Efficient Resource Management Techniques for Cloud Computing Environment

Thesis submitted

by

Sounak Banerjee

DOCTOR OF PHILOSOPHY (Engineering)

Department of Computer Science and Engineering

Faculty Council of Engineering and Technology

Jadavpur University

Kolkata, West Bengal-700032, India

2026

Efficient Resource Management Techniques for Cloud Computing Environment

Thesis submitted

by

Sounak Banerjee

DOCTOR OF PHILOSOPHY (Engineering)

under the supervision of

Prof. Sarbani Roy

Department of Computer Science and Engineering

JADAVPUR UNIVERSITY

Kolkata, West Bengal-700032, India

2026

Index No: 247/20/E

1. **Title of the Thesis:**

   EFFICIENT RESOURCE MANAGEMENT TECHNIQUES FOR CLOUD COMPUTING ENVIRONMENT

2. **Name, Designation and Institution of the Supervisor:**

   Dr. Sarbani Roy
   Professor
   Department of Computer Science and Engineering
   Jadavpur University
   Kolkata-700032

3. **List Of Publication:**

   (a) **Journal:**

      i. Sounak Banerjee, Sarbani Roy, and Sunirmal Khatua. Efficient resource utilization using multi-step-ahead workload prediction technique in cloud. The Journal of Supercomputing, 77:10636–10663, 2021.

      ii. Sounak Banerjee, Sarbani Roy, and Sunirmal Khatua. Sla-aware stochastic load balancing in dynamic cloud environment. Journal of Grid Computing, 19:1–24, 2021.

      iii. Sounak Banerjee, Sarbani Roy, and Sunirmal Khatua. Towards energy and QoS aware dynamic VM consolidation in a multi-resource cloud. Future Generation Computer Systems, 1;157:376-91, 2024.

   (b) **Conference:**

      i. Sounak Banerjee, Sarbani Roy, and Sunirmal Khatua. Game theory based energy-aware virtual machine placement towards improving resource efficiency in homogeneous cloud data center. In 2022 IEEE Calcutta Conference (CALCON), pages 293–298. IEEE, 2022.

4. **List of Patents:** NONE

5. **List of Presentations in International Conference:**

   (a) Sounak Banerjee, Sarbani Roy, and Sunirmal Khatua. Game theory based energy-aware virtual machine placement towards improving resource efficiency in homogeneous cloud data center. In 2022 IEEE Calcutta Conference (CALCON), pages 293–298. IEEE, 2022.

Statement of Originality

I, **Sounak Banerjee** registered on **18th February 2020**, having registration number-**1022004004**, do hereby declare that this thesis entitled "**Efficient Resource Management Techniques for Cloud Computing Environment**" contains literature survey and original research work done by the undersigned candidate as part of Doctoral studies.

All information in this thesis have been obtained and presented in accordance with existing academic rules and ethical conduct. I declare that, as required by these rules and conduct, I have fully cited and referred all materials and results that are not original to this work.

I also declare that I have checked this thesis as per the "Policy on Anti Plagiarism, Jadavpur University, 2019", and the level of similarity as checked by iThenticate software is 2%.

Sounak Banerjee

Signature of Candidate

Date: 30.03.2026

Sarbani Roy 30/03/2026

Professor
Comp. Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Dr. Sarbani Roy,

Professor,

Department of Computer Science and Engineering,

Jadavpur University.

Certificate from the Supervisor

*This is to certify that the thesis entitled "**Efficient Resource Management Techniques for Cloud Computing Environment**" submitted by **Sounak Banerjee**, who got his name registered on 18th February 2020, for the award of Ph.D. (Engineering) degree of Jadavpur University, is absolutely based upon his own work under the supervision of Dr. Sarbani Roy that neither his thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.*

Sarbani Roy 30/03/2026

Dr. Sarbani Roy

Professor,

Department of Computer Science and Engineering,

Jadavpur University.

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Dedicated to

The Almighty, and my family

Acknowledgements

This thesis represents the culmination of my perseverance and the invaluable contributions of many who supported my Doctor of Philosophy in Engineering at Jadavpur University. Without the guidance of my supervisor, collaboration with fellow researchers, and unwavering family support, completing this dissertation would have been impossible.

I am profoundly grateful to my supervisor, Professor Sarbani Roy from the Department of Computer Science and Engineering at Jadavpur University. Her steadfast support, encouragement, guidance, and patience have been instrumental in shaping this research. Her insightful feedback and constructive critiques directed my investigations while encouraging independent exploration and intellectual growth.

I would also like to extend my sincere gratitude to Dr. Sunirmal Khatua, Assistant Professor from the Department of Computer Science and Engineering at University of Calcutta, for his invaluable support throughout this research journey. His encouragement and assistance have been greatly appreciated and have contributed significantly to the completion of this work.

My sincere thanks go to the Research Advisory Committee members, Professor Nandini Mukherjee and Dr. Chandreyee Chowdhury, for their diligent biannual assessments throughout my doctoral studies.

I deeply appreciate the Heads of the Department, Professor Mahantapas Kundu, Professor Anupam Sinha, Professor Nandini Mukherjee along with Nirmalya Chowdhury, from the Department of Computer Science and Engineering at Jadavpur University, for granting me access to the DST-FIST lab. I am also grateful for the unwavering support of the entire faculty and non-teaching staff within our department.

I acknowledge the UGC-NET JRF Fellowship scheme (UGC-Ref. No.: 3709/(NET-DEC 2018)) for financial support from February 2020 to February 2025, which made this research possible.

It has been a delightful and enriching experience to work within the DST-FIST Lab at the Department of Computer Science and Engineering, Jadavpur University. The congenial working atmosphere has been a great boon, and I wish to express my sincere appreciation to all my labmates and friends from DST-FIST lab. In particular, I would like to extend my gratitude to Joy da, Joyeeta di, Moumita di, Priya di, Asif da, Subhayan, Rituparna di, Anwesha, Manjarini, Bidisha, Soumyadeep, and Rohit for their camaraderie, cooperation, and valuable discussions during our collaborative efforts.

I am grateful to all my teachers throughout my academic journey and to my friends for their inspiration and support.

Finally, I extend heartfelt thanks to my family for their unwavering presence through every challenge.

Sounak Banerjee

Jadavpur, Kolkata

Sounak Banerjee

Date: 30.03.2026

Abstract

Cloud computing has revolutionized computational resource access but introduced complex management challenges that traditional static allocation approaches cannot address, necessitating sophisticated strategies for dynamic workloads, energy efficiency, and service quality guarantees. This thesis presents a comprehensive framework for intelligent cloud resource management addressing three fundamental challenges through interconnected solutions that coordinate optimization across multiple time horizons while maintaining service level agreements and system stability. The first contribution develops proactive resource management through optimal VM placement, employing machine learning-based workload prediction that identifies temporal patterns through clustering and develops specialized models for each workload category, combined with statistical-stochastic frameworks treating resource consumption probabilistically for risk-aware allocation decisions, integrated with heuristic-based and game-theoretic VM placement strategies that achieve superior energy efficiency through optimal consolidation. The second contribution addresses temporal limitations through QoS-aware load balancing implementing probabilistic overload detection using stochastic resource modeling to anticipate deficit situations before SLA violations occur, coupled with intelligent migration optimization algorithms addressing VM selection and destination placement while incorporating cumulative SLA violation tracking for long-term service quality impact assessment, successfully transforming reactive resource management into proactive load balancing. The third contribution completes the framework through energy-efficient dynamic VM consolidation using Resource-Optimized VM Consolidation that employs stochastic load imbalance detection to identify consolidation opportunities while maintaining load balancing compatibility, resource intensity-aware VM distribution through game-theoretic optimization achieving Nash equilibrium solutions balancing energy efficiency with system resilience, and multi-objective optimization integrating energy minimization with migration frequency control. Comprehensive experimental validation using real-world workload traces demonstrates the integrated framework's effectiveness: proactive placement achieves 12% energy reduction with superior consolidation efficiency, SLA-aware load balancing delivers 10-39% reduction in migrations and 17-54% fewer overloaded hosts while maintaining utilization efficiency, and energy-efficient consolidation provides 21-65% reduction in active hosts, 17-46% energy decrease, and 37-70% fewer load imbalances compared to existing approaches, with coordinated optimization achieving simultaneous energy efficiency, service quality maintenance, and resource utilization effectiveness across diverse workload scenarios. The research establishes a stochastic modeling foundation enabling sophisticated decision-making under uncertainty, successfully bridging theoretical optimization with practical deployment requirements through modular design that provides measurable improvements across all performance dimensions, demonstrating that effective cloud resource management requires coordinated optimization rather than isolated solutions and that aggressive energy optimization can be achieved without

compromising system resilience or service quality, ultimately advancing sustainable and efficient cloud computing infrastructure management while enabling providers to deliver operational excellence through simultaneous optimization of competing objectives.

