_2, ..., c_n$ and a set of servers with capacities S = $s_1, s_2, ..., s_m$, the task is to find a consolidation of containers onto servers such that:
1. The total size of containers on each server does not exceed its capacity.
2. The number of used servers is minimized.
Proof
Given an instance of 3SAT with n variables and m clauses, it is necessary to construct an instance of the container consolidation problem.
1. For each variable xi in the 3SAT instance, create two containers with sizes $ci_1 = 2$ and $ci_2 = 1$. These two containers represent the truth values "true" and "false" for the variable x_i. 2. For each clause C_j = (a, b, c) in the 3SAT instance, create a server with capacity $s_j = 4$.

3. Now, for each literal in a clause, assign the corresponding container to the server representing that clause. For example:
- If the literal is x_i, assign container ci_1 to the server corresponding to the clause C_j.
- If the literal is negation of x_i, assign container ci_2 to the server corresponding to the clause C_j.
Now, let's analyze the reduction:
If the 3SAT instance is satisfiable:
If the 3SAT instance is satisfiable, there is a truth value assignment to the variables that fulfils all of the clauses. In the corresponding container consolidation instance, it is necessary to assign containers representing "true" for the variables set to true and containers representing "false" for the variables set to false. Since each server's capacity is 4, and each container has a size of 1 or 2, the assignment of containers to servers will be feasible.
If the container consolidation instance has a feasible solution:
If the container consolidation instance has a feasible solution, it means there exists an assignment of containers to servers such that each server's capacity is not exceeded. This implies that for each clause, at least one literal in that clause evaluates to true. Therefore, the corresponding 3SAT instance is satisfiable.
Since the reduction is valid and 3SAT is known to be NP-hard, the container consolidation problem is also NP-hard. Therefore, the container consolidation problem is NP-hard when using 3SAT as the reference problem.
In some cases, it may be possible to break a container-to-VM mapping problem into smaller sub problems that can be solved using dynamic programming. For example, if the problem involves mapping a large number of containers to a smaller number of VMs, the problem can be broken down into smaller sub problems by mapping groups of containers to subsets of VMs, and then combine the solutions to these sub problems to get the overall optimal mapping. However, there may be other mapping problems where dynamic programming (de Souza et al., 2022) is not the best approach. If the problem involves mapping containers to a large number of

VMs involving large number of constraints, it may be more appropriate to use a different optimization approach, such as linear programming, constraint programming, or heuristics. Linear programming is a useful tool for a wide range of optimisation issues, it may not be appropriate for every circumstance, including the container virtual machine (VM) mapping problem. (Karmakar et al., 2022). The container VM mapping problem involves mapping a set of containers to a set of VMs while minimizing the overall resource usage and meeting certain constraints such as minimizing the number of VMs used or ensuring that each container has enough resources. This problem is a combinatorial optimization problem, which means that it involves finding the best combination of mappings from a large set of possible combinations. Linear programming is also not well-suited for combinatorial optimization problems because it assumes that the solution space is continuous and can be represented by a linear equation. However, in the container VM mapping problem, the solution space is discrete, meaning that it involves selecting a subset of available VMs for each container. Therefore, other optimization techniques such as heuristic approaches, genetic algorithms, or integer linear programming may be more suitable for solving the container VM mapping problem. These techniques can handle the combinatorial nature of the problem and find a good solution within a reasonable time frame. The ILP shows a major issue with the energy consumption when compared with other strategies (M. K. Patra et al., 2022). Metaheuristic algorithms, on the other hand, are designed to handle such complex and dynamic problems by iteratively exploring the solution space and adjusting the search parameters based on the quality of the solutions found. The approaches have been used on virtual machine placement and migration. As containerization continues to gain popularity, research in this area is likely to remain a key focus for cloud computing researchers and practitioners. The research needs to focus on reducing the energy consumption of the cloud data center without affecting the SLA by developing optimized container placement and consolidation strategies. As it remains the key research area due to the growing popularity of containerization as a lightweight alternative to virtualization in cloud computing environments. Moreover, these approaches need to be tested on both homogenous and heterogenous environments and on large dataset i.e., more numbers of VMs and PMs.

5.3 Methodology

Container consolidation in cloud data centers is a strategy employed to enhance the efficiency of computing resource utilization by bundling several containerized applications onto a reduced number of physical machines. This approach empowers data center operators to optimize their infrastructure efficiency while curbing expenses linked to power, cooling, and physical footprint. Through container consolidation, data centers can curtail the necessity for numerous physical servers to execute a specific suite of applications, thereby diminishing the energy demand required for server operation and cooling. Additionally, by optimizing resource utilization, data centers can make better use of their existing hardware, reducing the need for new infrastructure investments.

5.3.1 Problem Formulation of Container Migration in Cloud Data Center

$$P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t) \tag{5.1}$$

$P_{dc}(t)$ is the usage of power of the DC at time t, N_S is the number of servers, $P_i(t)$ is the power consumption of server i at time t.

For each i server, CPU usage $(U_{i,t})$ is equal to $\sum_{j=1}^{N_{VM}} \sum_{k=1}^{N_c} U_{C_{(k,j,i)}(t)}$ and the usage of power of the server is estimated through the below equation:

$$P_i(t) = P_i^{idle} + \left(P_i^{max} - P_i^{idle}\right) \times U_{i,t} \qquad N_{vm} > 0 \tag{5.2}$$

$$0 \qquad N_{vm} = 0$$

P_i^{idle} is the idle usage of power of server i at time t, P_i^{max} is the maximum usage of power of server i at time t and $U_{i,t}$ CPU Utilization percentage of server i at time t.N_{VM} is the number of VMs. N_c is the number of containers.

The SLA is only breached if the VM that hosts the container does not get the specified amount of CPU. The SLA measure is defined in this context as the fraction of the gap in between sought and allotted quantity of CPU for each VM.

$$SLA = \sum_{i=1}^{N_s}\sum_{j=1}^{N_{vm}}\sum_{p=1}^{N_v} \frac{CPU_r\left(vm_{j,i,}tp\right) - CPU_a(vm_{j,i,}\ tp)}{CPU_r(vm_{j,i,}\ \ tp)} \tag{5.3}$$

$CPU_r\left(vm_{j,i,}tp\right)$ represents required CPU by VM_j on server i at time t_p, $CPU_a(vm_{j,i,}\ tp)$ represents the CPU amount assigned to VM_j at time t_p, N_s denotes number of servers, N_{vm} denotes the number of VMs, N_v denotes number of SLA breaches. The purpose of this study was to minimize DC energy consumption by exploring alternative host selection methods for container consolidation in cloud DC. The research problem is formulated as follows to reduce the power usage of a DC with M containers, N VMs, and K servers:

$$minP_{dc}\left(t\right) = \sum_{i=1}^{N_S} P_i(t) \tag{5.4}$$

Consider the following constraints:

$$\sum_{j=1}^{N_{vm}} U_{vm_{j,i}}\left(t\right) < S_{(i,r)},\ \forall i \in \left[1, N_s\right],\ \forall r \in CPU \tag{5.5}$$

$$\sum_{j=1}^{N_{vm}}\ vm_{(j,i,r)} < S_{(i,r)},\ \forall i \in \left[1, N_s\right],\ \forall i \in BW,\ Memory,\ Disk \tag{5.6}$$

$$\sum_{k=1}^{N_c} U_{C_{(k,j,i)}} < vm_{(j,i,r)},\ \forall j \in \left[1, N_{vm}\right],\ \forall i\ \in\ \left[1,\ N_s\right],\ \ \forall r \in CPU \tag{5.7}$$

$$\sum_{k=1}^{N_c}\ C_{(k,j,i,r)} < vm_{(j,i,r)},\ \forall j \in \left[1, N_{vm}\right],\ \forall i \in \left[1,\ N_s\right],\ \forall r\ \in \left[BW,\ Memory,\ Disk\right] \tag{5.8}$$

$U_{vm_{j,i}}\left(t\right)$ represents CPU utilization of VM j on server i at time t, $S_{(i,r)}$ represents server i capacity for resource r, $vm_{(j,i,r)}$ represents the resource r capacity of VM j on server i, $U_{C_{(k,j,i)}}$ denotes CPU utilization of container k on VM j and server i, $C_{(k,j,i,r)}$ represents the resource r capacity of container k on VM j, server i.

5.3.2 Process Flow

This section focuses on the process flow of designing and developing container migration algorithms/host selection algorithms for energy efficiency in cloud data center. Two algorithms, Energy Efficient Ant Colony Optimization (EEACO) and Energy Efficient Firefly Optimization (EEFFO) are proposed, implemented and compared with the pre-existing algorithms Random Host Selection (RHS), First Fit Host Selection (FFHS), Last Fit Host Selection (LFHS) and Corelation based Host Selection (CorHS) [78]. RHS algorithm selects a host at random from the available host list that has at least one VM that can host the container. The FFHS algorithm chooses the first host from the available hosts list that fulfils the resource requirements for the container. LFHS algorithm selects the first host from a sorted (descending order by CPU utilization) list of available hosts that fulfils the container resource requirements. The CorHS method calculates the correlation of CPU workload histories from containers and hosts. If it determines that there is no such the past, it just utilises LFHS. If the load history can be verified, the first host that matches the first correlation condition and has a VM that can handle the container is selected. This approach yields a value known as "Pearson's correlation coefficient," which measures the degree of dependence among two variables. According to Pearson's analysis, if there are two random variables P and Q with n samples denoted by p_i and q_i the correlation coefficient is calculated using equation 5.9 where $\bar{P}$ and $\bar{Q}$ denotes the sample means of P and Q respectively and r_{py} varies the range [-1, +1].

$$r_{py} = \frac{\sum_{i=1}^{n} (p_i - \bar{p})(q_i - \bar{q})}{\sqrt{\sum_{i=1}^{n} (p_i - \bar{p})2 \sum_{i=1}^{n} (q_i - \bar{q})2}} \tag{5.9}$$

The closer P and Q's correlation coefficients are near +1, the more is the probability of the variables to reach peak/valley values together. In other terms, if the container workload is unrelated to the host load, the container is less prone to OL the host. The VM is chosen using the First-Fit technique in all of the approaches described above. Host status module in Cloudsim 4.0 as shown in Figure 5.1 determines the OL and UL host machines. This is accomplished by the Host OL/UL Detector component, which monitors the host's resource use after each fixed time period (say 5 minutes). If the host is underutilized, the detector communicates to the consolidation module with host ID and containers IDs running on it. In the case of OL host status, the detector is responsible for sending a request to the container selector (CS) component for activation (Piraghaj et al., 2015b).

$$Host\,Status = \;Overloaded \quad if \quad U_{i,t} > T_{ol} \quad Underloaded \quad if \quad Ui,t < T_{ul} \tag{5.10}$$

$U_{(i,t)}$ represents percentage utilization of CPU of Server i at time t. The CS when invoked in case of OL host is done using the following algorithms:
MU (Piraghaj et al., 2015b): In MU, the container with the highest CPU utilization is selected and put to the migration list.
MCor algorithms (Piraghaj et al., 2015b): In MCor, the container with the highest correlation with the host load is selected and chosen for migration. The Container Migration List (CML) keeps data about the selected container.

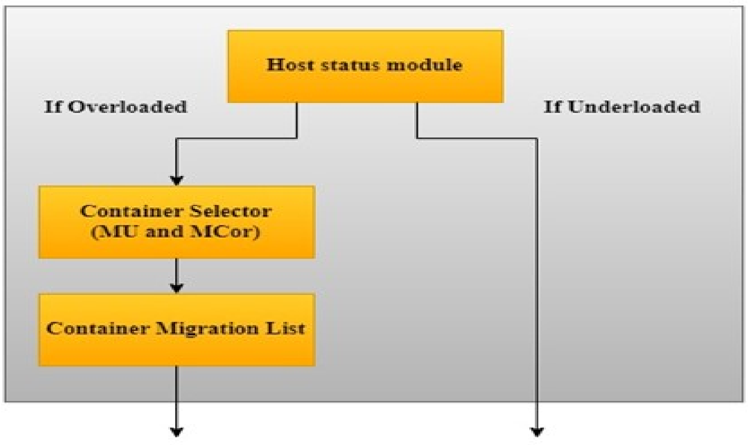


Figure 5.1: Host status module and its components in Cloudsim 4.0 (Piraghaj et al., 2015b)

The consolidation module as shown in Figure 5.2 is responsible for the migration of containers based on the output given by the host status module. The host status module consists of the OL host list and the OL destination selector. The OL host list contains hosts categorized as "OL" based on their current state, while the OL destination selector employs the

Host Selection Algorithm (HSA) to identify appropriate destinations for containers within the received CMLs (Container Migration Lists).

The VM creation component is in charge of predicting the number of VMs required in the subsequent phase. This calculation is derived from the count of containers for which the overflow destination selector failed to identify an appropriate host or virtual machine. The primary objective of this component is to create the most substantial VMs feasible on under-utilized hosts and then allocate containers to these newly created VMs. If any containers remain unallocated, it randomly selects a host from the list of inactive hosts and initiates VMs on that host, ensuring no containers are left without a migration destination.

In contrast, the UL destination selector utilizes an HSA (Host Selection Algorithm) to ascertain the best possible destination for containers that originate from under-loaded (UL) hosts. This decision takes into account not only the list of under-loaded hosts but also the choices made by the OL destination selector. In cases where this component identifies a suitable destination for each container on an under-loaded host, it communicates these destinations to the VM Host Migration Manager component, along with the destinations chosen by the overload destination selector. Furthermore, it supplies the host identification to the under-loaded host deactivation component, which then proceeds to shut down under-loaded hosts once all containers have been moved.

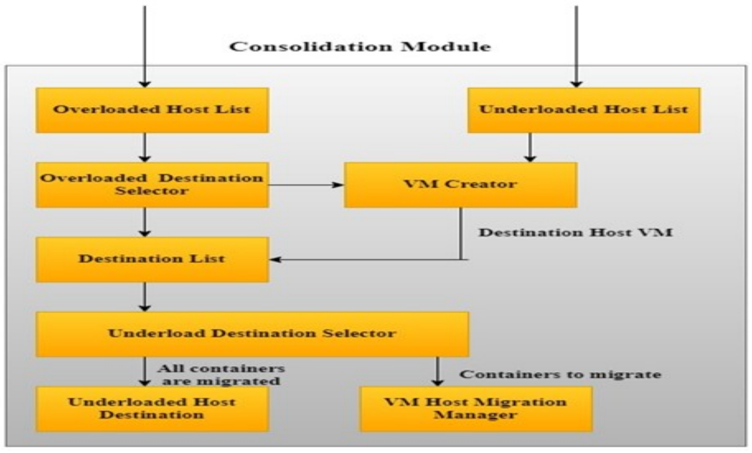

Consolidation Module
Overloaded Host List
Underloaded Host List
Overloaded Destination Selector
VM Creator
Destination Host VM
Destination List
Underload Destination Selector
All containers are migrated
Containers to migrate
Underloaded Host Destination
VM Host Migration Manager

Figure 5.2: Consolidation Module and its components in Cloudsim 4.0 (Piraghaj et al., 2015b)

5.3.3 Algorithms

The algorithm took the placement of containers on virtual machines (VMs) as input and aimed to migrate containers to the best host nodes. It started by initializing variables and updating global pheromones. The loop ran until a specified number of iterations were completed. Within the loop, the algorithm allocated containers to hosts based on equation 5.12 and updated the allocation probability if the host was underloaded. It then checked the energy consumption of each ant and updated the global pheromones accordingly. Following the conclusion of each iteration, local pheromone levels are refreshed or adjusted. By combining both local and global pheromones, ACO algorithms effectively exploited promising paths found by individual ants while simultaneously exploring new paths in the search space. This balance between exploitation and exploration enabled ACO to efficiently navigate complex problem landscapes and converge towards near-optimal or optimal solutions (Yan et al., 2018). Finally, the best allocation was returned based on the minimum energy consumption.

Algorithm 7 Energy Efficient Ant Colony Optimization Host Selection Algorithm (EEACO)

```
1: Input: Container Placement on VMs using DFFLSM
2: Output: Migration of containers to best host nodes
3: Initialize k, len, $\tau_{ij}$ , $minPower = MaxValue$, $bestAllocationMap = null$
4: update global pheromones according to Equation 5.11
5: **while** $iterationNum < len$ **do**
6:     **for** $allant \in antList$ **do**
7:         **for** $all \quad Container \in ContainerMigrationList$ obtained from overloaded node **do**
8:             allocate Container to host according to P'ij in Equation 5.12
9:             update P'ij according to Equation 5.12
10:        **end for**
11:        **for** each container on ContainerMigrationList **do**
12:            **if** No Underloaded Node and Overloaded Node left **then**
13:                Activate any inactive node
14:                Transfer the container to the node
15:            **end if**
16:        **end for**
17:    **end for**
18:    **for** $forallant \in antList$ do **do**
19:        **if** $powerSum < minPower$ then **then**
20:            $minPower \leftarrow powerSum$
21:            $bestAllocaton \leftarrow allocation$
22:            update global pheromones according to Equation 5.11
23:        **end if**
24:    **end for**
25:    update local pheromones according to Equation 5.13
26: **end while**
27: Return best allocation
```

The value of parameters used in EEACO is shown in Table 5.2.

If the power consumption of the a^{th} allocation results is lower than minPower, the global pheromone is updated and the powerSum is set to minPower

$$\tau_{ij} = \tau_{ij} + \frac{Q}{Power_a} \tag{5.11}$$

Here, $Power_a$ displays the power usage of the method discovered by a^{th} ant. Q is an information intensity constant that indicates the quantity of

Table 5.2: Parameters of EEACO

Parameter	**Value**
Q	100
α	1
ρ	0.7
len	50
k	10

pheromone left by the ants after one repetition. τ_{ij} denotes the released pheromone.

$p\prime_{ij}= ((\tau_{ij}^{\alpha})/d_{ij})/ \sum_{i=1}^{n}(\tau_{ij}^{\alpha})/d_{ij}$

(5.12)

$p\prime_{ij}$ denotes the probability of container i to be allocated to host j. α is the inspiration factor.

if the host is underloaded, d_{ij} is equal to number of underloaded hosts otherwise if it is a case for overloaded hosts, d_{ij} in that case is equal to 1.

With such adjustments, the likelihood of container migration to under loaded hosts is substantially lower than that to overloaded hosts, potentially reducing the phenomena of UL hosts.

$$\tau_{ij} = (1-\rho).\tau_{ij} + \delta\tau_{ij} \tag{5.13}$$

Here, ρ is the pheromone volatility factor and $\delta\tau_{ij}$ is the pheromone increment, which is defined as:

(5.14)

$$\Delta\tau_{ij} = \left\{ \begin{array}{l} \left\{ \sum_{a=1}^{k} \frac{Q}{Power_a}, if\ Containerj\ is\ allocatedto\ host\ i \right\} \\ \qquad 0, else \end{array} \right.$$

Here, k is the number of ants.

Time and Space Complexity

Step 1: Initialization step only performs a few variable assignments hence

the complexity *O(1)*.
Step 2: Updating global pheromones step involves iterating over each pair of nodes, which takes $O(n^2)$ time.
Step 3: The loop runs for “len” iterations, and each iteration involves iterating over all“”k” ants, and for each ant, iterating over al“ ”m” containers in ContainerMigrationList.
Step 4: The allocation and pheromone update steps take $O(n^2)$ time each, so the total time complexity of the loop is $O(len * k * m * n^2)$.
Step 5: Transferring containers step involves transferring the containers from overloaded nodes to underloaded nodes, which takes *O(m)* time.
Step 6: Updating local pheromones this step involves iterating over each pair of nodes, which takes $O(n^2)$ time.
Therefore, the total time complexity of EEACO is $O(len*k*m*n^2)$, where “len” is the number of iterations, “k” is the number of ants, “m” is the number of containers, and “n” is the number of nodes.
O(1) for variable initialization, $O(n^2)$ for the global pheromone matrix, $O(m*n*k)$ for the allocation matrix and the local pheromone matrix and $O(k)$ for the antList. So, Overall space complexity is $O(m * n * k + n^2)$ where m is the number of containers, n is the number of nodes and k the number of ants.
The algorithm initialized three arrays and ran a loop for a specified number of generations. For each overloaded node, it computed energy consumption and execution time using specific equations and calculated the index of attraction. The algorithm then sorted the index of attraction in ascending order according to energy consumption and calculated the distance. It sorted the host load in descending order and transfered the host load to the destination with the nearest energy consumption. Finally, the distance is updated, and the loop continued until the maximum number of generations is reached.

Algorithm 8 Energy Efficient Firefly Optimization Host Selection Algorithm (EEFFO)

```
1: Input: Container Placement on VMs using DFFLSM
2: Output: Migration of containers to best host nodes
3: Initialize Overload, Underload, Active
4: while $t < maxgeneration$ do
5:     for each Overloaded Node and Underloaded node do
6:         Compute Energy Consumption using equation 5.15
7:         $CE[] \leftarrow$ Energy Consumption Value
8:         Calculate Execution Time using equation 5.16
9:         $HET[] \leftarrow$ Execution Time value
10:    end for
11:    for each Overloaded Node do
12:        Compute Attractivesness Index as (CE, HET)
13:        $IA[] \leftarrow$ Attractiveness Index
14:    end for
15:    Sort $IA[]$ in an ascending order according to $EC$ values
16:    Compute Distance using equation 5.17
17:    Find the host with $CE$ value nearest to the calculated Distance value from the
   sorted $IA[]$
18:    for each container in the CML do
19:        Compute HostLoad of Overloaded Node using CPU usage
20:        $HostLoad[] \leftarrow$ HostLoad value
21:    end for
22:    Sort $HostLoad[]$ in a descending order
23:    Pick the first position element of sorted $IA[]$ as the destination
24:    Transfer the first $HostLoadOverloaded[]$ container from ContainerMigrationList
   to the first $IAOverloaded[]$ element.
25:    for each Underloaded Node do
26:        Compute Attractiveness Index (CE, HET)
27:        $IAUnderloaded[] \leftarrow$ Attractiveness Index
```

```
28:     end for
29:     Sort $IAUnderloaded[]$ in an ascending order according to $EC$ values
30:     Compute Distance using equation 5.17
31:     Find the host with CE value nearest to the calculated distance value from the
  sorted $IAUnderloaded[]$
32:     for each container on ContainerMigrationList do
33:         Compute HostLoad of Underloaded Node using CPU usage
34:         $HostLoadUnderloaded[] \leftarrow$ HostLoad value
35:     end for
36:     Sort HostLoadUnderloaded[] in a descending order
37:     Pick the first position element of sorted IAUnderloaded[] as the destination
38:     Transfer the first HostLoadUnderloaded[] container from ContainerMigrationList
  to the first IAUnderloaded[] element.
39:     for each container on ContainerMigrationList do do
40:         if no underloaded and overloaded node found then
41:             Activate any inactive node
42:             Transfer the container to the node
43:         end if
44:         Update distance according to equation 5.18
45:     end for
46: end while
```

$$CE_i = \frac{(\sum_{j=1}^{v}\sum_{k=1}^{u} CPU_{i,j,k})(\sum_{j=1}^{v}\sum_{k=1}^{u} mu_{i,j,k})}{M} \tag{5.15}$$

In equation 5.12, v represents the number of VMs running on the i^{th} node and u is the number of containers allocated to v VMs. $CPU_{i,j,k}$ and $mu_{i,j,k}$ are the processor and the memory usage of k containers operating in j^{th} VM of the i^{th} node respectively and M is the number of memory units.

$$HET_i = \sum_{j=1}^{v}\sum_{k=1}^{u} HET_{i,j,k} \tag{5.16}$$

Where $HET_{i,j,k}$ is the execution time of k containers running in j^{th} VMs on the i^{th} node.

$$Distance = Avg(IAmid, IAmax) \tag{5.17}$$

Where IA_{mid}, IA_{max} are the middle and the maximum vales from the AI list.

$$(Distance)^{t+1} = (Distance)^{t} + CPUutilization \quad w.r.t \quad to \quad VM + \epsilon \tag{5.18}$$

Where $(Distance)^{t+1}$ and $(Distance)^{t}$ are the distance at time t and t+1. ϵ is the gaussian distribution error.

Time and Space Complexity

Step 1: Line 4 runs for MaxGeneration times, so its time complexity is *O(MaxGeneration)*.

Step 2: Lines 5-10 loop over each OverloadedNode and UnderloadedNode, so their time complexity can be approximated as *O(n)*, where n is the total number of nodes.

Step 3: Lines 12-114 calculate Attractiveness Index for each OverloadedNode, so their time complexity can be approximated as *O(n)*.

Step 4: Line 15 sorts IAOverloaded[], which has a length of n, so its time complexity is *O(n log n)*.

Step 5: Lines 16-17 compute the distance and find the closest host, so their time complexity is *O(n log n)*.

Step 6: Lines 18-21 loop over each container on the ContainerMigrationList, so their time complexity can be approximated as *O(c)*, where c is the number of containers to be migrated.

Step 7: Line 22 sorts HostLoadOverloaded[], which has a length of c, so its time complexity is *O(c log c)*.

Step 8: Lines 23-24 transfer the container with the highest HostLoad from Overloaded Node to the selected destination, so their time complexity is *O(1)*.

Step 9: Line 25 updates the distance value, so its time complexity is *O(1)*.

Step 10: Lines 26-29 compute attractiveness index for each UnderloadedNode, so their time complexity can be approximated as *O(n)*.

Step 11: Line 30 sorts IAUnderloaded[], which has a length of n so its time complexity is *O(n log n)*.

Step 12: Lines 31-32 compute the Distance and find the closest host, so their time complexity is *O(n log n)*.

Step 13: Lines 33-36 loop over each container on the ContainerMigrationList, so their time complexity can be approximated as *O(c)*.

Step 14: Line 37 sorts HostLoadUnderloaded[], which has a length of C, so its time complexity is *O(c log c)*.

Step 15: Lines 38-39 transfer the container with the highest HostLoad from UnderloadedNode to the selected destination, so their time complexity is *O(1)*.

Step 16: Line 40 updates the distance value, so its time complexity is *O(1)*.

Step 17: Lines 41-46 loop over each container on the ContainerMigrationList and try to activate any inactive node, so their time complexity can be approximated as *O(c)*.

Overall, the time complexity of this algorithm can be approximated as $O(MaxGeneration * n * (logn + clogc))$, where n is the number of nodes, c is the number of containers to be migrated, and MaxGeneration is the maximum number of iterations. Overload, Underload, Inactive variables are used to store the set of overloaded, underloaded, and inactive nodes, respectively. The space required to store these variables is proportional to the number of nodes in the data center, which is typically a constant value for a given data center. Therefore, the space complexity for these variables can be considered as O(1). CE[] and HET[] arrays store the energy consumption and execution time values, respectively, for each overloaded and underloaded node. The space required to store these arrays is proportional to the number of nodes in the data center, which is typically a constant value for a given data center. Therefore, the space complexity for these arrays can be considered as O(n). IAOverloaded[] and IAUnderloaded[] arrays store the attractiveness index values for each overloaded and underloaded node, respectively. The space required to store these arrays is proportional to the number of nodes in the data center, which is typically a constant value for a given data center. Therefore, the space complexity for these arrays can be considered as O(n). HostLoadOverloaded[] and HostLoadUnderloaded[] arrays store the host load values for each overloaded and underloaded node, respectively. The space required to store these arrays is proportional to the number of containers in the data center, which can be considered as O(c). ContainerMigrationList is a list of containers that need to be migrated. The space required to store this list is proportional to the number of containers in the data center, which can be considered as O(c). Distance variable to store the estimated distance between overloaded and underloaded nodes. The space required to store this variable is proportional to the number of nodes in the data center, which is typically a constant value for a given data center. Therefore, the space complexity for this variable can be considered as O(1). Therefore, the total space complexity of the algorithm is the sum of the space required for all these variables, which is $O(n + c)$, where n is the number of nodes and c is the number of containers.

5.4 Experimental Setup

The experiments have been conducted in Cloudsim 4.0 with i5-2.4GHz processor and 8 GB RAM. Table 5.3 shows the objectives and various sets of experiments that were conducted for different combinations of overload (OL), Underload (UL) and Container selection (CS) algorithms.

Experimental set up was done for two different scenarios. Scenario 1 was for the homogenous environment in which the configurations of the VM and PM remained same. The second scenario was for heterogenous environment in which the configurations of the VM and the PM were different. The configurations in the homogenous environment and the heterogenous environment are enlisted in table 5.4, table 5.5, table 5.6 and table 5.7.

5.4.1 Scenario 1: Homogenous Environment

In this case, a data center was simulated with 700 homogenous servers and 1000 VMs. A total of 5000 containers are used which belonged to 3 different types. Table 5.4 shows the configurations of the PMs, whereas table 5.5 shows the setups of the VMs and containers.

Table 5.3: Objectives and Experiment sets for Container Consolidation

Set	Objective	CS	UL	OL	Overbooking Factor (percentile)	Correlation Threshold
1	Examine the impact of container consolidation/host selection algorithms on average energy consumption.	MU/MCor	70%	[80%, 90%, 100%]	80	0.5
			[50%, 60%, 70%]	80%		
2	Examine the impact of container consolidation/host selection algorithms on average container migrations.	MU/MCor	70%	[80% 90%, 100%]	80	0.5
			[50%, 60%, 70%]	80%		
3	Examine the impact of container consolidation/host selection algorithms on average VM used.	MU/MCor	70%	[80%, 90%, 100%]	80	0.5
			[50%, 60%, 70%]	80%		
4	Examine the impact of container consolidation/host selection algorithms on average PM used.	MU/MCor	70%	[80%, 90%, 100%]	80	0.5
			[50%, 60%, 70%]	80%		
5	Examine the impact of container consolidation/host selection algorithms on SLA.	MU/MCor	70%	[80%, 90%, 100%]	80	0.5
			50%, 60%, 70%	80%		
6	Examine the impact of container consolidation/host selection algorithms corresponding to different container overbooking factor on average energy consumption, average active VM, average active PM, average container migrations and average overall SLA.	MU	80%	70%	[20,40,80]	0.5

Table 5.4: PM configurations in homogenous environment for Container Consolidation

Physical Machine Setup and Power Models					
Type of Server	**CPU [3GHz] (37274 MIPS/core)**	**Memory (GB)**	**P idle (Watt)**	**P max (Watt)**	**Number**
1	4 Core	64	93	135	700

Table 5.5: VM and container configurations in homogenous environment for Container Consolidation

Type of Containers and VMs							
Type of Container	**CPU MIPS (1 core)**	**Memory (MB)**	**Type of VM**	**CPU [1.5GHz] (18636 MIPS/core)**	**Memory (GB)**	**Number**
1	4658	128				
2	9320	256	1	2	2	1000
3	18636	512				

5.4.2 Scenario 2: Heterogenous Environment

In this case, a DC was simulated with 700 heterogenous servers and 1000 VMs with the set of 3 different servers and 4 different types of VMs. A total of 5000 containers were used which belonged to 3 different types as that of in homogenous environment. The configurations of the PMs are shown in table 5.6 and that of VMs and containers are shown in table 5.7.

Table 5.6: PMs configurations in heterogenous environment for Container Consolidation

PM Setup and Power Models					
Type of Server	**CPU [3GHz] (37274 MIPS/core)**	**Memory (GB)**	**P idle (Watt)**	**P max (Watt)**	**Number**
1	4 Core	64	86	117	234
2	8 Core	128	93	135	233
3	16 Core	256	66	247	233

5.5 Results

In experiments, the following cases have been investigated:

A) Examine the impact of container consolidation/host selection algorithms

Table 5.7: VMs and container configurations in heterogenous environment for Container Consolidation

Types of Containers and VMs						
Type of Container	**CPU MIPS (1 core)**	**Memory (MB)**	**Type of VM**	**CPU [1.5GHz] (18636 MIPS /core)**	**Memory (GB)**	**Number**
1	4658	128	1	2	1	250
2	9320	256	2	4	2	250
3	18636	512	3	1	4	250
			4	8	8	250

on average energy consumption.

B) Examine the impact of container consolidation/host selection algorithms on average container migrations.

C) Examine the impact of container consolidation/host selection algorithms on average VM used.

D) Examine the impact of container consolidation/host selection algorithms on average PM used.

E) Examine the impact of container consolidation/host selection algorithms on SLA.

F) Examine the impact of container consolidation/host selection algorithms corresponding to different overbooking factor on average energy consumption, average active VM, average active PM, average container migrations and average overall SLA.

All the above cases had been considered for both homogenous and heterogeneous environment. Also, each case (except F) was determined for different OL value, UL value and CS algorithms (MU and MCor). In case (F), UL and OL was fixed to 70% and 80% respectively. The CS algorithm used in this case was MU and the overbooking factor was varied. The workload traces from PlanetLab (Park and Pai, 2006) to assess the algorithms considering the simulation setup and parameters indicated in table 5.4, table 5.5, table 5.6 and table 5.7. These traces include 10 days' worth of the testbed's workload from randomly chosen sources, which were gathered during March and April 2011 (Beloglazov and Buyya, 2012). Each container was allocated to a task that contained statistics on CPU usage for one day that was reported every five minutes. The dataset included CPU utilization percentages for over a thousand VMs hosted on servers distributed across more than 500 locations worldwide. The data organization followed a daily folder structure, with each folder containing CPU utilization records for VMs sampled every 5 minutes over the course of a day.

5.5.1 Homogenous Environment

5.5.1.1 Impact of host selection policies (container consolidation) on average energy consumption:

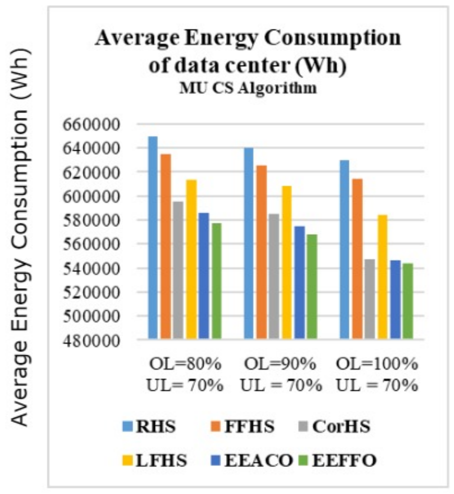
The energy usage of a DC was calculated by using the various host selection policies on different Container Selection (CS) algorithms and thresholds. CS algorithms, MU and MCor are defined in section 5.3.2 and host status thresholds are defined in section 5.4. The impact of host selection algorithms on average energy consumption for different Overload (OL) and Underload (UL) combinations is as shown in figure 5.3 and figure 5.4 respectively. Table 5.8 and table 5.9 shows the comparison of EEFFO and EEACO with other algorithms in terms of average energy consumption in homogenous environment.

Table 5.8: Comparison of EEFFO for average energy consumption reduction in Homogenous environment (in percent)

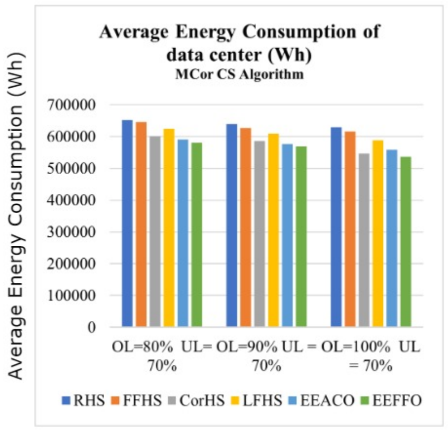
Energy Efficiency in Homogenous environment (in Percentage)						
		RHS	**FFHS**	**CorHS**	**LFHS**	**EEACO**
EEFFO	**MU (changing OL)**	11.75	9.67	1.87	6.24	1.37
EEFFO	**MU (changing UL)**	15.01	15.12	7.14	9.78	4.4
EEFFO	**MCor (changing OL)**	12.29	10.68	2.77	7.42	2.3
EEFFO	**MCor (changing UL)**	14.91	15.36	7.22	10.08	4.42

Table 5.9: Comparison of EEACO for average energy consumption reduction in Homogenous environment (in percent)

Energy Efficiency in Homogenous environment (in Percentage)					
		RHS	**FFHS**	**CorHS**	**LFHS**
EEACO	**MU (changing OL)**	10.52	8.41	0.5	4.93
EEACO	**MU (changing UL)**	11.1	11.22	2.87	5.63
EEACO	**MCor (changing OL)**	10.22	8.58	0.48	5.24
EEACO	**MCor (changing UL)**	10.97	11.44	2.93	5.92



(a)

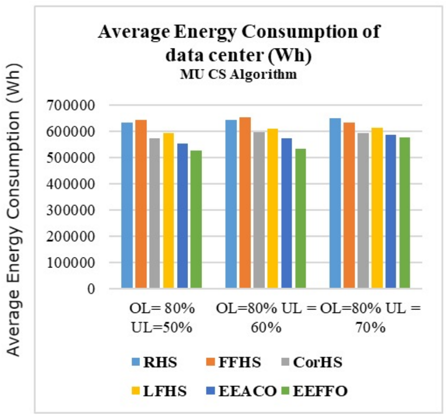


(b)

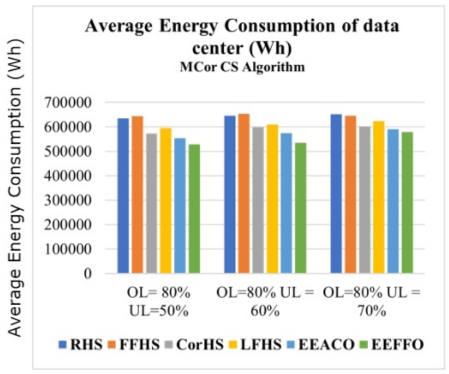
Figure 5.3: Impact of host selection algorithms on average energy consumption for different OL combinations (a) MU and (b) MCor in homogenous environment

5.5.1.2 Impact of host selection policies (container consolidation) on average container migrations:

The average container migration was calculated by using the various host selection algorithm on different CS algorithms and thresholds. Two separate tests were performed based on the OL threshold, UL threshold and different CS algorithms (MU and MCor). For each set, the average container migrations calculated and compared for different OL and UL combinations as shown in figure 5.5 and figure 5.6 respectively.

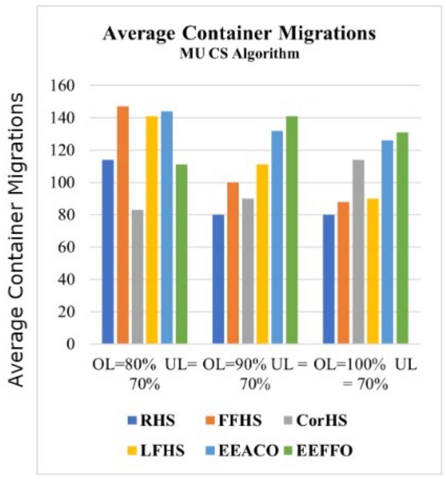


(a)

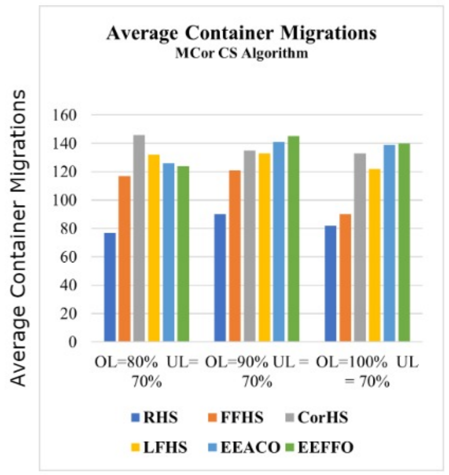


(b)

Figure 5.4: Impact of Host selection algorithms on average energy consumption for different UL combinations (a) MU and (b) MCor in homogenous environment



(a)



(b)

Figure 5.5: Impact of Host selection algorithms on average container migration for different OL combinations (a) MU and (b) MCor in homogenous environment

5.5.1.3 Impact of host selection algorithm (container consolidation) on average VM used:

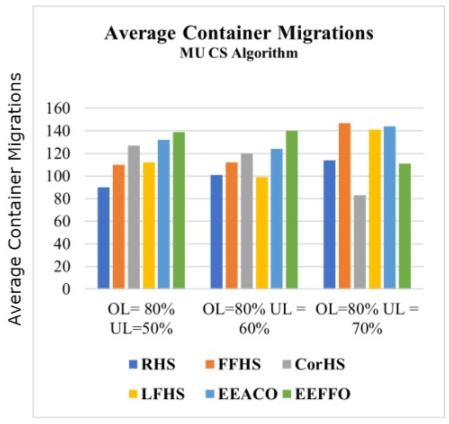
The average VM used were calculated by using the various host selection algorithm on different CS algorithms and thresholds. Two separate tests were performed based on the OL threshold, UL threshold and different CS algorithms (MU and MCor). For each set, the average active VMs used were calculated and compared for different OL and UL combinations as shown in figure 5.7 and figure 5.8 respectively. Table 5.10 and table 5.11 show the comparison of EEFFO and EEACO with other algorithms in terms of average active VM reduction in homogenous environment.