Contents

List of Figures

List of Tables

List of Symbols

$\mathcal{H}$	Set of physical hosts in the data center
H_i	Physical host i
$\mathcal{R}$	Set of resource types
r_k	Resource type k
d	Number of resource dimensions
M	Total number of hosts in the data center
P	Total number of VM types supported
C_{ir_k}	Capacity of resource r_k in host H_i
V_j	Virtual machine j
$\mathcal{V}^t$	Set of active VMs at time t
$\mathcal{V}_i^t$	Set of VMs allocated to H_i at time t
n_i^t	Number of VMs on host H_i at time t
C_{jr_k}	Capacity of r_k in V_j
$D_{jr_k}^t$	Demand for r_k by V_j at time t
Δt	Time period for averaging resource demand
$\overline{D}_{jr_k}^t$	Average demand for r_k by V_j during time period Δt
N^t	Number of VMs running at time t
$\mathcal{X}^t$	VM allocation matrix at time t
X_{ij}^t	allocation indicator for V_j onto H_i at time t
$\mathcal{X}^{t^*}$	VM allocation matrix after migration at time t
$\mathcal{S}_{Mig}^t$	Set of migrating VMs at time t
mig^t	Number of migrating VMs at time t
$\mathcal{H}_{active}^t$	Set of active hosts at time t
$\mathcal{U}_{ir_k}^t$	Utilization of H_i for r_k at time t
$\mathcal{U}_{ir_k}^{t^*}$	Utilization of H_i for r_k after migration
$DC_{r_k}^t$	DC utilization for r_k at time t
P_i^t	Power consumption of H_i at time t
H_{max}^i	Maximum power consumed by fully utilized H_i
$\mathcal{U}_{iCPU}^t$	CPU utilization of H_i at time t
E_i	Total energy consumption of H_i over period T
$\mathcal{D}_{jr_k}^T$	Time-series sequence V_j's usage on r_k
$L_{r_k}^t(H_i)$	Load of H_i for r_k at time t

$L^{t^*}_{r_k}(H_i)$	Load of H_i for r_k after migration at time t
$\widehat{DC^t_{r_k}}$	Normalized utilization of DC for r_k at time t
U_θ	Predefined Utilization threshold for overload detection
$\Pr^t_{\text{over}}(H_i)^{r_k}$	Probability of overload for r_k in H_i at time t
θ_{over}	Threshold for overloading probability
θ_{under}	Threshold for under-loading probability
τ	Under-utilization threshold
$\Pr^t_{\text{under}}(H_i)^{r_k}$	Probability of under-loading for resource r_k in host H_i at time t
$\mathcal{H}^t_{ovr}$	Set of overloaded hosts at time t
$\mathcal{H}^t_{undr}$	Set of under-loaded hosts at time t
λ^t_i	Set of overutilized resources in H_i at time t
Υ^t_i	Set of non-overutilized resources in H_i at time t
$PVal^t_{ir_k}(V_x)$	Performance value of V_x for r_k in H_i at time t
MS^t_{ix}	Migration score of V_x from H_i at time t
$\omega^t_{ir_k}$	Weight for r_k in H_i at time t
δt	Monitoring interval between consecutive time points
t_c	Current monitoring time point
t_{c+1}	Next monitoring time point
t^x_s	Start time of VM V_x
$Cu^x_{r_k}(t_c)$	Cumulative unallocated resource r_k for V_x from t^x_s to time t_c
$\gamma^x_{r_k}(t)$	Percentage of unallocated resource r_k for V_x at time t
ξ	Minimum percentage of resource requirements to be satisfied (SLO parameter)
$ORA^{t_c}_{r_k}(H_i)$	Aggregate overloaded resource amount of type r_k in H_i at time t_c
$\omega^{t_c}_{jr_k}$	Fraction of overloaded resource r_k allocated to V_j at time t_c
α	Iteration index for probabilistic estimation
Δ_k	Fractional increment for resource r_k
η	Fractional increment parameter
$\mathcal{N}^t_{Mig}$	Complete set of migrating VMs from all overloaded hosts at time t
$\mathcal{N}^{it}_{Mig}$	Set of VMs selected for migration from host H_i at time t
$\mathcal{D}^t_{H_i}$	Set of potential destination hosts for VMs from host H_i at time t
$\mathcal{V}^t_d$	Set of VMs hosted by destination host H_d at time t
$PVal^t_{r_k}(H_d)$	Performance value of destination host H_d for resource r_k at time t
DPS^t_{id}	Destination potentiality score for host H_d from source host H_i at time t
$\mathbf{D}$	Demand matrix of size $\mathsf{n}^t_i \times \mathsf{d}$
$\mathsf{D}_{j,k}$	Demand of VM V_j for resource r_k (element of demand matrix $\mathbf{D}$)
V_γ	Imaginary VM representing optimal migration candidate characteristics
$\zeta_{\gamma j}$	Euclidean distance between VM V_j and imaginary VM V_γ
V_β	Most suitable VM candidate for migration
$\mathcal{H}^t_{des}$	Set of destination hosts at time t
H_α	Imaginary host for resource estimation
$\mathcal{L}^t_{req}$	Set of minimum required resource amounts at time t
$\mathcal{L}^t_{r_k}$	Minimum required amount of resource r_k at time t

∇_{r_k}	Increment amount for resource r_k
∂	Increment rate parameter
$\mathcal{H}^t_{\mathsf{nor}}$	Set of normally-loaded hosts at time t
$\mathcal{H}^t_{\mathsf{idl}}$	Set of idle hosts at time t
V_α	Imaginary representative VM with average resource demands
$\mathcal{H}^t_{mat}$	Set of feasible hosts at time t
$\mathcal{J}^t_{r_k}$	Remaining required amount of resource r_k at time t
H_β	Ideal host with capacity matching remaining resource needs
$\xi_{(i,\beta)}$	Euclidean distance from host H_i to ideal host H_β
$\mathcal{H}^t_{sorted}$	Set of idle hosts sorted by distance from ideal host
$\mathcal{G}$	n-player finite strategic game
$\mathcal{S}$	Set of strategy profiles
S_j	Strategy set of player (VM) V_j
$\mathcal{F}$	Set of payoff functions for game
$\mathcal{Q}$	Strategy profile
Q_j	Specific strategy of player V_j
$F_j(\mathcal{Q})$	Payoff function for player V_j given strategy profile $\mathcal{Q}$
$\varphi^t_{\mathcal{Q}}$	Payoff generated for strategy profile $\mathcal{Q}$ at time t
Ψ^t_j	Average resource demand of VM V_j at time t
$\widehat{D^t_{jr_k}}$	Normalized demand of VM V_j for resource r_k at time t
$\mathcal{Q}^{max}$	Strategy profile yielding maximum payoff
Q^{max}_j	Optimal strategy for player V_j
$\mathcal{Q}^{max}_{-j}$	Strategies of all players except player V_j in profile $\mathcal{Q}^{max}$
PF^t_i	Preference factor of host H_i at time t
σ^t_{PF}	Standard deviation of preference factors at time t
$\Omega^t_{\mathcal{Q}}$	Feasibility indicator for strategy profile $\mathcal{Q}$ at time t
Γ^t_i	Total utilization of all resources in host H_i at time t
Δ^t_i	Utilization gap of host H_i at time t
r_m	Maximally utilized resource in host H_i
$\mathcal{U}^t_{r_m}$	Utilization of maximally utilized resource at time t
$\mathcal{H}^t_{sel}$	Set of selected hosts at time t

Chapter 1

Introduction

1.1 Background and Motivation

Cloud computing has revolutionized the way organizations access, deploy, and manage computational resources, offering unprecedented scalability, flexibility, and cost-effectiveness [1]. As this paradigm continues to evolve, the efficient management of cloud resources has become a cornerstone for delivering high-quality services while maintaining operational sustainability [2]. The dynamic and heterogeneous nature of cloud environments presents complex challenges that require sophisticated resource management strategies to optimize performance, ensure service quality, and minimize environmental impact [3]. This thesis investigates comprehensive approaches to address these challenges through innovative techniques in workload prediction, load balancing, and energy-efficient resource consolidation.

Cloud computing environments are characterized by their dynamic nature, where resource demands fluctuate continuously based on user behavior, application requirements, and temporal patterns [4]. This variability presents significant challenges for cloud service providers who must ensure optimal resource utilization while maintaining service level agreements (SLAs) and minimizing operational expenses [5]. The traditional approaches to resource management, which rely on static allocation strategies and reactive scaling mechanisms, often fall short in addressing the

complex demands of modern cloud workloads [6].

The importance of efficient resource management in cloud computing extends beyond mere cost optimization [7]. Poor resource allocation decisions can lead to cascading effects including service degradation, energy wastage, and reduced customer satisfaction [8]. Furthermore, with the growing emphasis on environmental sustainability and green computing initiatives, cloud providers are under increasing pressure to minimize their carbon footprint while maintaining high-performance service delivery [9].

1.2 Research Challenges and Scope

Contemporary cloud computing environments face several interconnected challenges that necessitate sophisticated resource management strategies [10]. The primary issues include:

- **Resource Allocation Inefficiency**: Traditional resource allocation methods often result in either over-provisioning, leading to resource wastage and increased costs, or under-provisioning, resulting in performance degradation and SLA violations [11]. The lack of accurate workload prediction mechanisms exacerbates this problem, making it difficult to proactively allocate resources based on anticipated demands [12].

- **Load Balancing Complexities**: As cloud applications become more complex and distributed, ensuring optimal load distribution across available resources while maintaining quality of service (QoS) requirements becomes increasingly challenging [13]. Existing load balancing techniques often fail to consider the multi-dimensional nature of cloud resources and the diverse QoS requirements of different applications [14].

- **Energy Consumption Concerns**: The rapid growth of cloud infrastructure has led to significant energy consumption, with data centers accounting for approximately 1-2% of global electricity usage [15]. Virtual machine (VM) sprawl and inefficient consolidation strategies contribute to unnecessary energy consumption, making energy-efficient resource management a critical concern for sustainable cloud operations [16].