Table 5.10: Comparison of EEFFO average active VM reduction in homogenous environment

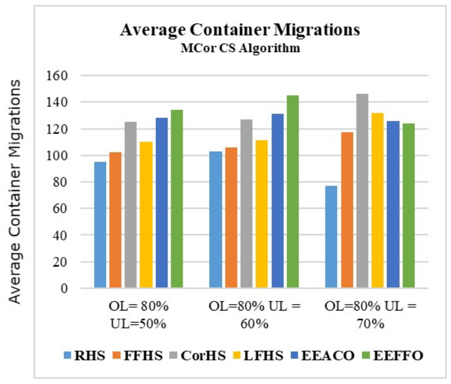
Average Active VM in Homogenous environment (in Percentage)						
		RHS	FFHS	CorHS	LFHS	EEACO
EEFFO	**MU (changing OL)**	8.28	9.65	17.44	19.67	4.49
EEFFO	**MU (changing UL)**	4.08	6.1	13.92	13.12	2.25
EEFFO	**MCor (changing OL)**	7.79	9.18	17.16	19.22	4.17
EEFFO	**MCor (changing UL)**	4.37	5.11	9.31	6.96	1.69

Table 5.11: Comparison of EEACO average active VM reduction in homogenous environment

Average Active VM in Homogenous environment (in Percentage)					
		RHS	FFHS	CorHS	LFHS
EEACO	**MU (changing OL)**	3.97	5.41	13.56	15.9
EEACO	**MU (changing UL)**	1.87	3.93	11.93	11.12
EEACO	**MCor (changing OL)**	3.78	5.23	13.56	15.71
EEACO	**MCor (changing UL)**	2.73	3.48	7.75	5.35

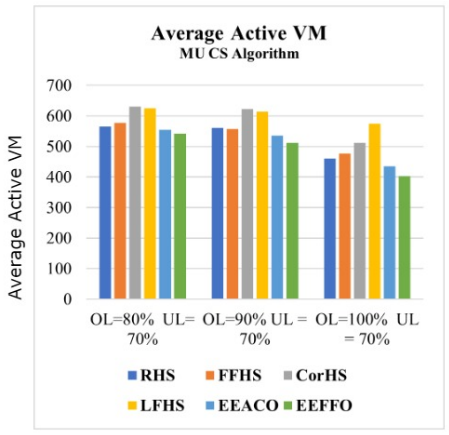


(a)

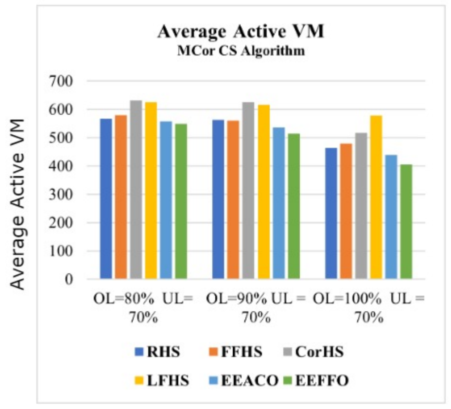


(b)

Figure 5.6: Impact of Host selection algorithms on average container migration for different UL combinations (a) MU and (b) MCor in homogenous environment



(a)



(b)

Figure 5.7: Impact of Host selection algorithms on average VMs used for different OL combinations (a) MU and (b) MCor in homogenous environment

5.5.1.4 Impact of host selection algorithm (container consolidation) on average PM used:

The average PMs used were calculated by using the various host selection algorithms for different CS algorithms and thresholds. Two separate tests were performed based on the OL threshold, UL threshold and different CS algorithms (MU and MCor). For each set, the average PMs used were calculated and compared for different OL and UL combinations as shown in figure 5.9 and figure 5.10 respectively. Table 5.12 and table 5.13 show the comparison of EEFFO and EEACO with other algorithms in terms of average active PMs reduction in homogenous environment.

Table 5.12: Comparison of EEFFO average active PMs reduction in Homogenous environment (in percentage)

Average Active PMs in Homogenous environment (in Percentage)						
		RHS	**FFHS**	**CorHS**	**LFHS**	**EEACO**
EEFFO	**MU (changing OL)**	15.64	23.13	23.17	18.4	5.66
EEFFO	**MU (changing UL)**	8.96	23.22	22.69	16.27	4.47
EEFFO	**MCor (changing OL)**	15.55	27.58	23.17	18.51	5.82
EEFFO	**MCor (changing UL)**	8.58	23.15	22.4	16.34	4.46

Table 5.13: Comparison of EEACO average active PMs reduction in Homogenous environment (in percentage)

Average Active PMs in Homogenous environment (in Percentage)					
		RHS	**FFHS**	**CorHS**	**LFHS**
EEACO	**MU (changing OL)**	7.57	23.13	18.55	13.5
EEACO	**MU (changing UL)**	4.7	19.63	19.07	12.36
EEACO	**MCor (changing OL)**	10.33	23.1	18.43	13.47
EEACO	**MCor (changing UL)**	4.31	19.56	18.78	12.43

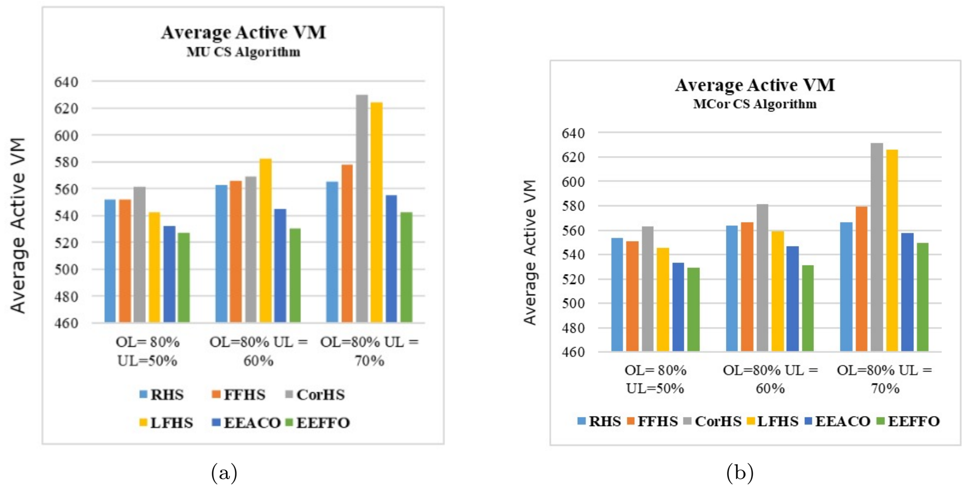


Figure 5.8: Impact of Host selection algorithms on average VMs used for different UL combinations (a) MU and (b) MCor in homogenous environment

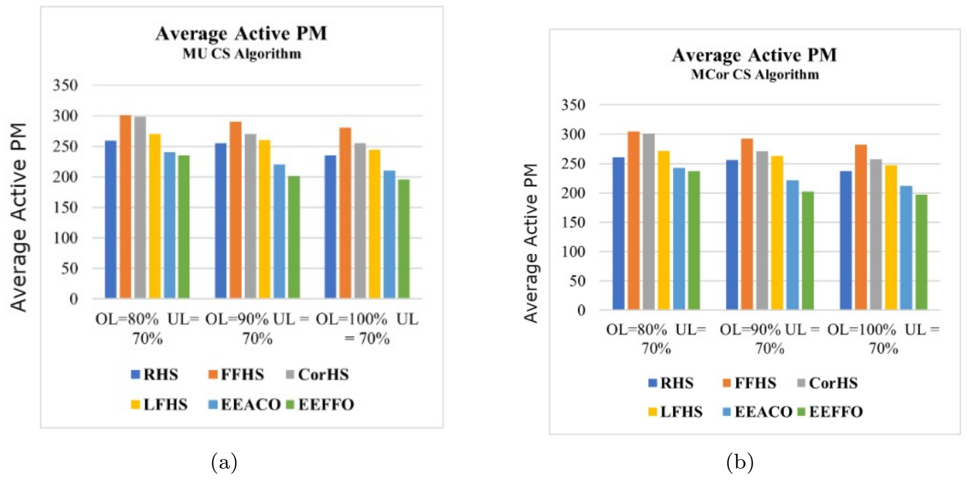
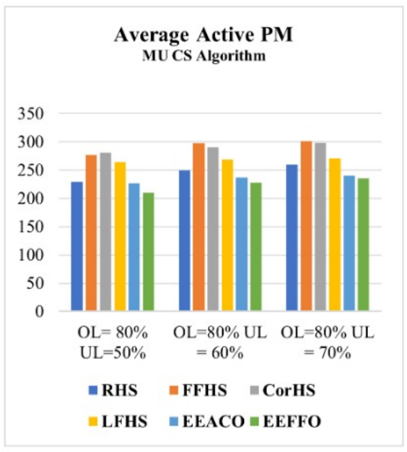


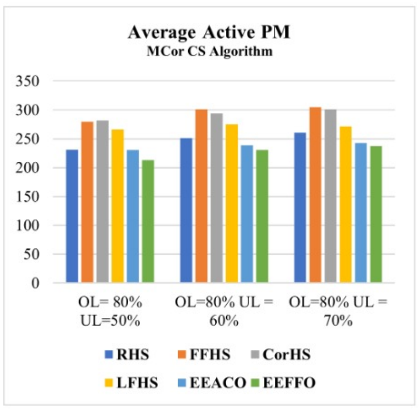
Figure 5.9: Impact of Host selection algorithms on average PMs used for different OL combinations (a) MU and (b) MCor in homogenous environment

5.5.1.5 Impact of host selection algorithm (container consolidation) on SLA:

According to equation 5.3, the SLA metric is defined as a proportion of the difference between the required and allocated amount of CPU for each VM. It is violated, if the VM that hosts the container does not get the necessary amount of CPU. The container placement has been done using algorithm 4.6, Discrete Firefly Container Placement algorithm with Local Search Mechanism (DFFLSM). The technique optimised the initial distribution of containers onto VMs, resulting in fewer SLA breaches, ensuring that each VM received the necessary resources. The impact of different host selection algorithms on average overall SLA violations for different OL-UL combinations and CS algorithms is as shown in figure 5.11 and figure 5.12



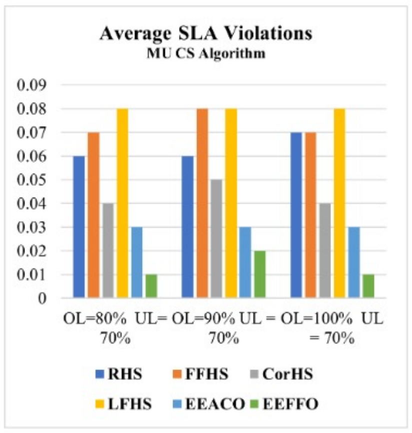
(a)



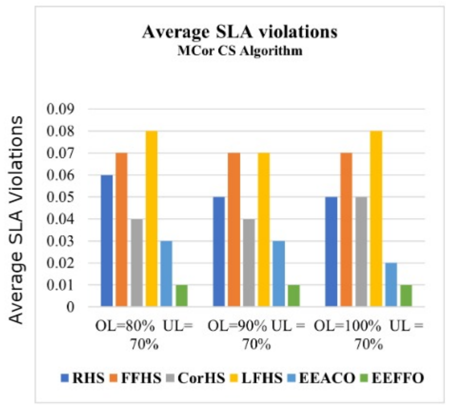
(b)

Figure 5.10: Impact of host selection algorithms on average PMs used for different UL combinations (a) MU and (b) MCor in homogenous environment

respectively.



(a)

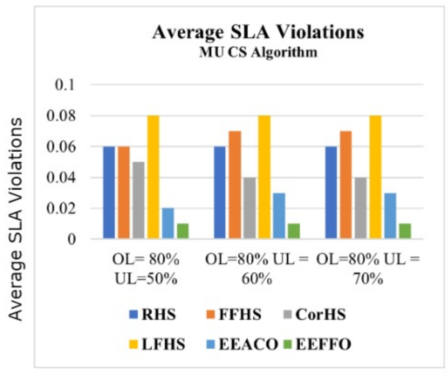


(b)

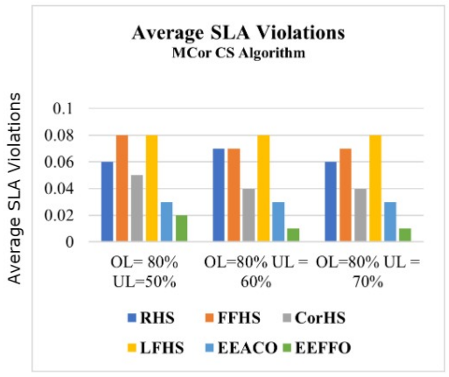
Figure 5.11: Impact of host selection algorithms on average overall SLA violations for different OL combinations (a) MU and (b) MCor in homogenous environment

5.5.1.6 Impact of overbooking of containers on average energy consumption, average active VM, average active PM, average container migrations and average overall SLA violations

Energy consumption and SLA violations are critical factors affecting the efficiency of consolidation algorithms, and overbooking plays a significant role in this regard. Containers have been assigned to VMs based on a specified percentile of each container's application workload. With a higher percentile, fewer containers were deployed on each VM. Host selection techniques generally led to a similar number of migrations because of minimal



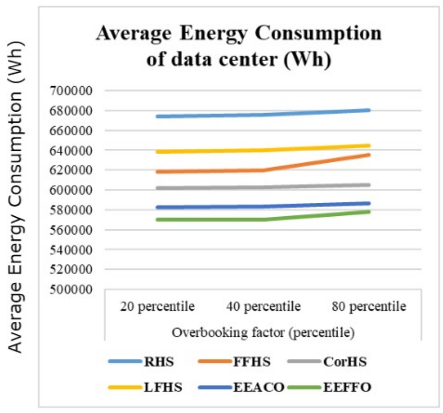
(a)



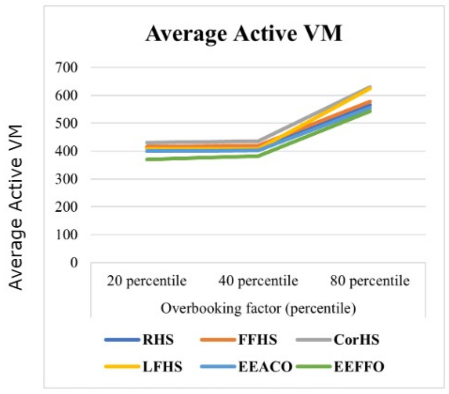
(b)

Figure 5.12: Impact of host selection algorithms on average overall SLA violations for different UL combinations (a) MU and (b) MCor in homogenous environment

workload variance, with migration decisions primarily based on PM load rather than VM load. For these set of experiments, MU CS algorithm was used and UL and OL values were fixed at 70% and 80% respectively. The overbooking factor is varied as 20, 40 and 80. Figure 5.13, figure 5.14 and figure 5.15 show the impact of host selection algorithms on average energy consumption of data center, average active VM, average active PM, average container migrations and average overall SLA violations for different values of overbooking factor (UL=70%, OL=80% and MU CS algorithm)

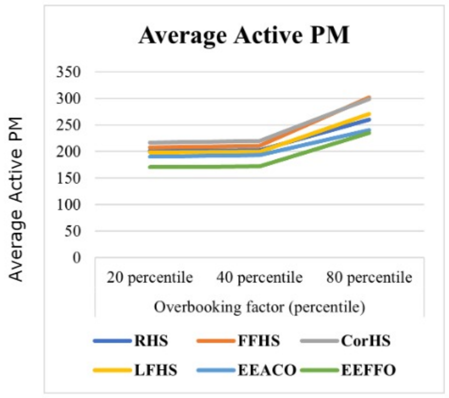


(a)

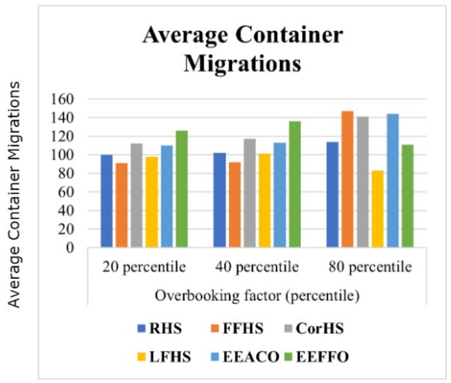


(b)

Figure 5.13: Impact of host selection algorithms on (a) average energy consumption of data center and (b) average active VM for different values of overbooking factor in homogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm)



(a)



(b)

Figure 5.14: Impact of host selection algorithms on (a) average active PM and (b) average container migrations for different values of overbooking factor in homogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm)

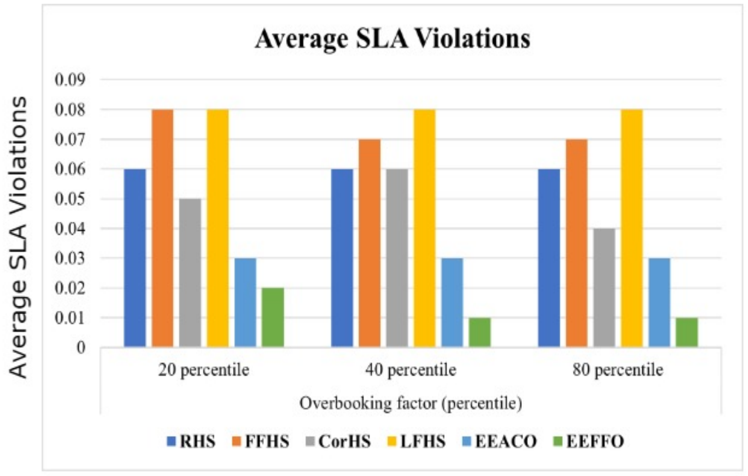


Figure 5.15: Impact of host selection algorithms on average overall SLA violations for different values of overbooking factor in homogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm)

5.5.2 Heterogenous Environment

5.5.2.1 Impact of host selection policies (container consolidation) on average energy consumption:

The usage of energy in DC was calculated by using the various host selection algorithms on different CS algorithms and thresholds. CS algorithms, MU and MCor are defined in section 5.3.3 and host status thresholds are defined in section 5.3.3. The impact of host selection algorithms on average energy consumption for different OL and UL combinations are as shown in figure 5.16 and figure 5.17 respectively. Table 5.14 and table 5.15 shows the comparison of EEFFO and EEACO with other algorithms in terms of average energy-consumption reduction in heterogenous environment.

Table 5.14: Comparison of EEFFO average energy consumption reduction in Heterogenous environment (in percent)

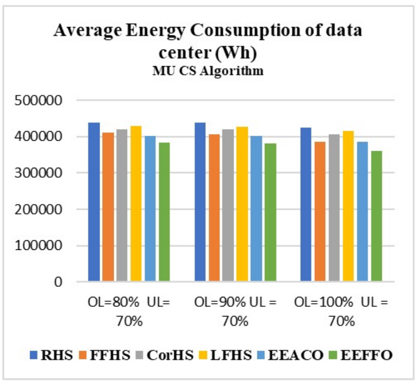
Energy Efficiency in Heterogenous environment (in Percentage)						
		RHS	**FFHS**	**CorHS**	**LFHS**	**EEACO**
EEFFO	**MU (changing OL)**	13.47	6.46	9.65	11.46	5.38
EEFFO	**MU (changing UL)**	13.29	8.32	10.52	11.18	4.31
EEFFO	**MCor (changing OL)**	13.1	6.62	9.76	11.38	5.39
EEFFO	**MCor (changing UL)**	12.89	8.42	10.62	11.1	4.31

Table 5.15: Comparison of EEACO average energy consumption reduction in Heterogenous environment (in percent)

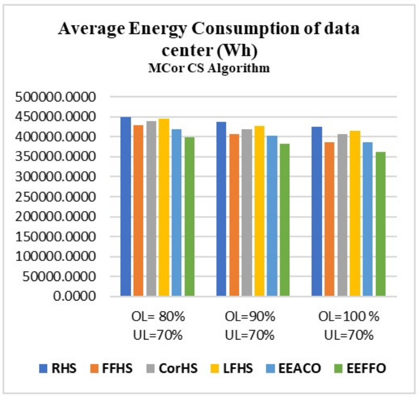
Energy Efficiency in Heterogenous environment (in Percentage)					
		RHS	**FFHS**	**CorHS**	**LFHS**
EEACO	**MU (changing OL)**	8.55	1.14	4.51	6.42
EEACO	**MU (changing UL)**	9.38	4.18	6.48	7.17
EEACO	**MCor (changing OL)**	8.15	1.3	4.62	6.33
EEACO	**MCor (changing UL)**	8.96	4.29	6.59	7.09

5.5.2.2 Impact of host selection policies (container consolidation) on average container migrations:

The average container migration was calculated by using the various host selection algorithms for different CS algorithms and thresholds. Two separate tests were performed based on the OL threshold, UL threshold and different CS algorithms (MU and MCor). For each set, the container migrations were calculated and compared for different combinations of OL and UL thresholds as shown in Figure 5.18 and figure 5.19 respectively.

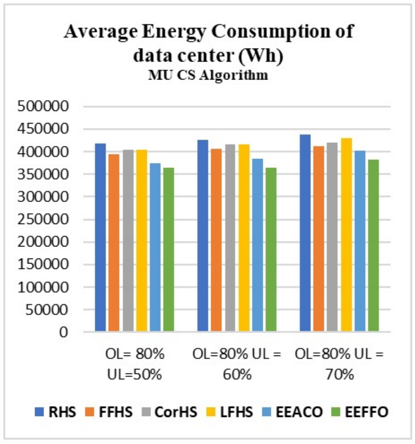


(a)

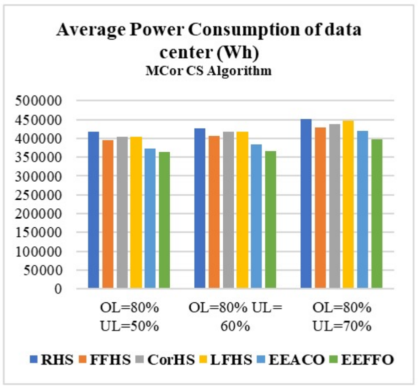


(b)

Figure 5.16: Impact of host selection algorithms on average energy consumption for different OL combinations (a) MU and (b) MCor in heterogenous environment



(a)

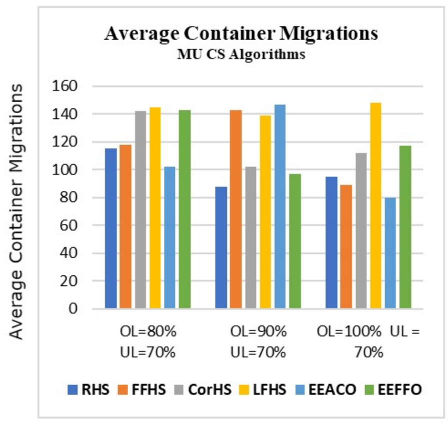


(b)

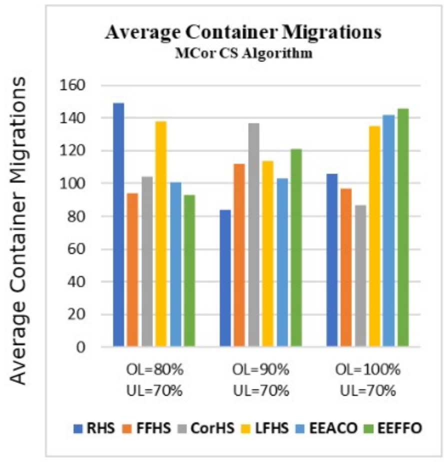
Figure 5.17: Impact of host selection algorithms on average energy consumption for different UL combinations (a) MU and (b) MCor in heterogenous environment

5.5.2.3 Impact of host selection algorithm (container consolidation) on average VM used:

The average VMs used were calculated by using the various host selection algorithms for different CS algorithms and thresholds. Two separate tests were performed based on the OL threshold, UL threshold and different CS algorithms (MU and MCor). For each set, the average VMs used were calculated and compared for different combinations of OL and UL thresholds are as shown in figure 5.20 and figure 5.21 respectively. Table 5.16 and table 5.17 show the percentage reduction of active VM in comparison to EEFFO and EEACO algorithms respectively.

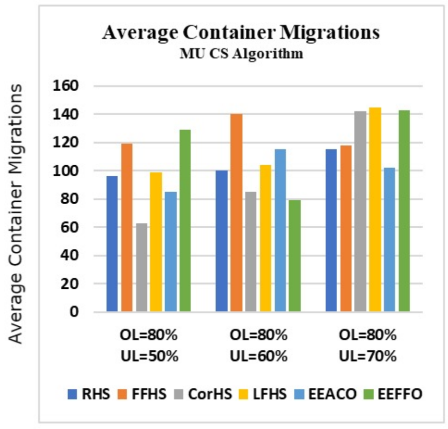


(a)

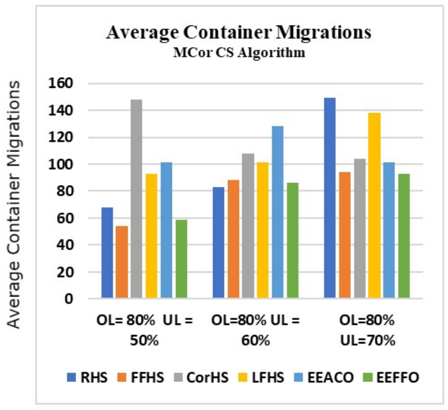


(b)

Figure 5.18: Impact of Host selection algorithms on average container migration for different OL combinations (a) MU and (b) MCor in heterogenous environment



(a)



(b)

Figure 5.19: Impact of Host selection algorithms on average container migration for different UL combinations (a) MU and (b) MCor in heterogenous environment

5.5.2.4 Impact of host selection algorithm (container consolidation) on average PM used:

The average PMs used were calculated by using the various host selection algorithms for different CS algorithms and thresholds. Two separate tests were performed based on the OL threshold, UL threshold and different CS algorithms (MU and MCor). For each set, the average PMs used were calculated and compared for different combinations of OL and UL thresholds as shown in figure 5.22 and figure 5.23 respectively. Table 5.18 and table 5.19 shows the comparison of EEFFO and EEACO with other algorithms in terms of average active PMs reduction in Heterogeneous environment.

Table 5.16: Comparison of EEFFO average active VM reduction in heterogenous environment

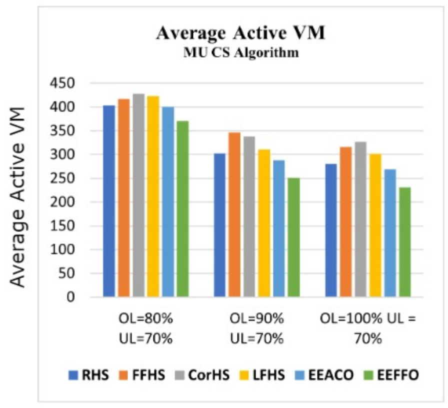
Average Active VM in Heterogenous environment (in Percentage)						
		RHS	FFHS	CorHS	LFHS	EEACO
EEFFO	**MU (changing OL)**	13.7	21.05	22.03	17.74	11.04
EEFFO	**MU (changing UL)**	8.5	11.59	13.67	12.74	7.62
EEFFO	**MCor (changing OL)**	13.68	20.86	22.02	17.36	10.65
EEFFO	**MCor (changing UL)**	8.79	10.32	13.82	12.49	7.49

Table 5.17: Comparison of EEACO average active VM reduction in heterogenous environment

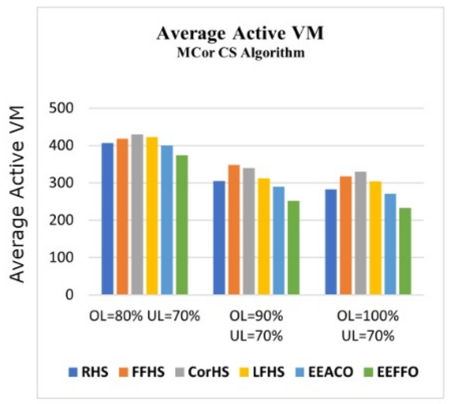
Average Active VM in Heterogenous environment (in Percentage)					
		RHS	FFHS	CorHS	LFHS
EEACO	**MU (changing OL)**	2.98	11.25	12.35	7.53
EEACO	**MU (changing UL)**	0.95	4.3	6.55	5.54
EEACO	**MCor (changing OL)**	3.39	11.43	12.72	7.51
EEACO	**MCor (changing UL)**	1.41	3.05	6.84	5.4

Table 5.18: Comparison of EEFFO average active PMs reduction in Heterogeneous environment (in percent)

Average Active PMs in Heterogeneous environment (in Percentage)						
		RHS	**FFHS**	**CorHS**	**LFHS**	**EEACO**
EEFFO	**MU (changing OL)**	14.25	17.16	9.23	13.16	5.93
EEFFO	**MU (changing UL)**	14.87	16.91	9.1	12.4	6.47
EEFFO	**MCor (changing OL)**	14.67	18.19	10.29	12.87	5.66
EEFFO	**MCor (changing UL)**	13.71	15.74	9.22	9.73	3.61

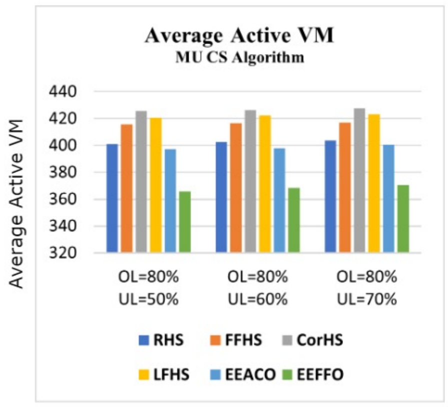


(a)

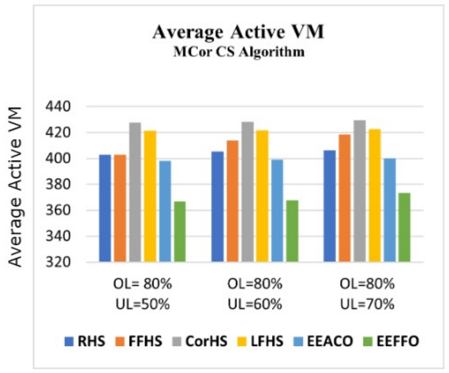


(b)

Figure 5.20: Impact of Host selection algorithms on average VMs used for different OL combinations (a) MU and (b) MCor in heterogenous environment



(a)



(b)

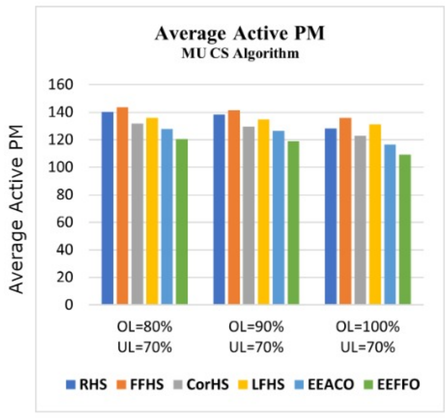
Figure 5.21: Impact of Host selection algorithms on average VMs used for different UL combinations (a) MU and (b) MCor in heterogenous environment

5.5.2.5 Impact of host selection algorithm (container consolidation) on SLA:

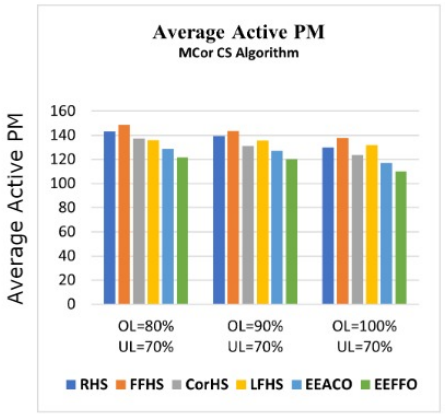
The container placement has been done using algorithm 4.6, Discrete Firefly Container Placement algorithm with local search mechanism. The technique optimised the initial placement of containers onto VMs, resulting in fewer SLA breaches in the experiment, ensuring that each VM receives the necessary resources. Figure 5.24 and figure 5.25 show the impact of host selection algorithms on average overall SLA violation for different OL-UL and CS algorithm combinations respectively.

Table 5.19: Comparison of EEACO average active PMs in Heterogeneous environment (in percent)

Average Active PMs in Heterogeneous environment (in Percentage)					
		RHS	**FFHS**	**CorHS**	**LFHS**
EEACO	**MU (changing OL)**	8.85	11.93	3.51	7.68
EEACO	**MU (changing UL)**	8.97	11.15	2.81	6.34
EEACO	**MCor (changing OL)**	9.55	13.28	4.91	7.64
EEACO	**MCor (changing UL)**	10.47	12.58	5.82	6.34



(a)

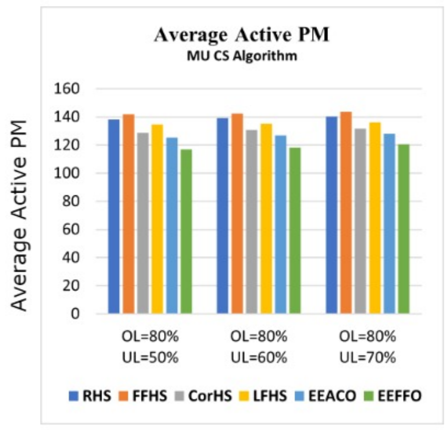


(b)

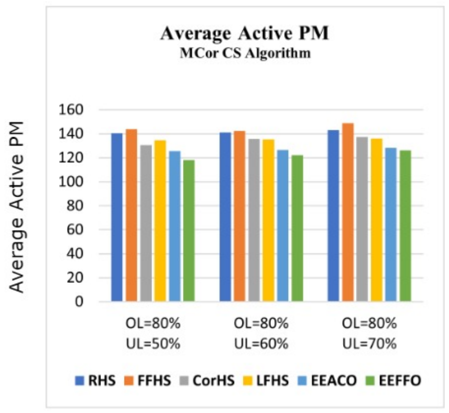
Figure 5.22: Impact of Host selection algorithms on average PMs used for different OL combinations (a) MU and (b) MCor in heterogenous environment

5.5.2.6 Impact of overbooking of containers on average energy consumption, average active VM, average active PM, average container migrations and average overall SLA violations:

Overbooking significantly impacts the consolidation algorithms effectiveness, particularly on the basis of usage of energy and SLA breaches. The allocation of containers to virtual machines is determined based on a predetermined percentile of the application workload associated with each container. The greater the percentile, the fewer containers that can fit on each VM. Because workload variance was minimal and migration decisions are based on PM load rather than VM load, most HSA had the same number of migrations. The MU CS algorithm is employed was this series of studies, and the UL and OL values are set at 70% and 80%, respectively.

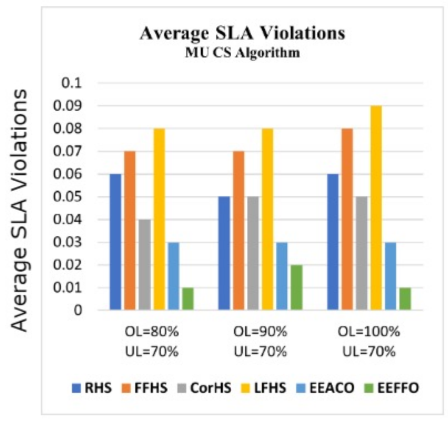


(a)

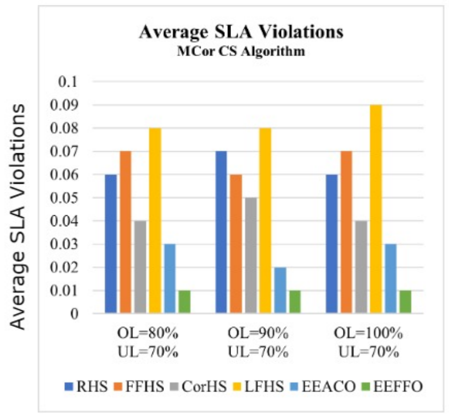


(b)

Figure 5.23: Impact of Host selection algorithms on average PMs used for different UL combinations (a) MU and (b) MCor in heterogenous environment



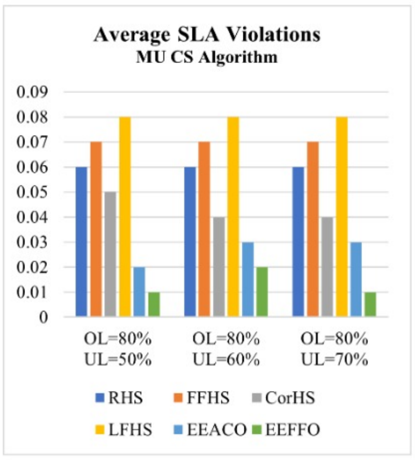
(a)



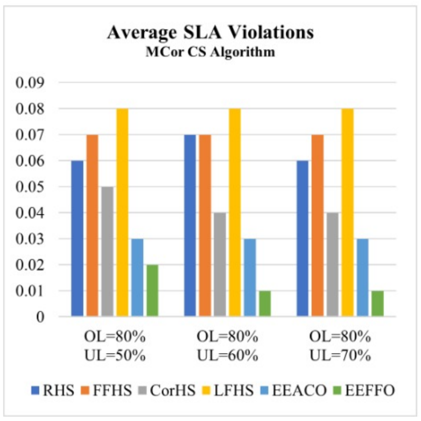
(b)

Figure 5.24: Impact of host selection algorithms on average overall SLA violation for different OL combinations (a) MU and (b) MCor in heterogenous environment

The overbooking factor was set to 20, 40, or 80. Figure 5.26, figure 5.27 and figure 5.28 show the impact of host selection algorithms on average energy consumption of data center, average active VM, average active PM, average container migrations and average overall SLA violations for different values of overbooking factor (UL=70%, OL=80% and MU CS algorithm)

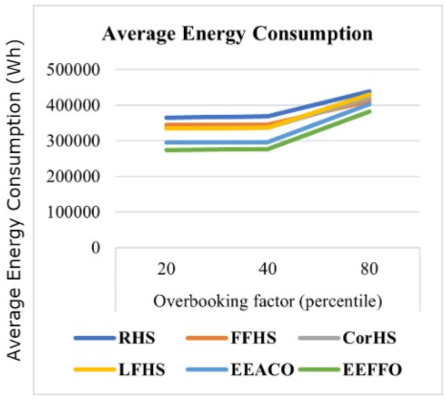


(a)

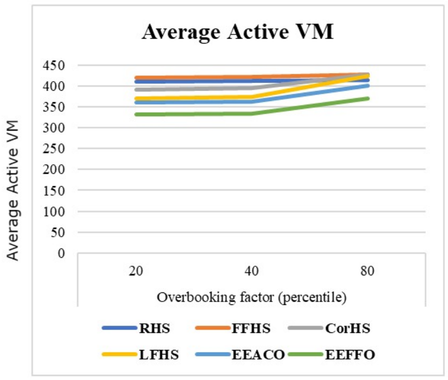


(b)

Figure 5.25: Impact of host selection algorithms on average overall SLA violation for different UL combinations (a) MU and (b) MCor in heterogenous environment

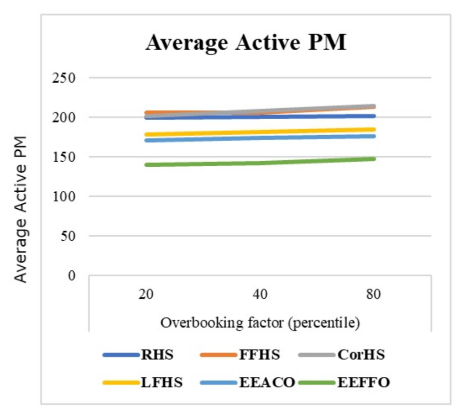


(a)

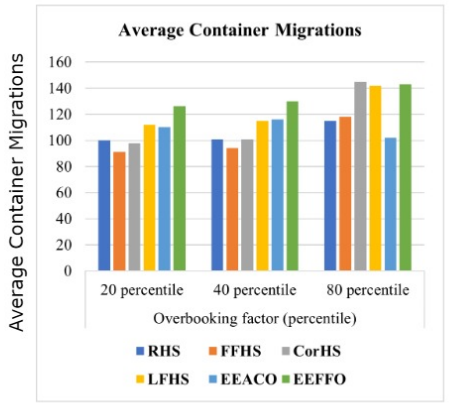


(b)

Figure 5.26: Impact of host selection algorithms on (a) average energy consumption of data center and (b) average active VM for different values of overbooking factor in heterogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm)



(a)



(b)

Figure 5.27: Impact of host selection algorithms on (a) average active PM and (b) average container migrations for different values of overbooking factor in heterogenous environment (UL=70 percent, OL=80 percent and MU CS algorithm)

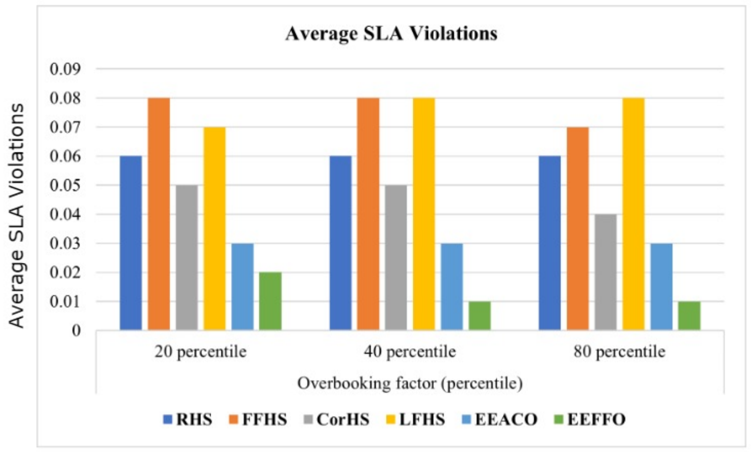


Figure 5.28: Impact of host selection algorithms on average overall SLA violations for different values of overbooking factor in heterogenous environment (UL=70 percent, OL=80 ercent and MU CS algorithm)

5.6 Summary

Container consolidation is a pivotal strategy employed in cloud data centers to enhance resource utilization and optimize operational efficiency. As cloud computing continues to grow in prominence, the need to effectively manage and allocate resources becomes paramount. Containerization technology, which encapsulates applications and their dependencies into isolated units known as containers, has revolutionized the deployment and scaling of applications.Container consolidation leverages this technology to consolidate multiple applications onto fewer physical servers, thereby reducing hardware and energy consumption. By efficiently packing multiple containers onto a single server, data centers can achieve higher resource utilization rates, leading to cost savings and a reduced environmental footprint. This practice is particularly beneficial in dynamic and scalable environments where workloads vary over time. The process involves intelligent load balancing, performance monitoring, and predictive scaling mechanisms to ensure optimal resource distribution and prevent performance bottlenecks. Through advanced orchestration platforms and management tools, container consolidation enables data centers to adapt swiftly to changing workloads while maintaining desired Service Level Agreements (SLAs). Moreover, this approach contributes to the effective utilization of the available hardware, minimizing wasted resources and overall infrastructure costs.

In this research, the EEFFO algorithm demonstrated remarkable energy-saving results in data centers, showcasing an 8.34% reduction in energy consumption in a homogeneous environment and an even more impressive 9.38% reduction in a heterogeneous environment, surpassing the performance of all other existing algorithms. Furthermore, EEFFO achieved a significant decrease in the average power modules (PM) utilized, with a reduction of 16.35% in homogeneous settings and 11.65% in heterogeneous settings when compared to both pre-existing and proposed algorithms. Additionally, EEFFO contributed to an overall improvement in Quality of

Service (QoS) by significantly reducing the number of Service Level Agreement (SLA) violations in comparison to both the pre-existing algorithms and its counterpart EEACO, thereby ensuring enhanced QoS for the data center operations.