These challenges are interconnected and require holistic solutions that address workload prediction, load balancing, and energy efficiency simultaneously. The complexity is further amplified by the heterogeneous nature of cloud workloads, varying user requirements, and the need for real-time decision making in dynamic environments.

1.3 Research Objectives

This thesis aims to address the aforementioned challenges through the development of comprehensive resource management techniques that enhance efficiency, performance, and sustainability in cloud computing environments. The primary objectives are:

1. **To develop advanced workload prediction techniques** that enable proactive resource allocation and improve virtual machine placement decisions in cloud environments.

2. **To design QoS-aware load balancing mechanisms** that optimize resource utilization while maintaining service quality and meeting diverse application requirements.

3. **To create efficient virtual machine consolidation approaches** that minimize energy consumption while ensuring performance guarantees and system stability.

The achievement of these objectives requires an interdisciplinary approach that combines theoretical foundations from machine learning, optimization theory, and distributed systems with practical implementation considerations for cloud environments. The research methodology encompasses algorithm design, mathematical modeling, simulation-based evaluation, and performance analysis using real-world datasets. By addressing these objectives systematically, this thesis aims to provide a comprehensive framework that bridges the gap between theoretical advancements and practical deployment in contemporary cloud computing infrastructures.

1.4 Research Contributions

This thesis makes three significant interconnected contributions to cloud resource management :

1.4.1 Workload Prediction Techniques for Efficient VM Placement

Development of sophisticated workload prediction models leveraging machine learning and statistical techniques to forecast resource demands with high accuracy. These models incorporate historical usage patterns, temporal correlations, and application characteristics to predict future workload requirements. The predicted information optimizes virtual machine placement decisions, enabling proactive rather than reactive resource allocation, significantly reducing resource contention and improving system performance.

However, while accurate workload prediction improves initial VM placement, the dynamic nature of cloud workloads requires continuous load redistribution as demand patterns change, leading to the second contribution.

1.4.2 QoS-Aware Load Balancing in Dynamic Environments

Design of intelligent load balancing algorithms that consider multiple quality of service parameters while distributing workloads across available resources. Unlike traditional approaches focusing solely on resource utilization metrics, the proposed solution incorporates application-specific QoS requirements, network latency, processing capabilities, and service-level agreements into load distribution decisions. This multi-objective optimization ensures service quality maintenance while achieving optimal resource utilization.

Integration of workload prediction with QoS-aware load balancing creates synergistic effects where predicted patterns inform load balancing decisions, enabling proactive distribution strategies. However, optimal load balancing may still result in resource fragmentation and energy inefficiency across multiple physical servers [17], necessitating the third contribution.

1.4.3 VM Consolidation for Energy Optimization

Development of advanced virtual machine consolidation techniques addressing energy efficiency in cloud data centers. The approach utilizes intelligent migration strategies and consolidation algorithms that minimize active physical servers while meeting performance requirements. The solution incorporates predictive analytics to anticipate future resource needs and balances energy savings with system stability and performance guarantees.

The consolidation process leverages insights from both workload prediction and load balancing components. Workload predictions identify optimal consolidation opportunities by forecasting low utilization periods, while QoS-aware load balancing ensures consolidation decisions maintain service quality [18]. This creates a holistic resource management framework where each component enhances the other's effectiveness, resulting in a comprehensive solution addressing prediction, distribution,

and optimization of cloud resources.

To better illustrate the interaction among the proposed components, consider a cloud-hosted e-commerce platform that experiences fluctuating traffic patterns throughout the day. During normal business hours the system operates under moderate load, but traffic rises sharply during a flash sale and drops to near-idle conditions after midnight. In the first phase, the Workload Prediction component analyzes historical traffic logs and temporal patterns to forecast the upcoming surge several minutes in advance, triggering proactive VM provisioning rather than waiting for performance degradation to occur. As the surge begins, the QoS-Aware Load Balancer dynamically redistributes incoming requests across newly provisioned VMs, ensuring that checkout transactions — which carry strict latency SLAs — are prioritized over background analytics jobs. Finally, as traffic subsides post-midnight, the VM Consolidation module identifies underutilized physical servers, migrates residual workloads onto fewer active hosts, and powers down idle servers — directly reducing energy consumption without violating any active SLAs. This sequential interplay illustrates how workload prediction informs load balancing decisions, and how both together create optimal conditions for energy-efficient consolidation, forming a self-reinforcing, closed-loop resource management cycle.

To validate this framework under realistic conditions, this research employs the Google Cluster Usage Traces [19] and the PlanetLab CPU traces [20] — both sourced from production-scale cloud environments. These datasets capture diverse workload types, bursty traffic patterns, and realistic usage variability, ensuring that the proposed techniques are evaluated against practical cloud scenarios and remain comparable with existing literature.

1.5 Thesis Organization

This thesis systematically addresses the research objectives through seven chapters, progressing from problem identification to solution development and experimental validation.

- **Chapter 2: Literature Survey** reviews existing research in workload prediction, load balancing, and energy-efficient consolidation, identifying current limitations and establishing theoretical foundations.
- **Chapter 3: Workload Prediction Framework** presents the advanced prediction architecture, detailing multi-layer prediction systems, uncertainty quantification mechanisms, and adaptive learning capabilities.
- **Chapter 4: QoS-Aware Load Balancing** describes the multi-objective load balancing framework, including optimization algorithms, QoS-aware routing mechanisms, and adaptive threshold systems.
- **Chapter 5: Energy-Efficient VM Consolidation** presents the integrated consolidation system with thermal-aware placement, migration cost optimization, and proactive consolidation strategies.
- **Chapter 6: Conclusion and Future Work** summarizes research contributions, discusses practical implications, and identifies future research directions.

The thesis combines theoretical foundations from machine learning, optimization theory, and distributed systems with practical implementation considerations. The methodology encompasses algorithm design, mathematical modeling, simulation-based evaluation, and performance analysis using industry-standard datasets.

Chapter 2

Literature Survey

2.1 Introduction

Cloud computing presents complex resource management challenges requiring sophisticated strategies for workload prediction, VM placement, dynamic load balancing, and energy-efficient consolidation. This chapter reviews existing literature across three critical areas that form the foundation of this thesis. The review examines key contributions, methodological approaches, and limitations in each area, organizing findings through comprehensive analysis and categorized comparisons.

2.2 Workload Prediction and Virtual Machine Placement

Early foundational work established that randomized algorithms outperform deterministic approaches in virtualized environments [21]. However, modeling real-world resource usage remains challenging due to system complexity [22].

2.2.1 Workload Prediction Approaches

Traditional time-series methods dominated early workload prediction research. Autoregressive models became fundamental tools for CPU load prediction [23]. They offered low computational overhead and reasonable accuracy. However, moving average and ARIMA-based methods struggle with dynamic cloud workloads [24, 25]. Short-term prediction presents particular challenges due to resource dynamicity [26].

Machine learning approaches emerged to overcome these limitations. Pioneering work combined ANN models with linear regression for empirical resource usage prediction [27]. Self-adaptive prediction using ensemble models and fuzzy neural networks followed [28]. Hybrid approaches combined k-means clustering with Extreme Learning Machines [29]. These methods addressed the ineffectiveness of linear basis functions in capturing complex data relationships.

Deep learning architectures revolutionized workload prediction capabilities. LSTM networks effectively capture non-linear patterns through specialized memory mechanisms [30]. They enable accurate host load prediction. GRU-based Encoder-Decoder networks provide multi-step-ahead prediction [31]. They automatically learn periodicity and trends in time-series data. The architectures adaptively select relevant input features through joint encoder-decoder training. Deep learning models were preferred over traditional approaches primarily because of their ability to handle long-term temporal dependencies. Methods like ARIMA rely on fixed-window assumptions and struggle with dynamic, non-linear workload patterns. Machine learning approaches improved on this but remained limited to shorter prediction windows. LSTM networks address this through memory cells that selectively retain past information over time, making them naturally suited for workloads where historical patterns strongly influence future demand. GRU-based models extend this further by enabling multi-step prediction while automatically learning trends

and periodicity — without requiring manual feature engineering. Deep learning was therefore adopted where workload patterns were complex, non-linear, and temporally extended — conditions where shallower models consistently underperformed.

Energy-aware prediction frameworks enable sustainable cloud operations. Ensemble algorithms forecast energy efficiency [32]. Frameworks combining machine learning clustering with stochastic theory achieve precise workload prediction and energy savings [33].

TABLE 2.1: Workload Prediction Approaches Comparison

Category	**Relevant Works**	**Resource Focus**	**Prediction Horizon**	**Key Advantage**	**Primary Limitation**
Time-Series	[23],[24],[25], [32],[34]	CPU, Energy	Short to Long-term	Low computational overhead, established methods	Linear assumptions, poor for variable data
Machine Learning	[27],[28],[29], [35],[36],[37]	Multi-resource	Single to Multi-step	Non-linear pattern capture, adaptability	Limited prediction windows, static clustering
Deep Learning	[12],[30],[31], [38],[39],[40]	Host Load	Multi-step	Long-term dependencies, complex patterns	Single resource focus, isolated approaches
Hybrid Methods	[33],[41],[42], [43]	Multi-resource	Adaptive	Combined methodologies, flexibility	Limited adaptability, computational overhead

2.2.2 VM Placement Approaches

Early VM placement research established that randomized algorithms outperform deterministic approaches [44]. Adaptive resource management strategies that integrate energy and power considerations have been pioneered [45]. These approaches maintain service levels while optimizing resource allocation. Classical heuristic approaches provided foundations for VM placement optimization. Best Fit Decreasing algorithms and bin-packing variants were developed [46, 47]. However, these methods often failed to consider energy efficiency explicitly.