Chapter 6

Statistical Analysis

This chapter assesses the effectiveness of proposed and implemented energy efficiency algorithms in data centers by conducting a statistical analysis, employing the Tukey HSD test to detect significant differences. It aims to provide empirical evidence of algorithmic performance and its impact on data center energy efficiency.

6.1 Statistical Analysis

The Tukey HSD (Honestly Significant Difference) test is a statistical technique employed to ascertain the statistical significance of a relationship between two datasets. It helps determine whether a numerical change observed in one variable is likely to be causally related to a numerical change observed in another variable. In essence, the Tukey test serves as a means to rigorously test an experimental hypothesis. Moreover, it is utilized to evaluate if an interaction among three or more variables holds statistical significance, indicating that it goes beyond being a simple sum or product of individual degrees of significance. It is designed to determine which specific group means differ significantly from each other after a significant result is obtained from ANOVA. The Tukey HSD test helps identify pairwise differences between group means in an ANOVA when the null hypothesis of equal means has been rejected. When conducting an ANOVA, if the overall F-test indicates that there are significant differences among the group means, the Tukey HSD test can be used to perform post hoc pairwise comparisons to identify which specific groups differ significantly from each other.

The Tukey HSD test has been applied at container placement and container consolidation level for finding the significant difference between the numerical values of mean energy consumption and mean overall SLA violations for proposed and pre-existing values for both homogenous and heterogeneous environments. Each set of experiment is simulated 50 times to get the optimized results.

6.1.1 Container Placement in Cloud Data Center

In Tukey test, the significance value represents the threshold below which the differences between group means to be statistically significant were considered. In a Tukey test conducted at a 95% confidence level, the significance value is set at 0.05. When the significance value falls below 0.05, it indicates that the distinctions between the means of the groups being compared are statistically significant. In other words, the null hypothesis is rejected, indicating that there is a significant difference in the means of the groups under consideration.

Table 6.1 shows the Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous and heterogeneous data center for different container placement algorithms.

Table 6.2 shows the Tukey multiple comparisons of significance values for energy consumption and SLA violations in homogenous and heterogeneous data center for different container placement algorithms.

DFFLSM showed a significant difference of average energy consumption when compared to FF, FFD, Random and ACO container placement algorithms with a P value of 0.002, 0.005, < 0.001 and 0.015 respectively in homogenous environment as shown in table 6.2.

DFFLSM showed a significant difference of overall SLA violations when compared to FF, FFD, Random, ACO and DFF container placement algorithms with a P value of < 0.001, < 0.001, < 0.001, 0.003 and 0.069 respectively in homogenous environment as shown in table 6.2.

DFFLSM showed a significant difference of average energy consumption

Table 6.1: Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous and heterogeneous data center for different container placement algorithms

		Homogenous Environment		**Heterogenous Environment**	
(I) VAR00001	**(J) VAR00001**	**Mean Diff. (Energy Consumption)**	**Mean Diff. (SLA violations)**	**Mean Diff. (Energy Consumption)**	**Mean Diff. (SLA violations)**
FF	**FFD**	9315.72665	0.01	10363.85335	0.01
	Random	-38013.98830*	-0.01	-51040.7456	-0.01
	ACO	17676.26565	.03500*	187208.79920*	.03000*
	DFF	34636.65595*	.05000*	259464.05580*	.04500*
	DFFLSM	53886.53500*	.06500*	327550.14525*	.06000*
FFD	**FF**	-9315.72665	-0.01	-10363.85335	-0.01
	Random	-47329.71495*	-.02000*	-61404.59895*	-.02000*
	ACO	8360.539	.02500*	176844.94585*	.02000*
	DFF	25320.9293	.04000*	249100.20245*	.03500*
	DFFLSM	44570.80835*	.05500*	317186.29190*	.05000*
Random	**FF**	38013.98830*	0.01	51040.7456	0.01
	FFD	47329.71495*	.02000*	61404.59895*	.02000*
	ACO	55690.25395*	.04500*	238249.54480*	.04000*
	DFF	72650.64425*	.06000*	310504.80140*	.05500*
	DFFLSM	91900.52330*	.07500*	378590.89085*	.07000*
ACO	**FF**	-17676.26565	-.03500*	-187208.79920*	-.03000*
	FFD	-8360.539	-.02500*	-176844.94585*	-.02000*
	Random	-55690.25395*	-.04500*	-238249.54480*	-.04000*
	DFF	16960.3903	0.015	72255.25660*	.01500*
	DFFLSM	36210.26935*	.03000*	140341.34605*	.03000*
DFF	**FF**	-34636.65595*	-.05000*	-259464.05580*	-.04500*
	FFD	-25320.9293	-.04000*	-249100.20245*	-.03500*
	Random	-72650.64425*	-.06000*	-310504.80140*	-.05500*
	ACO	-16960.3903	-0.015	-72255.25660*	-.01500*
	DFFLSM	19249.87905	0.015	68086.08945*	.01500*
DFFLSM	**FF**	-53886.53500*	-.06500*	-327550.14525*	-.06000*
	FFD	-44570.80835*	-.05500*	-317186.29190*	-.05000*
	Random	-91900.52330*	-.07500*	-378590.89085*	-.07000*
	ACO	-36210.26935*	-.03000*	-140341.34605*	-.03000*
	DFF	-19249.87905	-0.015	-68086.08945*	-.01500*
* The mean difference is significant at the 0.05 level.					

Table 6.2: Tukey multiple comparisons of significance values for energy consumption and SLA violations in homogenous and heterogeneous data center for different container placement algorithms

Multiple Comparisons of Container Placement Algorithms					
Dependent Variable: VAR00002					
Tukey HSD					
		Homogenous Environment		**Heterogeneous Environment**	
(I) VAR00001	**(J) VAR00001**	**Sig.(Energy Consumption)**	**Sig.(SLA violations)**	**Sig.(Energy Consumption)**	**Sig.(SLA violations)**
FF	**FFD**	0.758	0.271	0.971	0.087
	Random	0.012	0.271	0.076	0.087
	ACO	0.244	0.001	< .001	< .001
	DFF	0.018	< .001	< .001	< .001
	DFFLSM	0.002	< .001	< .001	< .001
FFD	**FF**	0.758	0.271	0.971	0.087
	Random	0.004	0.02	0.035	0.003
	ACO	0.822	0.007	< .001	0.003
	DFF	0.072	< .001	< .001	< .001
	DFFLSM	0.005	< .001	< .001	< .001
Random	**FF**	0.012	0.271	0.076	0.087
	FFD	0.004	0.02	0.035	0.003
	ACO	0.002	< .001	< .001	< .001
	DFF	< .001	< .001	< .001	< .001
	DFFLSM	< .001	< .001	< .001	< .001
ACO	**FF**	0.244	0.001	< .001	< .001
	FFD	0.822	0.007	< .001	0.003
	Random	0.002	< .001	< .001	< .001
	DFF	0.273	0.069	0.017	0.015
	DFFLSM	0.015	0.003	< .001	< .001
DFF	**FF**	0.018	< .001	< .001	< .001
	FFD	0.072	< .001	< .001	< .001
	Random	< .001	< .001	< .001	< .001
	ACO	0.273	0.069	0.017	0.015
	DFFLSM	0.19	0.069	0.022	0.015
DFFLSM	**FF**	0.002	< .001	< .001	< .001
	FFD	0.005	< .001	< .001	< .001
	Random	< .001	< .001	< .001	< .001
	ACO	0.015	0.003	< .001	< .001
	DFF	0.19	0.069	0.022	0.015

when compared to FF, FFD, Random and ACO container placement algorithms with a P value of < 0.001, $< .001$, $< .001$ and $< .001$ respectively in heterogeneous environment as shown in table 6.2.

DFFLSM showed a significant difference of overall SLA violations when compared to FF, FFD, Random, ACO and DFF container placement algorithms with a P value of $< .001$, $< .001$, $< .001$, $< .001$ and 0.015 respectively in heterogeneous environment as shown in table 6.2.

6.1.2 Container Consolidation in Cloud Data Center

Table 6.3 shows the Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous data center for different container consolidation algorithms with MU as container selection algorithm.

Table 6.4 shows the Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous data center for different container consolidation algorithms with MCor as container selection algorithm.

Table 6.5 shows the Tukey multiple comparisons of significance values for energy consumption and SLA violations in homogenous data center for different container consolidation algorithms with MU as container selection algorithm.

Table 6.6 shows the Tukey multiple comparisons of significance values for energy consumption and SLA violations in homogenous data center for different container consolidation algorithms with MCor as container selection algorithm.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, 0.001, 0.235, 0.01$ $and \quad 0.862$ respectively in homogenous environment for MU Container Selection and OL threshold changing as shown in table 6.5.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS,

Table 6.3: Tukey multiple comparisons of mean difference for energy consumption and SLA violations in homogenous data center for different container consolidation algorithms with MU as container selection algorithm

Tukey HSD		**Energy Consumption (MU, OL Changing)**	**Energy Consumption (MU, UL Changing)**	**SLA violations (MU, OL Changing)**	**SLA violations (MU, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**
RHS	**FFHS**	14478.92	-8485.07	-0.02	-0.01
	CorHS	54449.2	54543.48	0.02	0.02
	LFHS	33872.94	37766.31	-0.02	-0.02
	EEACO	64353.1	75420.68	0.03	0.04
	EEFFO	71839.18	108813.91	0.05	0.05
FFHS	**RHS**	-14478.92	8485.07	0.02	0.01
	CorHS	39970.29	63028.56	0.03	0.02
	LFHS	19394.03	46251.38	-0.01	-0.02
	EEACO	49874.19	83905.75	0.05	0.04
	EEFFO	57360.26	117298.98	0.06	0.06
CorHS	**RHS**	-54449.2	-54543.48	-0.02	-0.02
	FFHS	-39970.29	-63028.56	-0.03	-0.02
	LFHS	-20576.26	-16777.18	-0.04	-0.04
	EEACO	9903.9	20877.2	0.02	0.02
	EEFFO	17389.98	54270.42	0.03	0.04
LFHS	**RHS**	-33872.94	-37766.31	0.02	0.02
	FFHS	-19394.03	-46251.38	0.01	0.02
	CorHS	20576.26	16777.18	0.04	0.04
	EEACO	30480.16	37654.37	0.05	0.06
	EEFFO	37966.24	71047.6	0.07	0.07
EEACO	**RHS**	-64353.1	-75420.68	-0.03	-0.04
	FFHS	-49874.19	-83905.75	-0.05	-0.04
	CorHS	-9903.9	-20877.2	-0.02	-0.02
	LFHS	-30480.16	-37654.37	-0.05	-0.06
	EEFFO	7486.08	33393.23	0.02	0.02
EEFFO	**RHS**	-71839.18	-108813.91	-0.05	-0.05
	FFHS	-57360.26	-117298.98	-0.06	-0.06
	CorHS	-17389.98	-54270.42	-0.03	-0.04
	LFHS	-37966.24	-71047.6	-0.07	-0.07
	EEACO	-7486.08	-33393.23	-0.02	-0.02

Table 6.4: Tukey multiple comparisons of mean difference for energy consumption and SLA violations in data center for different container consolidation algorithms with MCor as container selection algorithm

Multiple comparisons for Container Consolidation Algorithms in Homogenous Environment					
Tukey HSD		**Energy Consumption (MCor,OL Changing)**	**Energy Consumption on (MCsor, UL Changing)**	**SLA violations (MCor, OL Changing)**	**SLA violations (MCor, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**
RHS	**FFHS**	9861.22	-8336.45	-0.02	-0.01
	CorHS	52336.86	54681.77	0.02	0.02
	LFHS	29341.34	37989.76	-0.02	-0.02
	EEACO	62031.01	75563.52	0.03	0.04
	EEFFO	71349.91	108192.68	0.05	0.05
FFHS	**RHS**	-9861.22	8336.45	0.02	0.01
	CorHS	42475.64	63018.21	0.03	0.03
	LFHS	19480.12	46326.2	-0.01	-0.01
	EEACO	52169.8	83899.97	0.04	0.05
	EEFFO	61488.69	116529.12	0.06	0.06
CorHS	**RHS**	-52336.86	-54681.77	-0.02	-0.02
	FFHS	-42475.64	-63018.21	-0.03	-0.03
	LFHS	-22995.52	-16692.01	-0.04	-0.04
	EEACO	9694.15	20881.75	0.01	0.02
	EEFFO	19013.05	53510.91	0.03	0.03
LFHS	**RHS**	-29341.34	-37989.76	0.02	0.02
	FFHS	-19480.12	-46326.2	0.01	0.01
	CorHS	22995.52	16692.01	0.04	0.04
	EEACO	32689.67	37573.76	0.05	0.05
	EEFFO	42008.57	70202.92	0.07	0.07
EEACO	**RHS**	-62031.01	-75563.52	-0.03	-0.04
	FFHS	-52169.8	-83899.97	-0.04	-0.05
	CorHS	-9694.15	-20881.75	-0.01	-0.02
	LFHS	-32689.67	-37573.76	-0.05	-0.05
	EEFFO	9318.9	32629.16	0.02	0.02
EEFFO	**RHS**	-71349.91	-108192.68	-0.05	-0.05
	FFHS	-61488.69	-116529.12	-0.06	-0.06
	CorHS	-19013.05	-53510.91	-0.03	-0.03
	LFHS	-42008.57	-70202.92	-0.07	-0.07
	EEACO	-9318.9	-32629.16	-0.02	-0.02

Table 6.5: Tukey multiple comparisons of significance values for energy consumption and SLA violations in data center for different container consolidation algorithms with MU as container selection algorithm

Tukey HSD		**Energy Consumption (MU, OL Changing)**	**Energy Consumption (MU, UL Changing)**	**SLA violations (MU, OL Changing)**	**SLA violations (MU, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Sig.**	**Sig.**	**Sig.**	**Sig.**
RHS	**FFHS**	0.374	0.965	0.147	0.902
	CorHS	0.002	0.02	0.147	0.147
	LFHS	0.018	0.096	0.049	0.049
	EEACO	$< .001$	0.004	0.007	0.003
	EEFFO	$< .001$	$< .001$	$< .001$	$< .001$
FFHS	**RHS**	0.374	0.965	0.147	0.902
	CorHS	0.008	0.01	0.007	0.049
	LFHS	0.168	0.042	0.902	0.147
	EEACO	0.002	0.002	$< .001$	0.002
	EEFFO	0.001	$< .001$	$< .001$	$< .001$
CorHS	**RHS**	0.002	0.02	0.147	0.147
	FFHS	0.008	0.01	0.007	0.049
	LFHS	0.138	0.676	0.003	0.003
	EEACO	0.693	0.493	0.147	0.049
	EEFFO	0.235	0.02	0.007	0.003
LFHS	**RHS**	0.018	0.096	0.049	0.049
	FFHS	0.168	0.042	0.902	0.147
	CorHS	0.138	0.676	0.003	0.003
	EEACO	0.029	0.097	$< .001$	$< .001$
	EEFFO	0.01	0.005	$< .001$	$< .001$
EEACO	**RHS**	$< .001$	0.004	0.007	0.003
	FFHS	0.002	0.002	$< .001$	0.002
	CorHS	0.693	0.493	0.147	0.049
	LFHS	0.029	0.097	$< .001$	$< .001$
	EEFFO	0.862	0.148	0.147	0.147
EEFFO	**RHS**	$< .001$	$< .001$	$< .001$	$< .001$
	FFHS	0.001	$< .001$	$< .001$	$< .001$
	CorHS	0.235	0.02	0.007	0.003
	LFHS	0.01	0.005	$< .001$	$< .001$
	EEACO	0.862	0.148	0.147	0.147

Table 6.6: Tukey multiple comparisons of significance values for energy consumption and SLA violations in data center for different container consolidation algorithms with MCor as container selection algorithm

Tukey HSD		**Energy Consumption (MCor, OL Changing)**	**Energy Consumption (MCor, UL Changing)**	**SLA violations (MCor, OL Changing)**	**SLA violations (MCor, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Sig.**	**Sig.**	**Sig.**	**Sig.**
RHS	**FFHS**	0.918	0.967	0.069	0.559
	CorHS	0.017	0.019	0.069	0.087
	LFHS	0.178	0.092	0.02	0.23
	EEACO	0.007	0.004	0.007	0.007
	EEFFO	0.004	$< .001$	$< .001$	0.001
FFHS	**RHS**	0.918	0.967	0.069	0.559
	CorHS	0.044	0.01	0.003	0.015
	LFHS	0.489	0.041	0.812	0.942
	EEACO	0.017	0.002	$< .001$	0.002
	EEFFO	0.008	$< .001$	$< .001$	$< .001$
CorHS	**RHS**	0.017	0.019	0.069	0.087
	FFHS	0.044	0.01	0.003	0.015
	LFHS	0.347	0.677	0.001	0.007
	EEACO	0.923	0.489	0.271	0.23
	EEFFO	0.51	0.021	0.003	0.015
LFHS	**RHS**	0.178	0.092	0.02	0.23
	FFHS	0.489	0.041	0.812	0.942
	CorHS	0.347	0.677	0.001	0.007
	EEACO	0.124	0.096	$< .001$	0.001
	EEFFO	0.046	0.006	$< .001$	$< .001$
EEACO	**RHS**	0.007	0.004	0.007	0.007
	FFHS	0.017	0.002	$< .001$	0.002
	CorHS	0.923	0.489	0.271	0.23
	LFHS	0.124	0.096	$< .001$	0.001
	EEFFO	0.934	0.158	0.02	0.23
EEFFO	**RHS**	0.004	$< .001$	$< .001$	0.001
	FFHS	0.008	$< .001$	$< .001$	$< .001$
	CorHS	0.51	0.021	0.003	0.015
	LFHS	0.046	0.006	$< .001$	$< .001$
	EEACO	0.934	0.158	0.02	0.23

CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.02, 0.005$ *and* 0.148 respectively in homogenous environment for MU Container Selection and UL threshold changing as shown in table 6.5.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.007, < .001$ *and* 0.147 respectively in homogenous environment for MU Container Selection and OL threshold changing as shown in table 6.5.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.003, < .001$ *and* 0.147 in homogenous environment for MU Container Selection and UL threshold changing as shown in table 6.5.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of 0.004, 0.008, 0.51, 0.046 and 0.934 respectively in homogenous environment for MCor Container Selection and OL threshold changing as shown in table 6.6.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.021, 0.006$ *and* 0.158 respectively in homogenous environment for MCor Container Selection and UL threshold changing as shown in table 6.6.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.003, < .001$ *and* 0.02 respectively in homogenous environment for MCor Container Selection and OL threshold changing as shown in table 6.6.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $0.001, < .001, 0.015, < .001$ *and* 0.23 in homogenous environment for MU Container Selection and UL threshold changing as shown in table 6.6.

Table 6.7 shows the Tukey multiple comparisons of mean difference for energy consumption and SLA violations in heterogenous data center for different container consolidation algorithms with MU as container selection algorithm.

Table 6.8 shows the Tukey multiple comparisons of mean difference for energy consumption and SLA violations in heterogenous data center for different container consolidation algorithms with MCor as container selection algorithm.

Table 6.9 shows the Tukey multiple comparisons of significance values for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithms with MU as container selection algorithm.

Table 6.10 shows the Tukey multiple comparisons of significance values for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithms with MCor as container selection algorithm.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, < .001, < .001 \quad and \quad < .001$ respectively in heterogeneous environment for MU Container Selection and OL threshold changing as shown in table 6.9.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of 0.001, 0.016, 0.004, 0.004 and 0.382 respectively in heterogeneous environment for MU Container Selection and UL threshold changing as shown in table 6.9.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $0.002, < .001, 0.007, < .001$ $and 0.147$ respectively in heterogeneous environment for MU Container Selection and OL threshold changing as shown in table 6.9.