Advanced placement strategies have evolved beyond simple heuristics. Energy-efficient algorithms that consider computational requirements and thermal characteristics have been developed [48, 49]. Machine learning-based approaches for quality-aware placement decisions have emerged [50]. These methods incorporate power and thermal management considerations into initial VM allocation strategies. Modern placement frameworks leverage predictive analytics and SLA-aware strategies [51]. They make proactive placement decisions that balance energy efficiency with system stability from the initial deployment phase.

2.3 Dynamic Load Balancing and QoS Management

Dynamic load balancing addresses the critical challenge of handling host overload through VM migration while maintaining QoS guarantees. The literature reveals evolution from reactive threshold-based approaches to sophisticated predictive frameworks, though most approaches still rely on deterministic estimations with limited uncertainty handling.

2.3.1 Load Balancing Evolution

Load balancing methods have been proposed to deal with physical machine overload problems using VM migration [52, 53, 54]. Early approaches focused on quantifying virtualized server loads through predetermined resource weights [52]. VM selection strategies based on lowest product of resource utilizations and volume-based approaches using combined load metrics have been developed [54]. Sandpiper automated overload detection using volume defined as the product of CPU, network, and memory loads [55].

However, these methods assume equal or predefined weights for different resources, which may not be correct due to time-varying demands. Dynamic resource

assignment approaches have emerged to address this limitation. Resource usage intensity aware load balancing methods dynamically assign weights based on usage intensities [56]. Game-theoretic approaches for multi-server load balancing with load-dependent server availability consideration have been proposed [57]. These methods provide more sophisticated load distribution strategies compared to traditional approaches.

TABLE 2.2: Dynamic Load Balancing Approaches Comparison

Category	**Relevant Works**	**Detection Method**	**QoS Integration**	**Key Innovation**	**Major Limitation**
Reactive	[52],[53],[54], [55],[58]	Threshold-based	Basic performance metrics	Simple implementation, proven effectiveness	Static weights, delayed response
Adaptive	[56],[57],[59], [60],[61],[62], [63]	Dynamic weighting	Performance degradation aware	Adaptive resource weighting, context awareness	Still reactive, complexity overhead
Predictive	[64],[65],[66], [67],[68],[69]	Forecast-based	Prediction-driven	Proactive intervention, workload anticipation	Prediction errors, model dependency
Probabilistic	[70],[71],[72], [73],[74]	Statistical distributions	Statistical guarantees	Uncertainty quantification	Incomplete framework, limited scope

2.3.2 Load Balancing for Performance Management

SLA-aware load balancing methods with minimum resource wastage objectives have been developed [59]. Performance overhead management approaches for virtual machines have been comprehensively reviewed, covering scenarios from single-server virtualization to multiple geodistributed data centers [60].

Interference-aware approaches gained attention by considering VM performance degradation and inter-VM dependencies. Heterogeneity and interference-aware virtual machine provisioning for predictable performance has been developed [61]. Live

migration techniques that make virtual machines interference-aware in cloud environments have been proposed [62]. Dynamic resource management using virtual machine migrations for improved system performance has been established [75].

2.3.3 Prediction-Based and QoS-Aware Approaches

Prediction-based load balancing schemes address dynamic workload challenges. CloudInsight creates ensemble models for accurate workload predictions [64]. CloudScale for elastic scaling of VM resources according to predicted demands has been proposed [65]. Advanced techniques utilizing Markov stochastic processes for load tendency prediction have been introduced [66].

Fault-tolerant approaches in federated cloud environments have been developed to enhance reliability of cloud services [70]. These methods redistribute VMs to achieve enhanced reliability while reducing transfer costs. Most approaches make decisions based on deterministic estimations of resource demands [71]. Due to dynamic workloads and estimation errors, resource demand estimation may deviate significantly from actual values, resulting in SLA violations. These methods address limitations of deterministic approaches in handling uncertainty and dynamic cloud environments.

2.4 Energy-Efficient VM Consolidation

Dynamic VM consolidation addresses energy efficiency in cloud data centers through intelligent workload redistribution and server consolidation. Research encompasses load detection, VM selection, and placement optimization, though most approaches address components independently rather than providing integrated frameworks.

2.4.1 Foundational Consolidation Approaches

Early energy-aware consolidation established threshold-based techniques for power reduction. Energy-efficient resource allocation strategies utilizing adaptive threshold

mechanisms for maintaining optimal server utilization while ensuring performance guarantees were developed [76, 77]. This was extended through intelligent heuristic frameworks combining multi-objective optimization for dynamic consolidation [78].

TABLE 2.3: Energy-Efficient VM Consolidation Approaches Comparison

Category	**Relevant Works**	**Component Focus**	**Energy Optimization**	**Integration Level**	**Major Limitation**
Detection-focused	[76],[77],[79], [79],[80]	Load imbalance detection	Threshold-based energy awareness	Component-level	Isolated detection, limited coordination
Selection-focused	[81],[82],[83], [84],[85]	VM selection strategies	Migration cost consideration	Component-level	Selection isolation, limited placement coordination
Placement-focused	[86],[87],[88], [89],[90],[91]	Destination optimization	Energy-aware placement	Component-level	Placement isolation, no selection integration
Integrated	[92],[93],[94], [95]	Multiple components	Comprehensive energy optimization	Multi-component	Limited scope, incomplete integration

Static threshold limitations led to adaptive approaches. AFED-EF was developed using adaptive four-threshold frameworks for host classification [79]. Self-adaptive multi-threshold techniques addressed dynamic workload challenges [96, 97]. Temperature metrics alongside utilization measures were incorporated for thermal-aware consolidation [98]. Resource utilization models provided alternative detection mechanisms. Cumulative available-to-total ratio techniques were introduced for precise host state identification [81, 82]. However, these foundational approaches often neglected multi-resource considerations and comprehensive QoS requirements.

2.4.2 Advanced Multi-Objective Consolidation Methods

Modern frameworks integrate energy efficiency with sophisticated QoS optimization. Energy-efficient and quality-aware consolidation balancing power consumption with

performance guarantees through overloading impact and migration overhead considerations was developed [99, 86].

Machine learning revolutionized consolidation strategies. Adaptive deep reinforcement learning automatically learning optimal policies through continuous workload interaction was introduced [92, 93]. Deep learning-based workload prediction frameworks enabling proactive consolidation decisions were developed [100, 101]. Meanwhile, swarm intelligence approaches emerged to address complex optimization challenges. Ant colony systems treating VM placement as constrained combinatorial optimization were proposed [87]. ACO was enhanced for energy and bandwidth minimization [88]. Parallel ant colony optimization (PACO) combining parallelization with SIMD operations was developed [102].

2.5 Research Gaps and Limitations

The comprehensive literature review reveals critical limitations in existing cloud resource management approaches. **Fragmented Optimization** dominates current research. Workload prediction, VM placement, load balancing, and consolidation are addressed as separate problems. This separation leads to poor overall system performance. **Limited Uncertainty Handling** persists in most methods. Researchers use deterministic approaches despite unpredictable workload patterns. This causes systems to adapt poorly to changes. It also results in frequent SLA violations. **Static-Dynamic Coordination Gap** exists between initial setup and runtime operations. There is no smooth transition between these phases. Initial placement decisions become outdated as workloads change.

Simplistic Multi-objective Approaches use basic weighting methods. These methods cannot handle complex trade-offs between energy efficiency, resource use, and service quality. **Limited Integration Scope**characterizes most existing solutions. Current approaches focus on single aspects of resource management. They

fail to consider the interconnected nature of cloud resource challenges. **Scalability Constraints** affect many proposed methods. High computational complexity limits their applicability to large cloud environments. These limitations show a fundamental gap in current research. There is a lack of integrated frameworks that coordinate multiple resource management phases. Handling uncertainty while maintaining practical applicability remains a major challenge across diverse cloud environments.

Chapter 3

Proactive Resource Management through optimal VM placement

3.1 Introduction

Cloud computing has fundamentally transformed organizational resource management. It offers unprecedented scalability and cost-effectiveness [103, 104]. However, this evolution introduces complex resource management challenges. Traditional static allocation approaches cannot adequately address these challenges [105]. The dynamic and heterogeneous nature of cloud workloads demands sophisticated proactive resource management strategies. Energy efficiency requirements and service quality guarantees further complicate these demands.

Contemporary cloud environments face three critical interconnected challenges. Resource allocation inefficiency stems from significant gaps between user-requested and actual consumption patterns [106]. Traditional approaches consistently overestimate requirements by substantial margins. This inefficiency makes proactive resource allocation difficult. It leads to resource fragmentation and increased energy consumption. Energy consumption concerns have become increasingly critical.

Cloud infrastructure accounts for approximately 1-2% of global electricity usage [15]. VM sprawl and inefficient consolidation strategies contribute to unnecessary energy consumption. Reactive placement algorithms fail to optimize energy efficiency during initial deployment. Workload heterogeneity presents significant complexity in developing effective prediction and placement strategies. Different application types exhibit distinct temporal patterns. These range from web applications with diurnal traffic patterns to batch processing jobs with burst computational requirements.

Current VM placement strategies exhibit fundamental limitations. Traditional reactive approaches respond to demands only after manifestation. This often results in suboptimal allocation and frequent migration overhead [107]. Existing proactive methods rely on simplistic models. These fail to capture complex temporal dynamics and uncertainty inherent in cloud workloads. Most approaches treat workload prediction and VM placement as separate problems. This misses synergistic optimization opportunities. Existing approaches typically employ generic models that fail to capture diverse characteristics, resulting in suboptimal placement decisions.