EEFFO showed a significant difference of average overall SLA viola-

Table 6.7: Tukey multiple comparisons of mean difference for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MU as container selection algorithm

Tukey HSD		**Energy Consumption (MU, OL Changing)**	**Energy Consumption (MU, UL Changing)**	**SLA violations (MU, OL Changing)**	**SLA violations (MU, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**
RHS	**FFHS**	28902.64	21403.43	-0.02	-0.01
	CorHS	17795.55	10889.48	0.01	0.02
	LFHS	9952.66	10859.34	-0.03	-0.02
	EEACO	35717.6	42358.95	0.03	0.04
	EEFFO	55845.27	57258.99	0.04	0.05
FFHS	**RHS**	-28902.64	-21403.43	0.02	0.01
	CorHS	-11107.09	-10513.95	0.03	0.03
	LFHS	-18949.99	-10544.09	-0.01	-0.01
	EEACO	6814.96	20955.52	0.04	0.05
	EEFFO	26942.63	35855.56	0.06	0.06
CorHS	**RHS**	-17795.55	-10889.48	-0.01	-0.02
	FFHS	11107.09	10513.95	-0.03	-0.03
	LFHS	-7842.9	-30.15	-0.04	-0.04
	EEACO	17922.05	31469.47	0.02	0.02
	EEFFO	38049.72	46369.51	0.03	0.03
LFHS	**RHS**	-9952.66	-10859.34	0.03	0.02
	FFHS	18949.99	10544.09	0.01	0.01
	CorHS	7842.9	30.15	0.04	0.04
	EEACO	25764.94	31499.62	0.05	0.06
	EEFFO	45892.61	46399.65	0.07	0.07
EEACO	**RHS**	-35717.6	-42358.95	-0.03	-0.04
	FFHS	-6814.96	-20955.52	-0.04	-0.05
	CorHS	-17922.05	-31469.47	-0.02	-0.02
	LFHS	-25764.94	-31499.62	-0.05	-0.06
	EEFFO	20127.67	14900.04	0.02	0.01
EEFFO	**RHS**	-55845.27	-57258.99	-0.04	-0.05
	FFHS	-26942.63	-35855.56	-0.06	-0.06
	CorHS	-38049.72	-46369.51	-0.03	-0.03
	LFHS	-45892.61	-46399.65	-0.07	-0.07
	EEACO	-20127.67	-14900.04	-0.02	-0.01

Table 6.8: Tukey multiple comparisons of mean difference for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MCor as container selection algorithm

Tukey HSD		**Energy Consumption (MCor,OL Changing)**	**Energy Consumption (MCor, UL Changing)**	**SLA violations (MCor, OL Changing)**	**SLA violations (MCor, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**	**Mean Diff.**
RHS	**FFHS**	25700.72	21409.68	0	-0.01
	CorHS	14545.28	10790.26	0.02	0.02
	LFHS	7873.92	10854.74	-0.02	-0.02
	EEACO	33635.87	42347.34	0.04	0.04
	EEFFO	54196.78	57187.16	0.06	0.05
FFHS	**RHS**	-25700.72	-21409.68	0	0.01
	CorHS	-11155.43	-10619.42	0.02	0.03
	LFHS	-17826.8	-10554.94	-0.02	-0.01
	EEACO	7935.16	20937.66	0.04	0.04
	EEFFO	28496.06	35777.49	0.06	0.06
CorHS	**RHS**	-14545.28	-10790.26	-0.02	-0.02
	FFHS	11155.43	10619.42	-0.02	-0.03
	LFHS	-6671.36	64.48	-0.04	-0.04
	EEACO	19090.59	31557.08	0.02	0.02
	EEFFO	39651.49	46396.9	0.04	0.03
LFHS	**RHS**	-7873.92	-10854.74	0.02	0.02
	FFHS	17826.8	10554.94	0.02	0.01
	CorHS	6671.36	-64.48	0.04	0.04
	EEACO	25761.95	31492.6	0.06	0.05
	EEFFO	46322.85	46332.42	0.07	0.07
EEACO	**RHS**	-33635.87	-42347.34	-0.04	-0.04
	FFHS	-7935.16	-20937.66	-0.04	-0.04
	CorHS	-19090.59	-31557.08	-0.02	-0.02
	LFHS	-25761.95	-31492.6	-0.06	-0.05
	EEFFO	20560.9	14839.82	0.02	0.02
EEFFO	**RHS**	-54196.78	-57187.16	-0.06	-0.05
	FFHS	-28496.06	-35777.49	-0.06	-0.06
	CorHS	-39651.49	-46396.9	-0.04	-0.03
	LFHS	-46322.85	-46332.42	-0.07	-0.07
	EEACO	-20560.9	-14839.82	-0.02	-0.02

Table 6.9: Tukey multiple comparisons of significance values for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MU as container selection algorithm

Tukey HSD		**Energy Consumption (MU, OL Changing)**	**Energy Consumption (MU, UL Changing)**	**SLA violations (MU, OL Changing)**	**SLA violations (MU, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Sig.**	**Sig.**	**Sig.**	**Sig.**
RHS	**FFHS**	$< .001$	0.138	0.147	0.434
	CorHS	$< .001$	0.65	0.434	0.147
	LFHS	0.013	0.652	0.018	0.049
	EEACO	$< .001$	0.007	0.018	0.003
	EEFFO	$< .001$	0.001	0.002	$< .001$
FFHS	**RHS**	$< .001$	0.138	0.147	0.434
	CorHS	0.008	0.677	0.018	0.018
	LFHS	$< .001$	0.675	0.434	0.434
	EEACO	0.071	0.148	0.002	$< .001$
	EEFFO	$< .001$	0.016	$< .001$	$< .001$
CorHS	**RHS**	$< .001$	0.65	0.434	0.147
	FFHS	0.008	0.677	0.018	0.018
	LFHS	0.039	1	0.003	0.003
	EEACO	$< .001$	0.03	0.147	0.049
	EEFFO	$< .001$	0.004	0.007	0.007
LFHS	**RHS**	0.013	0.652	0.018	0.049
	FFHS	$< .001$	0.675	0.434	0.434
	CorHS	0.039	1	0.003	0.003
	EEACO	$< .001$	0.029	$< .001$	$< .001$
	EEFFO	$< .001$	0.004	$< .001$	$< .001$
EEACO	**RHS**	$< .001$	0.007	0.018	0.003
	FFHS	0.071	0.148	0.002	$< .001$
	CorHS	$< .001$	0.03	0.147	0.049
	LFHS	$< .001$	0.029	$< .001$	$< .001$
	EEFFO	$< .001$	0.382	0.147	0.434
EEFFO	**RHS**	$< .001$	0.001	0.002	$< .001$
	FFHS	$< .001$	0.016	$< .001$	$< .001$
	CorHS	$< .001$	0.004	0.007	0.007
	LFHS	$< .001$	0.004	$< .001$	$< .001$
	EEACO	$< .001$	0.382	0.147	0.434

Table 6.10: Tukey multiple comparisons of significance values for energy consumption and SLA violations in heterogeneous data center for different container consolidation algorithm with MCor as container selection algorithm

Tukey HSD		**Energy Consumption (MCor,OL Changing)**	**Energy Consumption (MCor, UL Changing)**	**SLA violations (MCor, OL Changing)**	**SLA violations (MCor, UL Changing)**
(I) VAR00001	**(J) VAR00001**	**Sig.**	**Sig.**	**Sig.**	**Sig.**
RHS	**FFHS**	0.428	0.138	1	0.902
	CorHS	0.849	0.657	0.087	0.049
	LFHS	0.985	0.652	0.23	0.147
	EEACO	0.22	0.007	0.003	0.003
	EEFFO	0.038	0.001	$< .001$	$< .001$
FFHS	**RHS**	0.428	0.138	1	0.902
	CorHS	0.94	0.67	0.087	0.018
	LFHS	0.729	0.674	0.23	0.434
	EEACO	0.985	0.149	0.003	0.002
	EEFFO	0.341	0.016	$< .001$	$< .001$
CorHS	**RHS**	0.849	0.657	0.087	0.049
	FFHS	0.94	0.67	0.087	0.018
	LFHS	0.993	1	0.007	0.003
	EEACO	0.679	0.029	0.087	0.147
	EEFFO	0.13	0.004	0.007	0.007
LFHS	**RHS**	0.985	0.652	0.23	0.147
	FFHS	0.729	0.674	0.23	0.434
	CorHS	0.993	1	0.007	0.003
	EEACO	0.426	0.029	$< .001$	$< .001$
	EEFFO	0.073	0.004	$< .001$	$< .001$
EEACO	**RHS**	0.22	0.007	0.003	0.003
	FFHS	0.985	0.149	0.003	0.002
	CorHS	0.679	0.029	0.087	0.147
	LFHS	0.426	0.029	$< .001$	$< .001$
	EEFFO	0.62	0.386	0.23	0.147
EEFFO	**RHS**	0.038	0.001	$< .001$	$< .001$
	FFHS	0.341	0.016	$< .001$	$< .001$
	CorHS	0.13	0.004	0.007	0.007
	LFHS	0.073	0.004	$< .001$	$< .001$
	EEACO	0.62	0.386	0.23	0.147

tions for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.007, < .001 \quad and \quad 0.434$ in heterogeneous environment for MU Container Selection and UL threshold changing as shown in table 6.9.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of 0.038, 0.341, 0.13, 0.073 and 0.62 respectively in heterogeneous environment for MCor Container Selection and OL threshold changing as shown in table 6.10.

EEFFO showed a significant difference of average energy consumption for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of 0.001, 0.016, 0.004, 0.004 and 0.386 respectively in heterogeneous environment for MCor Container Selection and UL threshold changing as shown in table 6.10.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.007, < .001 \quad and \quad 0.23$ respectively in heterogeneous environment for MCor Container Selection and OL threshold changing as shown in table 6.10.

EEFFO showed a significant difference of average overall SLA violations for container consolidation algorithms when compared to RHS, FFHS, CorHS, LFHS and EEACO with a P value of $< .001, < .001, 0.007, < .001 \quad and \quad 0.147$ in heterogeneous environment for MU Container Selection and UL threshold changing as shown in table 6.10.

6.2 Summary

In this chapter, a comprehensive statistical analysis of the proposed algorithms was conducted using the Tukey HSD test. The results demonstrated a significant difference between the performance of these algorithms and the pre-existing ones. This analysis provides empirical evidence of the effectiveness and superiority of the proposed algorithms in enhancing various aspects of the data center's operations and energy efficiency.

Chapter 7

Discussion

This chapter discusses the outcomes of a series of experiments conducted to evaluate workload characterization and categorization, container placement, and container consolidation algorithms. It sheds light on the effectiveness of the proposed algorithms and their impact on optimizing data center operations. The results provide valuable insights into the practical application and performance of these algorithms in real-world scenarios.

7.1 Discussion

Cloud computing guarantees high throughput, adaptability, and cost-effectiveness to address evolving processing necessities. As the quantity of data continues to grow at an appalling rate, an increasing number of firms are turning to data centers to make effective choices and achieve a competitive edge. The cloud-computing model is used for a multitude of applications. These applications range in their features and place varied demands on the resources of Physical Machines (PMs). Database-based applications (which do intense read and write activities on disk, for example) have different needs than science-based computing programs (which require substantial processing power from the CPU). To effectively configure cloud resources, network managers must be able to characterize and predict workload on VMs. Clustering the tasks into groups or clusters is feasible based on the different demands of dissimilar tasks of cloud applications. The clustering process can identify characterizations that can

improve the efficiency of historical workload traces over a wide range of critical performance parameters, such as increasing the utilization of PMs hosted in cloud data centers.
The utilization and prominence of cloud computing as among the most well-known internet-based inventions for supplying computational power and infrastructural facilities to IT organizations for executing/hosting cloud workloads is expanding every day and is anticipated to expand even further. Consumers upload heterogeneous cloud workloads to the cloud through internet services, banking applications, online payment processing assistance, portable computing assistance, and graphics-based services, with varying QoS parameters in the form of SLA. Workload of different types are clustered using two different clustering techniques. After clustering is completed, the performance is examined to determine which clustering works best.

- The majority of tasks (93.38 percent) when clustered by K Means algorithm belonged to low resource utilization category (CPU, memory, storage, space, and network bandwidth).
- GMM showed maximum 16.61 % of the tasks consumed CPU (very Low), Disk (very Low), Network (very Low) and Memory (very high) resources.
- The results demonstrated that K means outperforms GMM in both Calinski Harabasz Index and the Davies-Bouldin Index.
- After clustering, classification is carried through utilizing several classification techniques. The decision tree showed the maximum accuracy of 99.18%.

Compared to the existing study included in section 2.1, this work provided an in depth explanation of different clustering and classification strategies used for cloud data center workloads.

Containers is an OS-level virtualization approach that may be used on VMs or on physical systems (PMs). Its major role is to offer an independent system to run applications. Unlike a VM, which needs to run the whole OS, a container can use the same operating system kernel as other containers. Consequently, a container is deemed lightweight, using few resources and requiring little set-up time. Container as a Service (CaaS) is rapidly being used in the cloud to give end customers with more service alternatives, such as Fargate by Amazon and Kubernetes by Google. Under the commonly used CaaS architecture, where containers can only be deployed on VMs, finding a practical placement with an optimal energy usage remains a difficulty. The container placement basically refers to allocating containers to VMs and VMs to appropriate computing nodes in order to fulfil an intended objective under certain resource restrictions. CaaS is a relatively new cloud service concept that is built on container virtualization, has evolved to provide containers as services. CaaS addresses the issue of applications created in a single PaaS context and whose implementation is limited to the needs of that PaaS ecosystem. CaaS liberates the application by removing constraints and removing it from the PaaS construction environment. Container-based programs are consequently transportable and may run in any environment.

Simulating container placement and consolidation in a homogeneous server environment, where servers have uniform capabilities, is essential for evaluating basic placement strategies and their impact on resource utilization and performance. However, real-world cloud data centers are characterized by heterogeneous servers with diverse configurations. Introducing simulation in a heterogeneous environment adds complexity, reflecting actual data centers and providing a realistic representation of challenges in dynamic, mixed-capacity settings. Homogeneous simulations may oversimplify performance expectations, ignoring variability in server capabilities. Heterogeneous simulations explore performance variations among servers, requiring adaptive algorithms for load balancing and optimal resource utilization. While basic placement and consolidation strategies work in homogeneous settings, their efficacy in optimizing resource utilization becomes more challenging in heterogeneous environments. Simulating in such settings assesses the adaptability of placement algorithms in dynamically allocating workloads to servers with varying capacities. Insights gained from homogeneous simulations may not fully capture challenges in heterogeneous environments, especially regarding scalability and varied scaling behaviors across different servers. Simulating in the latter helps evaluate the impact of scaling strategies on performance and resource usage. Exploring energy-efficient strategies in homogeneous environments is a starting point, but simulating in heterogeneous environments allows for a more comprehensive examination, considering diverse servers with distinct power profiles and efficiency characteristics, necessitating tailored strategies for optimal energy utilization in container placement and consolidation.

The contributions of the container placement work done in order to reduce energy consumption are as follows:

- An optimized meta-heuristic container placement algorithm, Discrete Firefly with Local Search mechanism (DFFLSM) and without local search mechanism (DFF) have been proposed and implemented for container placement.
- DFF and DFFLSM algorithms are compared with the FF, FFD, Random and ACO in terms of average energy consumption, average active VM average active PM and average overall SLA violations.
- The investigation has been done for both homogenous and heterogenous environments taking into account the workload traces of PlanetLab.
- The proposed algorithms were compared with pre-existing ones by varying the thresholds of RAM of VMs.
- DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average energy efficiency by 9.32% and 40.85% in homogenous and heterogenous environment respectively.
- DFFLSM outperformed FF, FFD, Random, ACO and DFF in terms of average active VM by 36.07% and 33.48% in homogenous and heterogenous environment respectively.
- DFFLSM outperformed FF, FFD Random, ACO and DFF in terms

of average active PM by 18.30% and 21.89% in homogenous and heterogenous environment respectively.

- DFF outperformed FF, FFD, Random and ACO container placement algorithms in terms of energy efficiency. It reduced energy consumption of DC by and 6.62% and 34.06% in homogenous and heterogenous environment respectively as compared to all other algorithms.
- DFF outperformed FF, FFD, Random and ACO in terms of average active VM by 28.93% and 29.47% in homogenous and heterogenous environment respectively.
- DFF also reduced the average PM used by 13.55% and 17.88% in homogenous and heterogenous environment respectively as compared to the pre-existing algorithms.
- DFFLSM and DFF reduced the overall average SLA violations as compared to pre-existing container placement algorithms i.e. FF, FFD Random, ACO for both homogenous and heterogenous environments.
- The experimentation results for checking the impact of different container placement algorithms for different values of overbooking factor showed that there is a significant increase of average active VMs and PMs corresponding to a higher percentile of overbooking factor leading to high-energy consumption. However, the results of DFF and DFFLSM show that these algorithms perform better in comparison to pre-existing algorithms.

Server consolidation is advocated as an important energy-aware method in cloud DCs in this direction. It is a key feature that is made possible by virtualization technology, which includes live movement of VMs and containers (S. S. Patra, 1 C.E.). To conserve energy, many virtual machines (VMs) and containers are densely packed into the smallest possible number of physical machines (PMs), allowing idle hosts to be powered off or switched to sleep mode. However, consolidation faces a challenge in finding the right balance between efficiency and QoS. Hence, an effective consolidation process must consider both SLAs and energy efficiency.

There are two forms of consolidation: static consolidation and dynamic consolidation. In static consolidation, which involves the allocation of VMs to PMs without migration, no VM movement occurs. Dynamic consolidation, on the other hand, dynamically relocates VMs between PMs according on their current resource utilisation. This process is repeated until the most energy-efficient design with the fewest active PMs is obtained.

To offer QoS while minimising energy waste and better resource allocation, cloud systems require resource management solutions. In order to innovate and compare resource-managing techniques, review platforms that ease experiment design while also making them reproducible and accurate are required. Simulators are great tools for creating such an evaluation environment on the cloud (Zhao et al., 2012). They are especially useful in the early phases of research to discover and discard inefficient algorithm, or when access to large scale distributed infrastructure is prohibitively expensive. Testing and assessing resource management rules in a production environment during the initial verification step is both dangerous and costly. In this regard, a variety of simulation tools for evaluating algorithms created exclusively for CC settings are being developed. Regardless of the fact that containers will be one of the key application deployment strategies in the cloud, most simulations see virtualized cloud DCs as their base (Beloglazov and Buyya, 2010).

The contributions of the container consolidation work done in order to reduce energy consumption while maintaining QoS are as follows:

- Two energy aware metaheuristic algorithms, Energy Efficient Ant Colony Optimization (EEACO) and Energy Efficient Firefly Optimization (EEFFO) have been proposed and implemented for selection of host in order to consolidate containers.
- The proposed host selection algorithms EEACO and EEFFO have been compared with pre-existing algorithms in Cloudsim 4.0 in terms of energy efficiency, average container migrations, average active VM , average active PM and average overall SLA violations.
- The comparison of the aforementioned algorithms is done considering two possibilities of Container Selection algorithms inbuilt in Cloudsim

4.0 i.e. Maximum Usage (MU) and Maximum Correlation (MCor).

- The comparison of the suggested algorithms is done considering different Overload and Underload host machine thresholds. The experiments have been conducted to investigate the different possibilities.
- The investigation has been done for both homogenous and heterogenous environments taking into account PlanetLab workload.
- The results showed that EEFFO outperformed all the pre-existing algorithms and proposed EEACO in terms of average energy consumed. In the case of homogenous environment, EEFFO reduced energy consumption up to 8.34% as compared to the pre-existing algorithms like RHS, FFHS, CorHS LFHS and proposed EEACO algorithm. Also, in case of heterogenous environment, the EEFFO algorithm optimized the energy efficiency data center by reducing the energy up to 9.38% as compared to the pre-existing algorithms like RHS, FFHS, CorHS , LFHS and proposed EEACO algorithm.
- The results showed that EEFFO outperformed all the pre-existing algorithms and proposed EEACO in terms of active VM used. In the case of homogenous environment, EEFFO reduced the average active VM by 9.19% as compared to the pre-existing algorithms like RHS, FFHS, CorHS LFHS and proposed EEACO algorithm. In case of heterogenous environment, the EEFFO algorithm optimized the average VM used by 13.85% as compared to the pre-existing algorithms like RHS, FFHS, CorHS , LFHS and proposed EEACO algorithm.
- The results showed that EEFFO outperformed all the pre-existing algorithms and proposed EEACO in terms of active PM used. In the case of homogenous environment, the EEFFO reduced the average active PM by 16.35% as compared to the pre-existing algorithms like RHS, FFHS, CorHS, LFHS and proposed EEACO algorithm. In case of heterogenous environment, the EEFFO algorithm optimized the average PM used by 11.65% as compared to the pre-existing algorithms like RHS, FFHS, CorHS, LFHS and proposed EEACO algorithm.

- The proposed algorithms showed less number of SLA violations in comparison to pre-existing algorithms. However, EEFFO outperformed all other algorithms and showed least number of SLA breaches.

- The experimentation results of checking the impact of different container consolidation / host selection algorithms for different values of overbooking factor showed that there is a significant increase of average active VMs and PMs corresponding to a higher percentile of overbooking factor leading to high-energy consumption. However, the results of EEFFO and EEACO are better in comparison to pre-existing algorithms.

Chapter 8

Conclusion and Future Work

This chapter delves into the future prospects of container placement and consolidation in cloud data centers. Emerging trends and technologies are explored, shedding light on the potential developments that could impact the optimization of data center resources and energy efficiency. It offers a glimpse into the evolving landscape of this field.

8.1 Conclusion and Future Work

The utilization and prominence of cloud computing as among the most well-known internet-based inventions for supplying computational power and infrastructural facilities to IT organizations for executing/hosting cloud workloads is expanding every day and is anticipated to expand even further. Consumers upload heterogeneous cloud workloads to the cloud through internet services, banking applications, online payment processing assistance, portable computing assistance, and graphics-based services, with varying QoS parameters in the form of SLA. Virtualization is a key innovation that supports cloud computing and provides considerable benefits in reducing server size. This method reduces cloud computing operating expenses while making greater use of system services. VMs with distinct equipment and application requirements can then be placed into a single or several hosts. The Container Placement (CP) strategy of assigning containers to VMs and VMs to PMs must appropriately organize the containers and VMs to make the most use of the resources available. The primary goal of CP is to lower the cloud data center's energy usage. This, in return, reduces maintenance expenses and benefits the environment. Moreover, limiting the number of VMs that run container is a sub-goal for lowering operational expenditure, which includes VM creation and transfer. Containerization is a technique that configures physical components in a container and transfers programs and their requirements across many OSs. Energy efficiency of DC is a significant issue that has been a concern for researchers, cloud providers and environmentalists as it increases CO_2 emissions and hence affects global warming. Having a DC consuming less energy not only increases Return on Investment (ROI) for the providers but also impacts the environment in a positive way. It improves resource utilization levels by maximizing energy efficiency through optimal container and server consolidation. Container consolidation has been proposed as a means of adjusting the allotment of containers and cloud servers in order to enable load balancing. Even though containers have less overhead than VMs, the cost should not be overlooked when performing the migration. Frequent container migration results in significant expenditures when transferring containers across servers. As a result, while delivering CaaS, there is a compromise between expense of migration and load balancing and hence energy efficiency.

The importance of classifying and describing workloads in cloud data centers is examined in research. Workload of different types are clustered using two different clustering techniques. Workload distribution is accomplished by combining distinct workload pairings in both clustering modes. After clustering is completed, the performance is examined to determine which clustering works best. The majority of tasks according to K Means algorithm (93.38%) have low resource utilization (CPU, memory, storage space, and network bandwidth). Short administrative tasks and application enquiries make up these virtual machines. GMM shows maximum 16.61 percent of the tasks consume CPU (very Low), Disk (very Low), Network (very Low) and Memory (very high) resources. However, the results demonstrate that K means beats in both Calinski Harabasz Index and the Davies-Bouldin Index. After clustering, classification is carried through utilizing several classification techniques. The decision tree shows the maximum accuracy of 99.18 %. The suggested DFFLSM container placement algorithm surpasses current algorithms such as FF, FFD, ACO, and DFF and optimizes data center energy usage. When compared to all other methods, it reduces average DC energy usage by 9.32% and 40.85% in homogeneous and heterogeneous environments, respectively. It also decreases the average active PM in homogeneous and heterogeneous environments by 18.30% and 21.89%, respectively, when compared to pre-existing and suggested methods. In terms of energy efficiency, DFF surpasses FF, FFD, Random, and ACO container placement algorithms. When compared to all other methods, it reduces DC energy usage by 6.62% and 34.06% in homogeneous and heterogeneous environments, respectively. It also decreases the average active PM in homogeneous and heterogeneous environments by 13.55% and 17.88%, respectively, when compared to pre-existing and suggested methods. In container consolidation, the proposed algorithm EEFFO reduces energy consumption of data center by 8.34% and 9.38% in homogenous and heterogenous environment respectively as compared to all other algorithms. It also reduces the average PM used by 16.35% and 11.65% in homogenous and heterogenous environment respectively as compared to the pre-existing algorithms and proposed algorithms. The number of overall SLA violations for EEFFO and EEACO is less in comparison to pre-existing algorithms, thus improving QoS also.

8.2 Limitations and Future Work

There is a new sort of container known as an application container, which is dedicated to a single process that executes the resident programme. Application containers are considered a groundbreaking development in the era of cloud computing. They are valued for their lightweight nature, ease of construction and management, and the potential to substantially reduce startup times. These containers serve as the cornerstone of modern Platform as a Service (PaaS) offerings. In this research there a few issues in this field, for reducing energy consumption in cloud data centre by determining effective VM sizes for container placement and container consolidation techniques.

In future research, potential areas of investigation could include:

- Investigating the performance of intrusion phenomenon in the container placement procedure, since this type of placement not only tries to minimize energy usage but also the interference issues. Container interference problems encompass difficulties and clashes that may emerge when numerous containers utilize the same foundational infrastructure or resources within a containerized environment, like Docker or Kubernetes. These challenges frequently result from resource competition, dependency conflicts, or configuration issues, and they have the potential to adversely affect the effectiveness, consistency, and dependability of applications operating within containers.
- The consolidation algorithms can be enhanced to take into account container communications. Accessing the networking at the container level, on the other hand, raises the complexity of the techniques since the amount of containers in a cloud data centre exceeds the number of VMs. Containers are often more numerous than virtual computers. The identification of underload and overload PMs necessitates the use of migration of containers as well as virtual machines.

Chapter 9

References

9.1 References

Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M., and Steinder, M. (2015). Docker Containers across Multiple Clouds and Data Centers. Proceedings - 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing, UCC 2015, 368–371.

Abts, D., Marty, M. R., Wells, P. M., Klausler, P. and Liu, H.: Energy proportional datacenter networks. Proceedings - International Symposium on Computer Architecture. 338–347 (2010) doi:10.1145/1815961.1816004

Affetti, L., Bresciani, G., and Guinea, S. (2015). aDock: a cloud infrastructure experimentation environment based on open stack and docker. In roceedings of the 8th IEEE International Conference on Cloud Computing (pp. 203–210).

Ahn, J. and Park, H. S.: Measurement and modeling the power consumption of router interface. International Conference on Advanced Communication Technology. 860–863 (2014) doi:10.1109/ICACT.2014.6779082

Akindele, T., Tan, B., Mei, Y., and Ma, H. (2022). Hybrid Grouping Genetic Algorithm for Large-Scale Two-Level Resource Allocation of Containers in the Cloud. Lecture Notes in Computer Science (Including Subseries

Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 13151 LNAI, 519–530.

alibaba/clusterdata: cluster data collected from production clusters in Alibaba for cluster management research. (n.d.). Retrieved March 17, 2022.

Ali-Eldin, A., Rezaie, A., Mehta, A., Razroev, S., Luna, S. S. de, Seleznjev, O., Tordsson, J., and Elmroth, E. (2014). How will your workload look like in 6 years? Analyzing Wikimedia's workload. Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014, 349–354. https://doi.org/10.1109/IC2E.2014.50.

Al-Moalmi, A., Luo, J., Salah, A., Li, K., and Yin, L. (2021). A whale optimization system for energy-efficient container placement in data centers. Expert Systems with Applications, 164, 113719.

Ammar, A. M., Luo, J., Tang, Z., and Wajdy, O. (2019). Intra-Balance Virtual Machine Placement for Effective Reduction in Energy Consumption and SLA Violation. IEEE Access, 7, 72387–72402. https://doi.org/10.1109/ACCESS.2019.2920010.

Andrae, A. (2019). Comparison of Several Simplistic High-Level Approaches for Estimating the Global Energy and Electricity Use of ICT Networks and Data Centers. International Journal of Green Technology, 5(1), 50–63. https://doi.org/10.30634/2414-2077.2019.05.06.

Andrae, A., and Edler, T. (2015). On Global Electricity Usage of Communication Technology: Trends to 2030. Challenges, 6(1), 117–157.

Anselmi, J., Amaldi, E., and Cremonesi, P. (2008, September). Service consolidation with end-to-end response time constraints. In 2008 34th Euromicro Conference Software Engineering and Advanced Applications (pp. 345-352). IEEE.

Attia, K. M., El-Hosseini, M. A. and Ali, H. A.: Dynamic power management techniques in multi-core architectures: A survey study. Ain Shams Engineering Journal. 8, 445–456 (2017).

Avram, M. G. (2014). Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective. Procedia Technology, 12, 529–534.

Balaji, K., Sai Kiran, P., and Sunil Kumar, M. (2023). Power aware virtual machine placement in IaaS cloud using discrete firefly algorithm. Applied Nanoscience (Switzerland), 13(3), 2003–2011.
https://doi.org/10.1007/S13204-021-02337-X/METRICS.

Barrett, D., and Kipper, G. (2010). How Virtualization Happens. Virtualization and Forensics, 3–24. https://doi.org/10.1016/B978-1-59749-557-8.00001-1.

Bazzi, H., Harb, A., Aziza, H. and Moreau, M.: Design of Hybrid CMOS Non-Volatile SRAM Cells in 130nm RRAM Technology. Proceedings of the International Conference on Microelectronics. 228–231 (2018)

Beloglazov, A., and Buyya, R. (2010). Energy efficient resource management in virtualized cloud data centers. CCGrid 2010 - 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing, 826–831.

Beloglazov, A., and Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. Concurrency and Computation: Practice and Experience, 24(13), 1397–1420.

Birke, R., Chen, L. Y., and Smirni, E. (2014). Multi-resource characterization and their (in)dependencies in production datacenters. IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World.

Blankstein, A. et al.: Hyperbolic Caching: Flexible Caching for Web Applications. Proceedings of the 2017 USENIX Annual Technical Conference (2017)

Bose, R., Roy, S., Mondal, H., Chowdhury, D. R., and Chakraborty, S. (2021). Energy-efficient approach to lower the carbon emissions of data centers. Computing, 103(8), 1703–1721. https://doi.org/10.1007/S00607-020-00889-4/METRICS.

Bouaouda, A., Afdel, K., and Abounacer, R. (2022). Forecasting the Energy Consumption of Cloud Data Centers Based on Container Placement with Ant Colony Optimization and Bin Packing. 5th Conference on Cloud and Internet of Things, CIoT 2022, 150–157.

Bouaouda, A., Afdel, K., and Abounacer, R. (2023). Meta-heuristic and Heuristic Algorithms for Forecasting Workload Placement and Energy Consumption in Cloud Data Centers. Advances in Science, Technology and Engineering Systems Journal, 8(1), 1–11. https://doi.org/10.25046/AJ080101.

Boukadi, K., Grati, R., Rekik, M., and Abdallah, H. ben. (2017). From VM to container: A linear program for outsourcing a business process to cloud containers. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10573 LNCS, 488–504.

Caglar, F., Shekhar, S., and Gokhale, A. (2013). A performance Interference-aware virtual machine placement strategy for supporting soft realtime applications in the cloud. Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA, Tech. Rep. ISIS-13-105.

Calzarossa, M., and Serazzi, G. (1993). Workload characterization: A survey. Proceedings of the IEEE, 81(8), 1136-1150.

Campbell, S., and Jeronimo, M. (2006). An introduction to virtualization. Published in "Applied Virtualization", Intel, 1-15.

Carla, C., MassariLuisa, and TesseraDaniele. (2016). Workload Characterization. ACM Computing Surveys (CSUR), 2, 759–770.

Carroll, M., van der Merwe, A., and Kotzé, P. (2011). Secure cloud computing: Benefits, risks and controls. 2011 Information Security for South Africa - Proceedings of the ISSA 2011 Conference.

Carvalho, C. A. B. de, Andrade, R. M. de C., Castro, M. F. de, Coutinho, E. F., and Agoulmine, N. (2017). State of the art and challenges of security SLA for cloud computing. Computers and Electrical Engineering, 59, 141–152. https://doi.org/10.1016/J.COMPELECENG.2016.12.030

Characterizing Application Workloads on CPU Utilization for Utility Computing. (n.d.). Retrieved March 17, 2022,
from https://www.hpl.hp.com/techreports/2004/HPL-2004-157.

Charles Reiss, Alexey Tumanov, Alexey Tumanov, Gregory R Ganger, and Randy H Katz. (2012). Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis.

Chen, C., He, K., and Guan, Q. (2018). Minimum migration time selection algorithm for container consolidation. 2018 IEEE International Conference on Information and Automation, ICIA 2018, 1664–1668.

Chen, F., Zhou, X., and Shi, C. (2019, April). The container deployment strategy based on stable matching. In 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA) (pp. 215-221). IEEE.

Chen, L., Dai, W. and Qiu, M.: A Greedy Approach for Caching in Distributed Data Stores. Proceedings - 2nd IEEE International Conference on Smart Cloud. 244–249 (2017) doi:10.1109/SmartCloud.2017.46

Chen, M., Zhang, H., Su, Y. Y., Wang, X., Jiang, G., and Yoshihira, K. (2011, May). Effective VM sizing in virtualized data centers. In 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops (pp. 594-601). IEEE.

Chen, Y.-L., Chang, M.-F., Yu, C.-W., Chen, X.-Z. and Liang, W.-Y.: Learning-Directed Dynamic Voltage and Frequency Scaling Scheme with Adjustable Performance for Single-Core and Multi-Core Embedded and Mobile Systems. Sensors. 18, 3068 (2018).

Cheng, Y., Chai, Z., and Anwar, A. (2018). Characterizing co-located datacenter workloads: An alibaba case study. Proceedings of the 9th Asia-Pacific Workshop on Systems, APSys 2018.

Chhikara, P., Tekchandani, R., Kumar, N., and Obaidat, M. S. (2021). An Efficient Container Management Scheme for Resource-Constrained Intelligent IoT Devices. IEEE Internet of Things Journal, 8(16), 12597–12609.

Corradi, A., Fanelli, M., and Foschini, L. (2014). VM consolidation: A real case based on OpenStack Cloud. Future Generation Computer Systems, 32(1), 118–127. https://doi.org/10.1016/J.FUTURE.2012.05.012.

Cuadrado-Cordero, I., Orgerie, A. C., and Menaud, J. M. (2017). Comparative experimental analysis of the quality-of-service and energy-efficiency of VMs and containers' consolidation for cloud applications. 2017 25th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2017. https://doi.org/10.23919/SOFTCOM.2017.8115516.

David, H., Fallin, C., Gorbatov, E., Hanebutte, U. R., and Mutlu, O. (2011, June). Memory power management via dynamic voltage/frequency scaling. In Proceedings of the 8th ACM international conference on Autonomic computing (pp. 31-40).

Dayarathna, M., Wen, Y., and Fan, R. (2016). Data center energy consumption modeling: A survey. IEEE Communications Surveys and Tutorials, 18(1), 732–794. https://doi.org/10.1109/COMST.2015.2481183.

de Souza, E. A. G., Nagano, M. S., and Rolim, G. A. (2022). Dynamic Programming algorithms and their applications in machine scheduling: A review. Expert Systems with Applications, 190, 116180. https://doi.org/10.1016/J.ESWA.2021.116180.

Delimitrou, C., and Kozyrakis, C. (2011). Cross-examination of datacenter workload modeling techniques. Proceedings - International Conference on Distributed Computing Systems, 72–79.

Deng, Q., Meisner, D., Bhattacharjee, A., Wenisch, T. F., and Bianchini, R. (2012, December). Coscale: Coordinating cpu and memory system dvfs in server systems. In 2012 45th annual IEEE/ACM international symposium on microarchitecture (pp. 143-154). IEEE.
Deng, Q., Meisner, D., Ramos, L., Wenisch, T. F., and Bianchini, R. (2011). Memscale: active low-power modes for main memory. ACM SIGPLAN Notices, 46(3), 225-238.

Dhiman, G., Pusukuri, K. K., and Rosing, T. (2008). Analysis of dynamic voltage scaling for system level energy management. USENIX HotPower, 8.

Ding, W., Gu, C., Luo, F., Chang, Y., Rugwiro, U., Li, X., and Wen, G. (2018). DFA-VMP: An efficient and secure virtual machine placement strategy under cloud environment. Peer-to-Peer Networking and Applications, 11(2), 318–333. https://doi.org/10.1007/S12083-016-0502-

Z/METRICS.

Dong, Z., Zhuang, W., and Rojas-Cessa, R. (2014). Energy-aware scheduling schemes for cloud data centers on Google trace data. 2014 IEEE Online Conference on Green Communications, OnlineGreenComm 2014.

Dorronsoro, B. et al.: A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems. Sustainable Computing: Informatics and Systems. 4, 252–261 (2014).

Farzai, S., Shirvani, M. H., and Rabbani, M. (2020). Multi-objective communication-aware optimization for virtual machine placement in cloud datacenters. Sustainable Computing: Informatics and Systems, 28, 100374.

Feitelson, D. G. (2015). Workload modeling for computer systems performance evaluation. Cambridge University Press.

Ferrari, D. (1972). Workload charaterization and selection in computer performance measurement. Computer, 5(4), 18-24.

Forestiero, A., Mastroianni, C., Meo, M., Papuzzo, G., and Sheikhalishahi, M. (2014). Hierarchical approach for green workload management in distributed data centers. In Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part I 20 (pp. 323-334). Springer International Publishing.

Fujita, S. et al.: Novel memory hierarchy with e-STT-MRAM for near-future applications. 2017 International Symposium on VLSI Technology, Systems and Application. 3–4 (2017) doi:10.1109/VLSI-TSA.2017.7942444

Gao, Y., Zhang, H., Zhu, Y., Tang, B. and Ma, H.: A Load-Aware Data Migration Scheme for Distributed Surveillance Video Processing with Hybrid Storage Architecture. Proceedings - 2017 IEEE 19th Intl Conference on High Performance Computing and Communications, 2017 IEEE 15th Intl Conference on Smart City and 2017 IEEE 3rd Intl Conference on Data Science and Systems. 563–570 (2018)

Ghribi, C. (2014). Energy efficient resource allocation in cloud computing environments. https://theses.hal.science/tel-01149701.

Gill, S. S., and Buyya, R. (2018). A Taxonomy and Future Directions for Sustainable Cloud Computing. ACM Computing Surveys (CSUR), 51(5).

Gmach, D., Rolia, J., Cherkasova, L., and Kemper, A. (2009). Resource pool management: Reactive versus proactive or let's be friends. Computer Networks, 53(17), 2905-2922.

Guan, X., Wan, X., Choi, B. Y., Song, S., and Zhu, J. (2017). Application Oriented Dynamic Resource Allocation for Data Centers Using Docker Containers. IEEE Communications Letters, 21(3), 504–507.

Guzek, M., Kliazovich, D. and Bouvry, P.: HEROS: Energy-Efficient Load Balancing for Heterogeneous Data Centers. Proceedings - 2015 IEEE 8th International Conference on Cloud Computing. 742–749 (2015).

Hanafy, W. A., Mohamed, A. E., and Salem, S. A. (2018). Novel selection policies for container-based cloud deployment models. ICENCO 2017 - 13th International Computer Engineering Conference: Boundless Smart Societies, 2018-January, 237–242. https://doi.org/10.1109/ICENCO.2017.8289794.

He, J. and Callenes-Sloan, J.: Optimizing energy in a DRAM based hybrid cache. Proceedings - International Symposium on Quality Electronic Design. 37–42 (2018)

Heller, B. et al.: Elastictree: Saving energy in data center networks. Proceedings of NSDI 2010: 7th USENIX Symposium on Networked Systems Design and Implementation 249–264 (2010)

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., ... and Stoica, I. (2011). Mesos: A platform for Fine-Grained resource sharing in the data center. In 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11).

How Much Energy Do Data Centers Really Use? - Energy Innovation: Policy and Technology. (n.d.). Retrieved February 4, 2023

Huang, D., and Wu, H. (2018). Virtualization. Mobile Cloud Computing, 31–64. https://doi.org/10.1016/B978-0-12-809641-3.00003-X.

Huang, S., and Feng, W. (2009). Energy-efficient cluster computing via accurateworkload characterization. 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009, 68–75.

Hussein, M. K., Mousa, M. H., and Alqarni, M. A. (2019). A placement architecture for a container as a service (CaaS) in a cloud environment. Journal of Cloud Computing, 8(1), 1–15. https://doi.org/10.1186/S13677-019-0131-1/FIGURES/17.

Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., and Heming, J. (2023). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. Information Sciences, 622, 178-210.

Ismaeel, S., Al-Khazraji, A., and Miri, A. (2019). An efficient workload clustering framework for large-scale data centers. 2019 8th International Conference on Modeling Simulation and Applied Optimization, ICMSAO 2019.

Jain, N., and Choudhary, S. (2016). Overview of virtualization in cloud computing. 2016 Symposium on Colossal Data Analysis and Networking, CDAN 2016. https://doi.org/10.1109/CDAN.2016.7570950.

Johari, S. and Kumar, A.: Algorithmic approach for applying load balancing during task migration in multi-core system. Proceedings of 2014 3rd International Conference on Parallel, Distributed and Grid Computing. 27–32 (2015). doi:10.1109/PDGC.2014.7030710

Kabir, M. H., Shoja, G. C., and Ganti, S. (2014, December). VM placement algorithms for hierarchical cloud infrastructure. In 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (pp. 656-659). IEEE.

Kalra, M., and Singh, S. (2015). A review of metaheuristic scheduling techniques in cloud computing. Egyptian informatics journal, 16(3), 275-295.

Kang, K. D., Alian, M., Kim, D., Huh, J., and Kim, N. S. (2018, October). VIP: Virtual performance-state for efficient power management of virtual machines. In Proceedings of the ACM Symposium on Cloud Computing (pp. 237-248).

Kansal, N. J., and Chana, I. (2016). Energy-aware Virtual Machine Migration for Cloud Computing - A Firefly Optimization Approach. Journal of Grid Computing, 14(2), 327–345. https://doi.org/10.1007/S10723-016-9364-0/METRICS.

Karmakar, K., Banerjee, S., Das, R. K., and Khatua, S. (2022). Utilization aware and network I/O intensive virtual machine placement policies for cloud data center. Journal of Network and Computer Applications, 205, 103442.

Kaur, K., Dhand, T., Kumar, N., and Zeadally, S. (2017). Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers. IEEE Wireless Communications, 24(3), 48–56.

Khosravi, A., Garg, S. K., and Buyya, R. (2013). Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In Euro-Par 2013 Parallel Processing: 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings 19 (pp. 317-328). Springer Berlin Heidelberg.

Kim, K. H., Buyya, R., and Kim, J. (2007, May). Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07) (pp. 541-548). IEEE.