This chapter addresses these challenges through a comprehensive proactive resource management framework. The framework integrates advanced workload prediction with intelligent VM placement strategies. The approach develops specialized prediction models using machine learning-based temporal pattern recognition through clustering [108]. It also employs statistical-stochastic frameworks treating resource consumption as probabilistic phenomena. The VM placement component introduces complementary strategies [109]. These include a heuristic-based approach extending traditional bin-packing algorithms for multi-dimensional constraints. It also features a game-theoretic approach modeling placement as strategic competition among VMs. The integration creates synergistic effects where accurate workload forecasts inform intelligent placement decisions. This enables proactive resource allocation that anticipates future demands rather than merely reacting to

current requirements. The approach significantly reduces resource contention likelihood and minimizes reactive VM migration needs. It achieves superior energy efficiency through optimal consolidation strategies.

3.2 Overview of the Environment

Cloud services operate in data centers composed of numerous Physical Machines (hosts). Users request specific resources (CPU, memory) from Cloud Service Providers (CSPs). CSPs fulfill these requests by creating Virtual Machines (VMs) on physical hardware. The Hypervisor, or Virtual Machine Monitor (VMM), enables multiple VMs to share host resources transparently and in isolation.

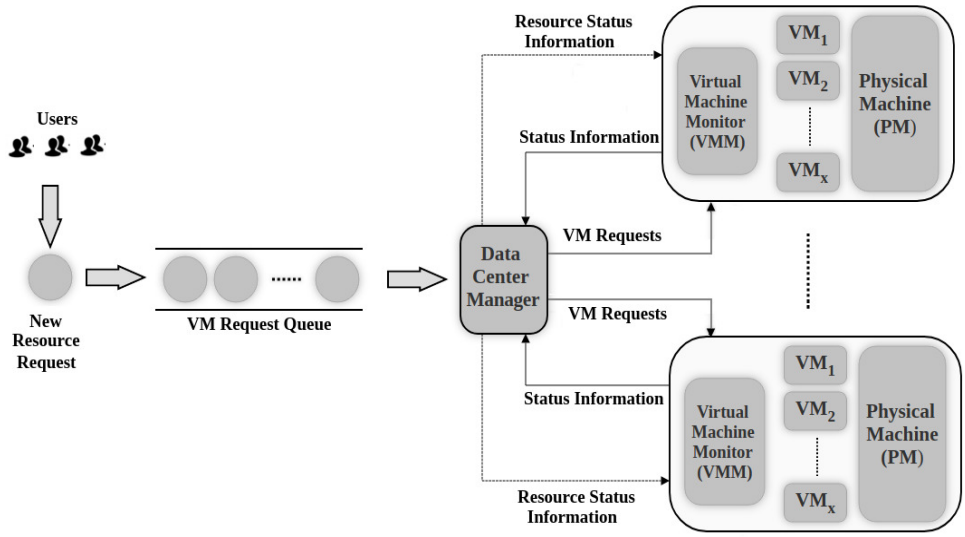

FIGURE 3.1: Flow of VM Requests in a DataCenter.

As illustrated in Figure 3.1, when VM requests arrive, they enter a queue managed by the Data Center Manager (DC Manager). The manager obtains status information from VMMs across all hosts. It then strategically places VMs and responds to users. Traditional placement approaches distribute VM requests with varying resource demands across multiple hosts with different available resources. While VMs are successfully placed, this approach often leaves hosts underutilized.

Analysis reveals a critical inefficiency in current resource management. Applications typically consume significantly less resources than allocated because users overestimate their requirements. The DC Manager operates solely on requested resources rather than actual usage patterns. This leads to activating more hosts than necessary. It increases energy consumption and operational costs unnecessarily.

An improved solution leverages actual resource consumption data instead of user estimates. By understanding real resource usage patterns, the same number of VMs can be accommodated on fewer hosts. This approach eliminates the need to activate unnecessary hosts. The result is reduced overall energy consumption while significantly improving resource utilization efficiency.

3.3 System Models

This section outlines the key components defining virtualized cloud data center behavior and performance: *Cloud Data Center Model*, *Workload Model*, and *Resource Utilization Model*. The Cloud Data Center Model captures the structural and resource configuration of hosts and VMs. The Workload Model represents user-submitted tasks as time-varying resource demands. The Resource Utilization Model quantifies resource consumption patterns, enabling system load and efficiency analysis.

3.3.1 Cloud Data Center Model

In a virtualized cloud environment, there is a pool of server nodes or hosts with various tasks running on them in the form of VMs. Consider a scenario where users submit a set of tasks to the cloud at an instant of time. Each task is presumed to be independent and non-preemptive and operates on a single VM. Executing one task at a time by a VM improves task security and eliminates resource contention among them.

The VMs require resources which are considered to have d ($\mathsf{d} > 1$) dimensions. A set $\mathcal{R}$ represents those resources, $\mathcal{R} = \{r_k \mid k = 1 \text{ to } \mathsf{d}\}$. Assume that a DC has a set $\mathcal{H}$ of M hosts, $\mathcal{H} = \{H_i \mid i = 1 \text{ to } M\}$ and supports a total of P-types of VMs.

Each host H_i is distinguished by a tuple of parameters, with each parameter corresponding to the capacity of one resource. Hence, H_i can be represented as $H_i = \langle C_{ir_k} \mid \forall r_k \in \mathcal{R} \rangle$, where C_{ir_k} is the capacity of resource $r_k \in \mathcal{R}$ in H_i. Likewise, a VM V_j of a specific type can be characterized as a tuple of d attributes, i.e., $V_j = \langle C_{jr_k} \mid \forall r_k \in \mathcal{R} \rangle$, where C_{jr_k} is V_j's capacity for resource r_k. Here, hosts of the same model and VMs of the same type have the same resource configurations.

3.3.2 Workload Model

In a virtualized cloud data center, a workload refers to the aggregate demand generated by independent, non-preemptive tasks submitted by users at a specific time. Each task operates on a single VM, and their execution drives the consumption of computing resources across the infrastructure.

At any time t, the active workload is expressed by the set of running VMs $\mathcal{V}^t$, each demanding varying fractions of their allocated resource capacities based on task requirements. There exists a significant discrepancy between a VM's requested resources and its actual resource demand. Typically, requested resources are substantially higher than actual resource consumption, leading to resource wastage in conventional placement approaches. Throughout this chapter, the term *resource demand* specifically refers to actual resource consumption by VMs.

For a VM V_j, the demand for resource r_k at time t is denoted by $D^t_{jr_k} = f^t_{jr_k}.C_{jr_k}$, where $f^t_{jr_k} \in [0, 1]$ is the fractional demand and C_{jr_k} is the VM's resource capacity. The workload evolves as a time-dependent vector of resource demands across all active VMs in the system.

3.3.3 Resource Utilization Model

Since each VM is constrained to run on a single host, the allocation of VM requests across different hosts can be represented as a mapping between the set $\mathcal{V}^t$ and the set $\mathcal{H}$. Given the number of VMs running on the DC at time t is N^t (i.e., $\mid \mathcal{V}^t \mid = N^t$), a matrix $\mathcal{X}^t = [X_{ij}^t]_{M \times N^t}$ is defined as the VM allocation matrix at that time, where X_{ij}^t is a binary indicator variable, such that:

$$X_{ij}^t = \begin{cases} 1, & \text{if } V_j \text{ is allocated to } H_i \text{ at } t^{th} \text{ time instant} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

At any instance, hosts having VMs running on them are in active state, while others switch to sleep mode to conserve energy. To represent the set of active hosts at time t, $\mathcal{H}_{active}^t \subseteq \mathcal{H}$ is used. Since VM demands vary with time, the resource consumption of hosts also varies. The load of a host at any time instant is defined as the aggregated demand of the VMs operating on it. This is measured for a host H_i by its utilization for all resources. At time instant t, $\mathcal{U}_{ir_k}^t$ is used to denote the utilization of H_i for resource r_k:

$$\mathcal{U}_{ir_k}^t = \frac{\sum_j X_{ij}^t \times D_{jr_k}^t}{C_{ir_k}} \times 100 \tag{3.2}$$

Hence, the DC's utilization for resource r_k at that time (denoted by $DC_{r_k}^t$) can be determined as:

$$DC_{r_k}^t = \frac{\sum_{H_i \in \mathcal{H}_{active}^t} \sum_j X_{ij}^t \times D_{jr_k}^t}{\sum_{H_i \in \mathcal{H}_{active}^t} C_{ir_k}} \times 100 \tag{3.3}$$

This provides a quantitative framework for evaluating both host-level and data center-level resource utilization, serving as a basis for making informed decisions about workload distribution and energy-aware VM placement in cloud environments.

3.3.4 Power and Energy Consumption Model

In cloud data centers, power consumption is a critical concern, directly impacting operational costs and environmental sustainability. Accurately modeling the power and energy consumption of physical hosts is essential for developing energy-efficient resource management strategies.

CPU utilization is widely recognized as a reliable indicator of system load and a key determinant of power consumption. Empirical studies [110] have shown that, with Dynamic Voltage and Frequency Scaling (DVFS) enabled, the relationship between CPU frequency and power consumption is approximately linear. Based on this observation, the power consumption of a host at time t can be modeled using the equation proposed in [16] as:

$$P_i^t = Q.H_{max}^i + (1 - Q) \times H_{max}^i \times \mathcal{U}_{iCPU}^t \tag{3.4}$$

where Q denotes the fraction of energy consumed by idle host, H_{max}^i is the maximum power consumed by the fully utilized host, and $\mathcal{U}_{i.CPU}^t$ specifically denotes the CPU utilization of host H_i at time t.

Since workloads vary over time, CPU utilization $\mathcal{U}_{i.CPU}^t$ is inherently time-dependent. Accordingly, the total energy consumption of H_i over a given period T can be expressed as:

$$E_i = \int_0^T P_i^t dt \tag{3.5}$$

This model serves as a fundamental basis for estimating energy usage in dynamic environments and enables the design of energy-aware VM placement and resource management strategies in cloud data centers.

3.4 Problem Description

This section formulates the multi-objective VM placement optimization problem in virtualized cloud data centers, focusing on achieving energy-efficient resource allocation through intelligent mapping of virtual machines onto physical hosts. The formulation addresses the complexity of balancing dual optimization objectives — minimizing energy consumption while maximizing resource utilization — under dynamic workload conditions. Through this formalization, the section captures the intricate trade-offs inherent in VM placement decisions while defining the constraint boundaries that govern feasible solutions in real-world cloud environments.

3.4.1 Problem Statement

The VM placement problem in a cloud data center involves determining an optimal distribution of a set of VMs onto a pool of physical hosts, based on their resource requirements. Formally, this can be viewed as a mapping from $\mathcal{V}^t$ to $\mathcal{H}$, $f : \mathcal{V}^t \rightarrow \mathcal{H}$ such that $f(V_j) = H_i$. This mapping must ensure that each VM is allocated to exactly one host, while satisfying the capacity constraints of the hosts. Since VMs exhibit dynamic and time-varying workloads, the placement strategy must adapt to fluctuations in resource demands while maintaining efficiency.

The strategy must guarantee feasibility and optimize performance. It aims to minimize energy consumption by reducing active host count while maximizing resource utilization on those hosts. Successfully balancing these objectives improves energy efficiency, reduces operational costs, and enhances resource management effectiveness.

Therefore, considering a time interval T, the multi-objective VM placement can be formulated as an optimization problem shown below:

The objective is:

$$Min \sum_{i=1}^{M} E_i \quad \text{and} \quad Max \sum_{i=1}^{M} \sum_{k=1}^{d} \sum_{t=1}^{T} \mathcal{U}_{ir_k}^t$$

Subject to the following constraints:

$$\sum_{H_i \in \mathcal{H}_{active}^t} C_{ir_k} \geq \sum_{i=1}^{M} \sum_{j=1}^{N^t} X_{ij}^t D_{jr_k}^t \tag{3.6}$$

$$r_k \in \mathcal{R} \sum_{j=1}^{N} X_{ij}^t D_{jr_k}^t \leq C_{ir_k}, \forall H_i \in \mathcal{H} \tag{3.7}$$

$$\sum_{j=1}^{N^t} X_{ij}^t = 1, \forall H_i \in \mathcal{H}_{active}^t \tag{3.8}$$

Constraint (3.6) ensures that the total capacity of active hosts at time t is sufficient to meet the cumulative resource demands of all VMs at that time. Constraint (3.7) guarantees that no host exceeds its capacity for any resource dimension, maintaining local feasibility. Constraint (3.8) enforces that each active host accommodates exactly one VM during the scheduling interval, reflecting a uniform VM distribution strategy. These constraints collectively ensure capacity compliance, workload feasibility, and controlled energy-aware allocation.

3.5 Workload Prediction Techniques

Accurate workload prediction forms the cornerstone of efficient resource management in cloud environments, enabling proactive decision-making for virtual machine placement and resource allocation. This section presents two complementary approaches that demonstrate superior performance in cloud workload forecasting: *Temporal Pattern Recognition through Machine Learning* and *Statistical-Stochastic Forecasting Modeling Framework*.

These approaches were selected for their ability to capture temporal patterns and inherent variability characteristic of cloud workloads. The Temporal Pattern Recognition approach leverages machine learning techniques to identify complex, non-linear relationships in time series data. The Statistical-Stochastic Forecasting Modeling Framework provides a mathematical foundation for quantifying uncertainty through probability distributions and assessing future resource demands. These complementary methodologies address the multifaceted challenges of workload prediction in dynamic cloud environments while emphasizing computational efficiency, prediction accuracy, and adaptability—qualities essential for practical implementation in production cloud infrastructures.

3.5.1 Temporal Pattern Recognition through ML

This section presents a machine learning framework for recognizing and predicting temporal patterns in cloud workload data. The framework addresses the challenge of accurately forecasting resource consumption metrics for VMs operating in data center environments.

VMs exhibit consistent workload patterns that can be identified and leveraged for future resource usage prediction. Since resource consumption data manifests as time-series information, a specialized framework has been developed that is capable of efficiently processing and analyzing the dynamic nature of temporal workload patterns. A time-series is a finite sequence of observations ordered in time, where time is the independent variable. Each VM's resource usage can be represented as a sequence of points $\mathcal{D}^T_{jr_k} = [D^1_{jr_k}, D^2_{jr_k}, \ldots, D^T_{jr_k}]$, where each point $D^t_{jr_k} \in \mathbb{R}$ denotes V_j's resource usage at time instance t. So, there are N^t such time-series at any time t.

A fundamental challenge in workload prediction arises from the heterogeneous

nature of applications running in data centers. Different applications generate distinct resource consumption patterns. Developing a single generic predictive model fails to properly capture this heterogeneity, resulting in reduced prediction accuracy. To address this limitation, the approach first categorizes VMs based on their resource consumption patterns before building specialized predictive models for each category, thereby significantly enhancing accuracy. Therefore, the Temporal Pattern Recognition framework consists of two primary components:

- **Workload Characterization**: This component identifies common workload patterns across multiple VMs and classifies them into several categories based on their temporal characteristics and resource consumption signatures.
- **Workload Prediction**: Using the categorization from the previous step, this component employs specialized machine learning models to estimate the future workload of VMs within each category, leveraging the identified temporal patterns.

The following subsections detail each component of this framework and explain how machine learning techniques are applied to recognize complex temporal patterns in cloud workload data.

3.5.1.1 Workload Characterization through Clustering

The workload characterization component serves as the foundation of the temporal pattern recognition approach, identifying groups of VMs that exhibit similar resource usage patterns over time. By categorizing VMs with homogeneous behavior, this component enables the development of specialized prediction models that accurately capture the temporal dynamics unique to each workload type.

Clustering Methodology: The characterization approach leverages unsupervised clustering techniques, which have demonstrated significant success in pattern

discovery applications. The methodology involves analyzing historical resource usage data to identify natural groupings of VMs with similar temporal characteristics. The objective of clustering in the context of multiple time-series is to discover a partition P consisting of clusters $\mathcal{C} = \{C_i \mid i = 1 \text{ to } k\}$, such that time-series with similar characteristics are grouped together based on appropriate similarity measures. Each C_i represents a distinct cluster, where the complete partition satisfies $P = \bigcup_{i=1}^{k} C_i$ and $C_i \cap C_j = \phi, \forall i \neq j$, ensuring that every VM belongs to exactly one cluster.

For this purpose, the Agglomerative Hierarchical Clustering (AHC) algorithm [111] is employed, which constructs a nested hierarchy of similar objects based on a pairwise distance matrix. This approach offers significant advantages, particularly because it does not require a priori specification of the number of clusters—a parameter that is difficult to determine given the dynamic nature of cloud workloads. The algorithm operates in a bottom-up manner, beginning with individual data points as singleton clusters and progressively merging similar clusters until a single comprehensive cluster is formed. The resulting hierarchical structure allows natural groupings of VMs with similar workload patterns to be identified at various levels of granularity. By analyzing this hierarchy, the optimal number of clusters that best represents the inherent patterns in the data center workload can be determined.

Time-Series Similarity Measurement: The effectiveness of hierarchical clustering depends on selecting an appropriate distance metric for comparing time-series data. Dynamic Time Warping (DTW) [112] is used as the similarity measurement method due to its superior ability to identify shape-based similarities in temporal data. Unlike Euclidean distance, which requires exact alignment of time points, DTW accommodates variations in speed and timing, making it suitable for analyzing cloud workload patterns with similar shapes but different phase alignments.

DTW uses dynamic programming to identify optimal alignment between temporal sequences. Given two time series of equal length $a = \{a_i \in \mathbb{R} \mid i\text{=}1 \text{ to } s\}$ and $b = \{b_i \in \mathbb{R} \mid j\text{=}1 \text{ to } s\}$, the algorithm constructs an $s \times s$ matrix where each element (i, j) represents cumulative distance, formally defined as:

$$e_{ij} = d_{ij} + \min\{e_{(i-1)(j-1)}, e_{(i-1)j}, e_{i(j-1)}\} \tag{3.9}$$

where, $d_{ij} = \mid a_i + b_j \mid$ represents the absolute difference between corresponding points.

Optimal Path Selection and Cluster Formation: The optimal alignment path is determined by selecting the path that minimizes the cumulative distance across the matrix. The DTW distance is formally expressed as:

$$D_{DTW} = \min_{\forall w \in P} \left\{ \sqrt{\sum_{k=1}^{K} d_{w_k}} \right. \tag{3.10}$$

where, P represents the set of all possible warping paths, w_k is the (i, j) coordinate at the k^{th} element of a warping path, and K denotes the path length.