Kliazovich, D., Arzo, S. T., Granelli, F., Bouvry, P. and Khan, S. U.: e-STAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing. Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing. 7–13 (2013) doi:10.1109/GreenCom-iThings-CPSCom.2013.28

Lasica, J. D. (2009). Identity in the Age of Cloud Computing: The next-generation Internet's impact on business, governance and social interaction.

Li, D., Shang, Y. and Chen, C.: Software defined green data center network with exclusive routing. IEEE Conference on Computer Communications. 1743–1751 (2014) doi:10.1109/INFOCOM.2014.6848112

Li, L., Tang, T., and Chou, W. (2015). A rest service framework for fine-grained resource management in container-based cloud. In Proceedings of the 8th IEEE International Conference on Cloud Computing (pp. 645–652).

Liu, J., Wang, S., Zhou, A., Xu, J., and Yang, F. (2020). SLA-driven container consolidation with usage prediction for green cloud computing. Frontiers of Computer Science, 14(1), 42–52. https://doi.org/10.1007/S11704-018-7172-3/METRICS.

LKML: Andrea Arcangeli: [PATCH 00/39] [RFC] AutoNUMA alpha10. https://lkml.org/lkml/2012/3/26/398. Accessed 26 January 2021.

Ma, Z., Shao, S., Guo, S., Wang, Z., Qi, F., and Xiong, A. (2020). Container Migration Mechanism for Load Balancing in Edge Network under Power Internet of Things. IEEE Access, 8, 118405–118416. https://doi.org/10.1109/ACCESS.2020.3004615.

Mahadevan, P., Banerjee, S. and Sharma, P.: Energy proportionality of an enterprise network. Proceedings of the 1st ACM SIGCOMM Workshop on Green Networking, Green Networking '10, 53–59 (2010).

Mann, Z. Á. (2018). Resource optimization across the cloud stack. IEEE Transactions on Parallel and Distributed Systems, 29(1), 169–182.

Mann, Z. A.: Multicore-Aware Virtual Machine Placement in Cloud Data Centers. IEEE Transactions on Computers. 65, 3357–3369 (2016)

Masdari, M., Nabavi, S. S., and Ahmadi, V. (2016). An overview of virtual machine placement schemes in cloud computing. Journal of Network and Computer Applications, 66, 106–127.

Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., and Pendarakis, D. (2010, June). Efficient resource provisioning in compute clouds via vm multiplexing. In Proceedings of the 7th international conference on Autonomic computing (pp. 11-20).

Mishra, A. K., Hellerstein, J. L., Cirne, W., and Das, C. R. (2010). Towards characterizing cloud backend workloads. ACM SIGMETRICS Performance Evaluation Review, 37(4), 34–41. https://doi.org/10.1145/1773394.1773400.

Mohseni, Z., Kiani, V. and Masoud Rahmani, A.: A Task Scheduling Model for Multi-CPU and Multi-Hard Disk Drive in Soft Real-time Systems. International Journal of Information Technology and Computer Science. 1, 1–13 (2019)

Moro, A., Mumolo, E., and Nolich, M. (2009). Ergodic continuous hidden markov models for workload characterization. ISPA 2009 - Proceedings of the 6th International Symposium on Image and Signal Processing and Analysis, 103–108. https://doi.org/10.1109/ISPA.2009.5297771.

Mulahuwaish, A., Korbel, S., and Qolomany, B. (2022). Improving datacenter utilization through containerized service-based architecture. Journal of Cloud Computing, 11(1), 44.
Nardelli, M., Hochreiner, C., and Schulte, S. (2017). Elastic provisioning of virtual machines for container deployment. ICPE 2017 - Companion of the 2017 ACM/SPEC International Conference on Performance Engineering, 5–10.

Niccolini, L., Iannaccone, G., Ratnasamy, S., Chandrashekar, J. and Rizzo, L.: Building a power-proportional software router. Proceedings of the 2012 USENIX Annual Technical Conference. 89–100 (2019)

Onan, A. (2019). Consensus Clustering-Based Undersampling Approach to Imbalanced Learning. Scientific Programming.

Pandit, D., Chattopadhyay, S., Chattopadhyay, M., and Chaki, N. (2014, February). Resource allocation in cloud using simulated annealing. In 2014 Applications and Innovations in Mobile Computing (AIMoC) (pp. 21-27). IEEE.

Panneerselvam, J., Liu, L., Antonopoulos, N., and Bo, Y. (2014). Workload analysis for the scope of user demand prediction model evaluations in cloud environments. Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014, 883–889.

Park, K. S., and Pai, V. S. (2006). CoMon. ACM SIGOPS Operating Systems Review, 40(1), 65–74. https://doi.org/10.1145/1113361.1113374.

Patel, E., and Kushwaha, D. S. (2020). Clustering Cloud Workloads: K-Means vs Gaussian Mixture Model. Procedia Computer Science, 171, 158–167.

Patel, J., Jindal, V., Yen, I. L., Bastani, F., Xu, J., and Garraghan, P. (2015). Workload Estimation for Improving Resource Management Decisions in the Cloud. Proceedings - 2015 IEEE 12th International Symposium on Autonomous Decentralized Systems, ISADS 2015, 25–32.

Patra, M. K., Misra, S., Sahoo, B., and Turuk, A. K. (2022). GWO-Based Simulated Annealing Approach for Load Balancing in Cloud for Hosting Container as a Service. Applied Sciences 2022, Vol. 12, Page 11115, 12(21), 11115. https://doi.org/10.3390/APP122111115.

Patra, S. S. (1 C.E.). Energy-Efficient Task Consolidation for Cloud Data Center, 8(1), 117–142. https://doi.org/10.4018/IJCAC.2018010106.

Pietri, I., and Sakellariou, R. (2014, September). Energy-aware workflow scheduling using frequency scaling. In 2014 43rd International Conference on Parallel Processing Workshops (pp. 104-113). IEEE.

Piraghaj, S. F., Dastjerdi, A. V., Calheiros, R. N., and Buyya, R. (2015b). A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers. 2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS), 368–375.

Piraghaj, S. F., Dastjerdi, A., Calheiros, R. N., and Buyya, R. (2015a). Efficient virtual machine sizing for hosting containers as a service. In roceedings of IEEEWorld Congress on Services. (pp. 31–38).

Prabhakaran, G. and Selvakumar, S. An diverse approach on virtual machines administration and power control in multi-level implicit servers. Journal of Ambient Intelligence and Humanized Computing (2021)

Pudukotai Dinakarrao, S. M.: Self-aware power management for multicore microprocessors. Sustainable Computing: Informatics and Systems. 29, Part B, 100480 (2021) doi:10.1016/j.suscom.2020.100480.

Qiu, M., Ming, Z., Li, J., Gai, K. and Zong, Z.: Phase-Change Memory Optimization for Green Cloud with Genetic Algorithm. IEEE Transactions on Computers. 64, 3528–3540 (2015)

Raj, V. M., and Shriram, R. (2011, March). Power aware provisioning in cloud computing environment. In 2011 International Conference on Computer, Communication and Electrical Technology (ICCCET) (pp. 6-11). IEEE.

Rasheduzzaman, M., Islam, M. A., Islam, T., Hossain, T., and Rahman, R. M. (2014). Task shape classification and workload characterization of google cluster trace. Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014, 893–898.

Saber, T., Thorburn, J., Murphy, L., and Ventresque, A. (2018). VM reassignment in hybrid clouds for large decentralised companies: A multi-objective challenge. Future Generation Computer Systems, 79, 751–764.

Sahinaslan, O., Sahinaslan, E., and Ari, I. S. (2022). A study on the transition to container technologies in data centers. AIP Conference Proceedings, 2483(1), 070002. https://doi.org/10.1063/5.0115600.

Sakamoto, M. and Yamaguchi, S.: Dynamic Memory Allocation in Virtual Machines based on Cache Hit Ratio. Proceedings - 2015 3rd International Symposium on Computing and Networking. 613–615 (2016) doi:10.1109/CANDAR.2015.34.

Shabeera, T. P., Madhu Kumar, S. D., Salam, S. M., and Murali Krishnan, K. (2017). Optimizing VM allocation and data placement for data-intensive applications in cloud using ACO metaheuristic algorithm. Engineering Science and Technology, an International Journal, 20(2), 616–628.

Shekhawat, V. S., Gautam, A., and Thakrar, A. (2018). Datacenter Workload Classification and Characterization: An Empirical Approach. 2018 13th International Conference on Industrial and Information Systems, ICIIS 2018 - Proceedings, 1–7. https://doi.org/10.1109/ICIINFS.2018.8721402.

Shen, S., van Beek, V., and Iosup, A. (2015). Statistical characterization of business-critical workloads hosted in cloud datacenters. Proceedings - 2015 IEEE / ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015, 465–474.

Shi, T., Ma, H., and Chen, G. (2018a). Energy-Aware Container Consolidation Based on PSO in Cloud Data Centers. 2018 IEEE Congress on Evolutionary Computation, CEC 2018 - Proceedings.

Shi, T., Ma, H., and Chen, G. (2018b). Multi-objective container consolidation in cloud data centers. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11320 LNAI, 783–795.

Shroff, G., and Shroff, G. (2011). Virtualization technology. Enterprise

Cloud Computing, 89–103. https://doi.org/10.1017/cbo9780511778476.012.

SMIMITE, O., and AFDEL, K. (2020). Hybrid Solution for Container Placement and Load Balancing based on ACO and Bin Packing. International Journal of Advanced Computer Science and Applications, 11(11), 606–615.

Spicuglia, S., Chen, L. Y., Birke, R., and Binder, W. (2015). Optimizing capacity allocation for big data applications in cloud datacenters. Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, 511–517.

Sturm, R., Pollard, C., and Craig, J. (2017). Managing Containerized Applications. Application Performance Management (APM) in the Digital Enterprise, 177–185. https://doi.org/10.1016/B978-0-12-804018-8.00013-9.

Sun, X., Ansari, N., and Wang, R. (2016). Optimizing Resource Utilization of a Data Center. IEEE Communications Surveys and Tutorials, 18(4), 2822–2846. https://doi.org/10.1109/COMST.2016.2558203.

Tan, B., Ma, H., and Mei, Y. (2019). A Hybrid Genetic Programming Hyper-Heuristic Approach for Online Two-level Resource Allocation in Container-based Clouds. 2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings, 2681–2688.

Tang, Z., Wang, Y., Wang, Q. and Chu, X.: The impact of GPU DVFS on the energy and performance of deep Learning: An Empirical Study. e-Energy 2019 - Proceedings of the 10th ACM International Conference on Future Energy Systems 315–325 (2019) doi:10.1145/3307772.3328315.

Tang, Z., Zhou, X., Zhang, F., Jia, W., and Zhao, W. (2019). Migration Modeling and Learning Algorithms for Containers in Fog Computing.

IEEE Transactions on Services Computing, 12(5), 712–725.

Terzi, C. and Korpeoglu, I.: 60 GHz wireless data center networks: A survey. Computer Networks. 185, 107730 (2021)

Tomes, E. and Altiparmak, N.: A Comparative Study of HDD and SSD RAIDs' Impact on Server Energy Consumption. Proceedings - IEEE International Conference on Cluster Computing. 625–626 (2017)

U-Chupala, P., Watashiba, Y., Ichikawa, K., Date, S., and Iida, H. (2017). Container Rebalancing: Towards Proactive Linux Containers Placement Optimization in a Data Center. Proceedings - International Computer Software and Applications Conference, 1, 788–795.

Ukidave, Y., Li, X. and Kaeli, D.: Mystic: Predictive Scheduling for GPU Based Cloud Servers Using Machine Learning. Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016 353–362 (2016) doi:10.1109/IPDPS.2016.73

Urgaonkar, R., Kozat, U. C., Igarashi, K., and Neely, M. J. (2010, April). Dynamic resource allocation and power management in virtualized data centers. In 2010 IEEE Network Operations and Management Symposium-NOMS 2010 (pp. 479-486). IEEE.

Venkatesan, V., Tay, Y. C., Zhang, Y. I. and Wei, Q. A 3-level cache miss model for a nonvolatile extension to transcendent memory. Proceedings of the International Conference on Cloud Computing Technology and Science. 218–225 (2015)

Wang, J. and Wang, B.: A hybrid main memory applied in virtualization environments. 2016 1st IEEE International Conference on Computer Communication and the Internet. 413–417 (2016) doi:10.1109/CCI.2016.7778955

Wang, K., Lin, M., Ciucu, F., Wierman, A., and Lin, C. (2015). Characterizing the impact of the workload on the value of dynamic resizing in data centers. Performance Evaluation, 85–86, 1–18.

Wang, N., Ho, K. H. and Pavlou, G.: AMPLE: An adaptive traffic engineering system based on virtual routing topologies. IEEE Communications Magazine. 50, 185–191 (2012)

Wikipedia access traces — WikiBench. (n.d.). Retrieved March 21, 2022, from http://www.wikibench.eu/?page id=60.

Wu, W., Xia, W., Yu, Z. and Liu, Q.: Exploring the potential of coupled array of SSD and HDD for multi-Tenant. 2018 3rd IEEE International Conference on Cloud Computing and Big Data Analysis. 653–657 (2018) doi:10.1109/ICCCBDA.2018.8386596

Xiao, P., Ni, Z., Liu, D. and Hu, Z. Improving the energy-efficiency of virtual machines by I/O compensation. Journal of Supercomputing 77, 11135–11159 (2021).

Xu, J., and Fortes, J. A. B. (2010). Multi-objective virtual machine placement in virtualized data center environments. Proceedings - 2010 IEEE/ACM International Conference on Green Computing and Communications, GreenCom 2010, 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom 2010, 179–188.

Xu, Z., Dong, F., Jin, J., Luo, J. and Shen, J.: GScheduler: Optimizing resource provision by using GPU usage pattern extraction in cloud environments. 2017 IEEE International Conference on Systems, Man, and Cybernetics. 3225–3230 (2017).

Yan, W., Chen, J., and Li, L. (2018). A power-aware ACO algorithm for the cloud computing platform. ACM International Conference Proceeding

Series, 1–6. https://doi.org/10.1145/3290420.3290428.

Yang, H. and Yan, X.: Memory Coherency Based CPU-Cache-FPGA Acceleration Architecture for Cloud Computing. Proceedings - 2015 2nd International Conference on Information Science and Control Engineering. 304–307 (2015) doi:10.1109/ICISCE.2015.74

Yaqub, E., Yahyapour, R., Wieder, P., Jehangiri, A. I., Lu, K., and Kotsokalis, C. (2014). Metaheuristics-based planning and optimization for SLA-aware resource management in PaaS clouds. Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014, 288–297. https://doi.org/10.1109/UCC.2014.38.

Yin, J., Lu, X., Zhao, X., Chen, H., and Liu, X. (2015). BURSE: A bursty and self-similar workload generator for cloud computing. IEEE Transactions on Parallel and Distributed Systems, 26(3), 668–680.

Zhang et al., (2011) Characterizing Task Usage Shapes in Google Compute Clusters – Google Research. Retrieved March 17, 2022, from https://research.google/pubs/pub37201

Zhang, H., Jiang, G., Yoshihira, K., and Chen, H. (2014). Proactive workload management in hybrid cloud computing. IEEE Transactions on Network and Service Management, 11(1), 90–100.

Zhang, W., Chen, L., Luo, J., and Liu, J. (2022). A two-stage container management in the cloud for optimizing the load balancing and migration cost. Future Generation Computer Systems, 135, 303–314.

Zhao, W., Peng, Y., Xie, F., and Dai, Z. (2012). Modeling and simulation of cloud computing: A review. Proceedings - 2012 IEEE Asia Pacific Cloud Computing Congress, APCloudCC 2012, 20–24.

Zheng, S., Huang, F., Li, C., and Wang, H. (2021). A Cloud Resource Prediction and Migration Method for Container Scheduling. 2021 IEEE Conference on Telecommunications, Optics and Computer Science, TOCS 2021, 76–80. https://doi.org/10.1109/TOCS53301.2021.9689034.

Zhong, Z., and Buyya, R. (2020). A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources. ACM Transactions on Internet Technology, 20(2).

Zhu, Q. et al.: Hibernator: Helping disk arrays sleep through the winter. Proceedings of the 20th ACM Symposium on Operating Systems Principles. 177–190 (2005) doi:10.1145/1095810.1095828

List of Publications

1. Avita Katal, Susheela Dahiya and Tanupriya Choudhury, “Energy efficiency in cloud computing data center: a survey on hardware technologies”, Cluster Computing, 25, 675–705 (2022). https://doi.org/10.1007/s10586-021-03431-z

2. Avita Katal, Susheela Dahiya and Tanupriya Choudhury, “Energy efficiency in cloud computing data centers: a survey on software technologies”, Cluster Computing, 26, 1845–1875 (2023). https://doi.org/10.1007/s10586-022-03713-0

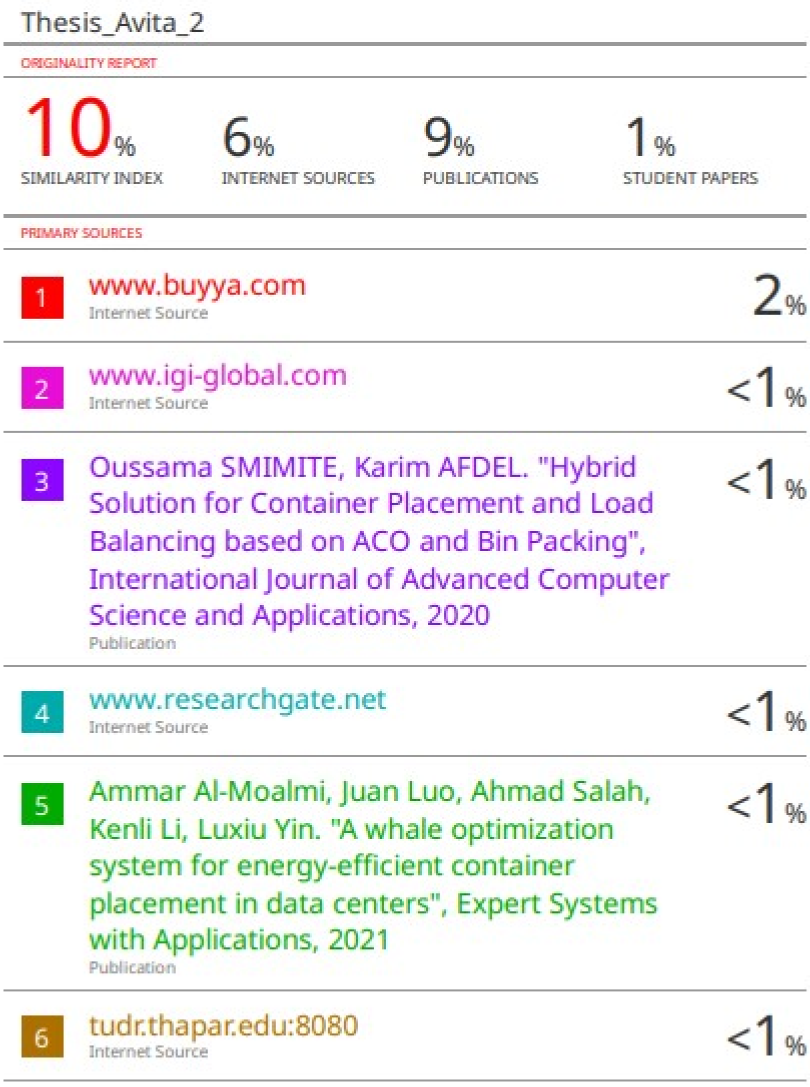
3. Avita Katal, Susheela Dahiya and Tanupriya Choudhury, “Workload Characterization and Classification: A Step Towards Better Resource Utilization in a Cloud Data Center”, Pertanika Journal of Science and Technology, 31(5), 2559–2575 (2023) https://doi.org/10.47836/pjst.31.5.27

4. Avita Katal, Tanupriya Choudhury, Susheela Dahiya. (2023). Comparison and Analysis of Container Placement Algorithms in Cloud Data Center. In: Rathore, V.S., Piuri, V., Babo, R., Ferreira, M.C. (eds) Emerging Trends in Expert Applications and Security. ICETEAS 2023. Lecture Notes in Networks and Systems, vol 682. Springer, Singapore. https://doi.org/10.1007/978-981-99-1946-822

5. Avita Katal, Tanupriya Choudhury, Susheela Dahiya, “Energy optimized container placement for cloud data centers: a meta-heuristic approach”, Journal of Supercomputing (2023). https://doi.org/10.1007/s11227-023-05462-2

UPES
UNIVERSITY OF TOMORROW

Plagiarism Report



Thesis_Avita_2

ORIGINALITY REPORT

10%	6%	9%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.buyya.com Internet Source	2%
2	www.igi-global.com Internet Source	<1%
3	Oussama SMIMITE, Karim AFDEL. "Hybrid Solution for Container Placement and Load Balancing based on ACO and Bin Packing", International Journal of Advanced Computer Science and Applications, 2020 Publication	<1%
4	www.researchgate.net Internet Source	<1%
5	Ammar Al-Moalmi, Juan Luo, Ahmad Salah, Kenli Li, Luxiu Yin. "A whale optimization system for energy-efficient container placement in data centers", Expert Systems with Applications, 2021 Publication	<1%
6	tudr.thapar.edu:8080 Internet Source	<1%