DTW is applied to the set of N time-series, each consisting of l feature vectors representing resource usage patterns. The algorithm generates a symmetric $N \times N$ distance matrix containing DTW values between each pair of VMs. This distance matrix serves as input to the AHC algorithm, which successively merges the closest cluster pairs to form a hierarchical dendrogram.

Ward's method is employed for determining inter-cluster similarity, minimizing the variance of clusters being merged. This method creates compact, evenly-sized clusters that effectively capture similar workload patterns by minimizing the total within-cluster variance. The Ward criterion calculates the increase in the sum of squared distances resulting from merging two clusters, effectively minimizing the

total within-cluster variance.

The clustering process yields distinct VM categories exhibiting similar temporal patterns in resource usage. This categorization enables developing specialized prediction models tailored to each workload pattern, significantly enhancing prediction accuracy compared to generic models and enabling more effective resource management decisions.

It is important to note that AHC is computationally expensive, as it requires computing and storing pairwise distances between all data points. This makes it time-consuming for large-scale datasets. However, in this work, the clustering is performed only once, offline, using historical workload traces. It is not repeated during live system operation. Moreover, the workloads considered in this study are long-running tasks with stable resource consumption patterns. Their behaviour does not change significantly over time. This further reduces the need for re-clustering. For new VMs arriving at runtime, a simple nearest-centroid classification is used to assign them to the closest existing cluster. This avoids running the full AHC pipeline online, keeping the framework efficient and feasible for large-scale deployments.

3.5.1.2 Workload Prediction through Supervised Learning

Following VM categorization into distinct clusters, the `CS-hostS` algorithm develops specialized multi-step forecasting models for each cluster. The algorithm systematically evaluates machine learning techniques combined with temporal feature extraction strategies to achieve optimal prediction accuracy for future resource demands. The prediction approach leverages supervised machine learning techniques with demonstrated success in time-series forecasting. The methodology constructs models that learn temporal dynamics specific to each VM cluster, enabling accurate predictions that account for unique workload characteristics.

`Lines 2-3` establish the optimization framework for each cluster C_i, where Γ_i extracts cluster-specific training and test datasets. The algorithm leverages supervised

CS-hostS: Selection of optimal ML models for each VM cluster.

Input: $\mathcal{C} : C_1, C_2, ..., C_k$, set of VM clusters,
$\mathcal{M} : M_{LR}, M_{k-NN}, M_{DT}, M_{SVM}, M_{GB}$, set of ML models,
$O_w : o_1, o_2, ..., o_w$, set of observation windows,
P_w : Fixed prediction window size.

Output: $\phi = (C_1, M_1^{opt}, o_1^{opt}), ..., (C_k, M_k^{opt}, o_k^{opt})$, optimal model configs.

```
1  φ ← ∅ ;                                                    // Initialize empty set
2  for each C_i ∈ C do
3      Γ_i ← (C_i.T_r, C_i.T_s) ;                             // Extract training and test data
4      for each M_j ∈ M do
5          for each o_l ∈ O_w do
6              D_sup ← Ω(Γ_i.T_r, o_l) ;                      // Transform to supervised format
7              M̂_j ← Ψ(M_j, D_sup) ;                                  // Train model
8              ε_jl ← Λ(M_j, Γ_i.T_s, o_l, P_w) ;                    // Compute error
9              (M_i^opt, o_i^opt) ← argmin_{M_j∈M, o_l∈O_w} ε_jl ;   // Identify optimal
                configuration
10         end
11     end
12     φ ← φ ∪ (C_i, M_i^opt, o_i^opt) ;                          // Add to optimal set
13 end
14 return φ
```

machine learning techniques with demonstrated success in time-series forecasting, constructing models that learn temporal dynamics specific to each VM cluster.

Nested loops (`lines 4-11`) systematically evaluate all model-window combinations. At `line 6`, the algorithm employs *Temporal Feature Extraction through Sliding Windows* via function $\Omega(\cdot)$ to transform VM time-series data into supervised learning format. The *Observation Window (O_w)* represents a fixed-size segment of historical resource usage data. The process begins by selecting the first segment of data points within the observation window, then shifts forward by one time step to select a new segment. This continues iteratively until all resource usage data is segmented. These segments serve as training instances for supervised learning models, where earlier points within each window are used as features to predict subsequent resource usage values. Window size critically influences the model's

ability to capture local trends—excessively small windows fail to capture temporal patterns (underfitting), while overly large windows incorporate irrelevant data or lead to model complexity (overfitting). This necessitates the algorithm's systematic evaluation across multiple O_w values to identify the optimal size for each cluster.

Model training occurs through function $\Psi(\cdot)$ (`line 7`), where each candidate model learns from the transformed supervised dataset. At `line 8`, the algorithm implements a *Multi-Step Forecasting Strategy* through function $\Lambda(\cdot)$, which computes prediction error over the *Prediction Window (P_w)*. This window defines the fixed number of steps ahead for which the model forecasts future resource demands. The multi-step strategy addresses a critical challenge in time-series forecasting: recursive use of predicted values as inputs for subsequent predictions causes error accumulation, degrading accuracy over extended horizons. To mitigate this, the algorithm employs a re-training-based approach where, upon reaching the prediction window boundary, the model retrains using actual resource usage values observed during that period. This periodic re-training strategy prevents the accumulation of prediction errors over time and enables adaptation to evolving workload patterns.

`Line 9` identifies optimal configurations (M_i^{opt}, o_i^{opt}) by minimizing computed prediction errors across all evaluated combinations. `Line 12` accumulates the set of optimal configurations Φ for each cluster. After identifying the optimal model-window pair, the selected models undergo periodic retraining at P_w intervals. The combination of cluster-specific model selection, optimized observation windows through systematic sliding window evaluation, and periodic re-training creates a robust workload prediction framework that accurately captures unique workload characteristics while maintaining prediction accuracy over extended horizons.

3.5.2 Statistical-Stochastic Forecasting Modeling Framework

This section introduces a complementary approach to workload estimation through statistical and stochastic modeling techniques. ML approaches offer powerful prediction capabilities but suffer from critical limitations. These include intensive training data requirements, cold-start problems with new VMs [113], and challenges with bursty workloads. Rare spikes and heavy-tailed demands are frequently under-predicted or ignored. ML models optimized for average error metrics often fail to quantify prediction uncertainty. This creates an impractical tradeoff. Under-prediction causes SLA violations while over-prediction wastes resources.

Statistical-stochastic modeling addresses these constraints by treating resource consumption as an inherently probabilistic phenomenon. This approach functions effectively with minimal historical data. It suits new VM deployments where ML models typically struggle. The framework explicitly models workload variability through probability distributions rather than point estimates. This captures the burstiness characteristic of cloud workloads and provides natural prediction intervals. The statistical framework integrates time-series forecasting with distribution parameter estimation. This enables resource allocation based on quantifiable risk levels.

3.5.2.1 Resource Consumption as a Stochastic Process

The statistical-stochastic forecasting framework re-conceptualizes VM resource consumption from deterministic to probabilistic perspectives. Traditional approaches treat resource usage as predictable values for precise forecasting. This leads to the limitations discussed earlier. The framework recognizes that resource consumption in cloud environments is inherently uncertain and variable.

Resource consumption is modeled as a stochastic process. Each VM's demand for a specific resource at any given time is treated as a random variable drawn

from a probability distribution. Specifically, $D^t_{jr_k}$ is defined as a random variable representing the demand for resource $r_k \in \mathcal{R}$ by virtual machine V_j at time t. Extensive empirical analysis of production cloud environments demonstrates that VM resource consumption patterns predominantly follow normal distributions [56, 114]. This includes comprehensive studies of Google cloud traces and PlanetLab datasets. The empirical evidence, combined with theoretical justification from the Central Limit Theorem [115], supports adopting normal distributions for modeling individual VM resource demands. Resource consumption represents the aggregate of many small, independent processes.

Based on these findings, the resource consumption of each VM is modeled as:

$$D^t_{jr_k} \sim \mathcal{N}(\mu^{r_k}(V_j), \sigma^{r_k}(V_j)^2)$$

where $\mu^{r_k}(V_j)$ represents the expected (mean) resource consumption of VM V_j for resource r_k, and $\sigma^{r_k}(V_j)^2$ represents its variance, capturing the degree of uncertainty and variability in the resource consumption pattern. This stochastic formulation provides several key advantages over deterministic approaches:

- *Natural Uncertainty Quantification*: The model inherently captures and quantifies uncertainty in resource predictions through the variance parameter.
- *Robust to Outliers*: The probabilistic nature provides resilience to occasional spikes or anomalous behavior that significantly impact deterministic models.
- *Risk-Aware Decision Making*: Complete probability distributions enable resource allocation decisions based on acceptable risk levels rather than worst-case scenarios.

The challenge becomes accurately estimating distribution parameters $\mu^{r_k}(V_j)$ and $\sigma^{r_k}(V_j)^2$ for each VM and resource type. This is addressed in the subsequent parameter estimation methodology.

3.5.2.2 Estimation of Distribution Parameters

The core of this framework lies in its probabilistic approach to VM workload forecasting. Rather than producing single-point predictions that fail to capture inherent variability, the model generates complete probability distributions for future resource demands. This section describes the forecasting methodology that enables accurate prediction of both the expected value and the uncertainty in VM resource consumption.

Historical resource consumption data is collected at regular intervals for each VM V_j and resource r_k. This allows continuous refinement of probability distribution parameters based on recent observations. The distribution parameters $\mu^{r_k}(V_j)$ and $\sigma^{r_k}(V_j)$ can be estimated for the next time interval based on previous observations. This approach effectively captures increasing or decreasing trends in resource consumption.

Time-Series Integration: Numerous time-series forecasting models exist for data center resource prediction [116, 117] (e.g., Holt-Winter, ARMA, ARIMA, EWMA). This approach adopts a model-agnostic methodology that integrates time-series techniques without dependence on specific forecasting algorithms. This design enhances framework flexibility and adaptability across diverse operational environments. Let $o_t^{jr_k}$ represent observed consumption of resource r_k by V_j at time t. The forecasting method produces point prediction $\hat{o}_{t+1}^{jr_k}$ for the next interval. The uncertainty is then modeled by analyzing historical prediction errors, which capture volatility patterns and form the basis for variance estimation. By incorporating both the predicted trend and the estimated variance, the framework provides a complete probabilistic characterization of future VM workload.

Given n prior observations $\{o_{t-n+1}^{jr_k}, o_{t-n+2}^{jr_k}, \ldots, o_t^{jr_k}\}$, estimation errors are computed as:

$$\epsilon_{t-n+1}^{jr_k} = o_{t-n+1}^{jr_k} - \hat{o}_{t-n+1}^{jr_k}, \epsilon_{t-n+2}^{jr_k} = o_{t-n+2}^{jr_k} - \hat{o}_{t-n+2}^{jr_k}, ..., \epsilon_t^{jr_k} = o_t^{jr_k} - \hat{o}_t^{jr_k}$$

Sample mean and variance are calculated as:

$$\mu_\epsilon^{jr_k} = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i^{jr_k} \tag{3.11}$$

$$\sigma_\epsilon^{jr_k^2} = \frac{1}{n}\sum_{i=1}^{n} (\epsilon_i^{jr_k} - \mu_\epsilon^{jr_k})^2 \tag{3.12}$$

Parameter Estimation Methodology: Using the error statistics, the distribution parameters for VM resource consumption are estimated as:

$$\mu^{r_k}(V_j) = \hat{o}_{t+1}^{jr_k} + \mu_\epsilon^{jr_k} \tag{3.13}$$

$$\sigma^{r_k}(V_j)^2 = \sigma_\epsilon^{jr_k^2} \tag{3.14}$$

This approach captures evolving trends and inherent uncertainty. Continuous parameter updates accommodate temporal dynamics, facilitating accurate workload characterization.

3.5.2.3 Workload Aggregation for Host-Level Forecasting

The statistical-stochastic framework enables probabilistic resource allocation decisions at the host level. This section demonstrates how individual VM resource consumption distributions aggregate to model total resource demand on each physical host. This aggregation is essential for determining overload probabilities and making informed resource management decisions.

For physical host H_i hosting a set of VMs, aggregate resource consumption represents the sum of individual VM demands. Now, since each VM's resource consumption follows a normal distribution, the aggregate consumption can be characterized using additive properties of normal distributions. For resource r_k on host H_i, total demand or load is the sum of individual VM consumptions, i.e., $L^t_{r_k}(H_i) = \sum_j X^t_{ij} \cdot D^t_{jr_k}$.

VMs typically run isolated and independent workloads with minimal direct interaction in multi-tenant cloud environments. Hypervisor-level resource isolation ensures that one VM's resource usage does not directly influence another's consumption behavior. Given independence assumption, aggregate resource consumption can be effectively modeled using normal distribution properties. The sum of independent normal random variables is also normally distributed. This enables modeling aggregate consumption as:

$$\sum_j X^t_{ij} \cdot D^t_{jr_k} \sim \mathcal{N}\left(\sum_j X^t_{ij} \cdot \mu^{r_k}(V_j), \sum_j X^t_{ij} \cdot \sigma^{r_k}(V_j)^2\right)$$

This aggregated distribution enables computation of overload probabilities—the likelihood that total resource demand exceeds host capacity C_{ir_k}. The probability that host H_i can accommodate aggregate demand for resource r_k without overload is:

$$Pr\left(\sum_j X^t_{ij}.D^t_{jk} \leq C_{ir_k}\right) = \Phi\left(\frac{C_{ir_k} - \sum_j X^t_{ij}.\mu^{r_k}(V_j)}{\sqrt{\sum_j \left[X^t_{ij}.\sigma^{r_k}(V_j)\right]^2}}\right) \tag{3.15}$$

where Φ is the cumulative distribution function of the standard normal distribution. This probabilistic characterization forms the cornerstone for subsequent VM placement and resource management strategies.

3.6 Virtual Machine Placement Techniques

Future resource demands of VMs must be translated into effective placement decisions. These decisions determine which host should accommodate which VMs. The primary challenge lies in balancing multiple competing objectives. The goal is minimizing active hosts to reduce energy consumption while avoiding host overloading that could lead to SLA violations. Host H_i is considered overloaded when any resource $r_k \in \mathcal{R}$ becomes overutilized. This means aggregate demand of all allocated VMs exceeds the predefined utilization threshold U_θ for that resource. Formally, host H_i is overloaded at time t if $\exists r_k \in \mathcal{R} : \mathcal{U}^t_{ir_k} \geq U_\theta$.

This section presents two VM placement strategies addressing these challenges from different perspectives. First, a heuristic approach based on modified Best Fit Decreasing (BFD) algorithm provides computationally efficient placement decisions for large-scale deployments. This approach prioritizes resource utilization efficiency while respecting capacity constraints. Second, a game-theoretic strategy models the placement problem as competitive resource allocation game. It captures complex interactions between VMs competing for limited resources while optimizing global system objectives.

These approaches were selected based on complementary strengths. Heuristic methods offer practical, scalable solutions implementable in production environments with minimal computational overhead. Game-theoretic formulations provide theoretically sound allocations that better capture complex multi-dimensional resource constraints and competing objectives. Together, they represent a comprehensive framework for VM placement adaptable to diverse requirements of modern cloud computing environments.

While both placement strategies are designed to optimize resource utilization

and energy efficiency, it is important to acknowledge that their effectiveness is directly tied to the accuracy of workload predictions. Inaccurate forecasts can cause the heuristic-based approach to allocate insufficient or excessive resources, potentially leading to host overload or unnecessary host activation. Similarly, for the game-theoretic approach, unreliable predictions distort the payoff calculations. To mitigate this, the ML-based prediction framework employs periodic re-training at every prediction window boundary, allowing models to adapt to evolving workload patterns. The stochastic framework counters this through continuous parameter updates using recent observations, keeping distribution estimates aligned with actual resource behavior. Together, these mechanisms reduce the downstream impact of prediction errors on placement quality and energy efficiency. However, some residual sensitivity remains — particularly during sudden workload bursts or abrupt behavioral shifts — which motivates the dynamic load balancing mechanisms discussed in the subsequent chapter.

3.6.1 Heuristic-Based VM Placement Strategy

The heuristic-based VM placement strategy employs a modified BFD algorithm adapted for multi-dimensional resource constraints. Traditional BFD algorithms prove effective for bin-packing problems but typically address only single-dimensional resource allocation [118]. The modified approach extends this concept to handle multiple resource types simultaneously. This makes it suitable for heterogeneous resource constraints in modern cloud DCs.

The placement problem is formalized as assignment of VMs $\mathcal{V}^t = \{V_1, V_2, ..., V_{N^t}\}$ to hosts $\mathcal{H} = \{H_1, H_2, ..., H_M\}$. Each VM has specific CPU and memory requirements. Each host has corresponding resource capacities. The objective is minimizing active hosts while ensuring resource demands do not exceed capacities and maintaining performance guarantees.

The modified BFD algorithm takes VMs with predicted resource profiles and available hosts as input. VMs are sorted in decreasing order using the product of CPU and memory requirements as the sorting metric. This prioritization places VMs with larger resource footprints first, as they are more challenging to accommodate efficiently. This approach minimizes resource fragmentation while maintaining balanced CPU and memory utilization. Considering multiple resource dimensions simultaneously achieves superior consolidation compared to single-dimensional approaches, resulting in fewer active hosts and reduced energy consumption.

3.6.1.1 Modified Best Fit Decreasing Approach

The `MBFD` algorithm presents the formal implementation of Modified BFD approach for VM placement with multi-dimensional resource constraints. This extends traditional bin-packing solutions to efficiently handle CPU and memory resources simultaneously while minimizing active hosts.

The `MBFD` algorithm implements multi-dimensional resource-aware placement through systematic processing. It initializes allocation set $\mathcal{B}$ and sets host utilization to zero (`lines 1-4`). VMs are ranked in descending order of resource demands (`line 5`), measured as the product of CPU and memory requirements. This prioritizes resource-intensive VMs that are more challenging to accommodate.

For each sorted VM, the algorithm iterates through available hosts to find suitable placement (`lines 6-20`). The `Fits` function (`line 8`) verifies VM resource requirements can be satisfied by host capacity across all dimensions. When a compatible host is identified, its power state is checked (`line 9`). Powered-off hosts are then activated (`line 10`).

Before finalizing allocation, `MBFD` calculates projected resource load after VM placement (`line 12`) and verifies that it remains below threshold U_θ (`line 13`). Here, host load is defined as the product of consumed CPU and memory capacities.

MBFD: Modified Best Fit Decreasing VM Placement Algorithm for VMs running at time t

Input: $\mathcal{V}^t$: set of VMs at time t with predicted CPU and memory usage,
$\mathcal{H}$: $H_1, \ldots, H_M$ set of hosts with power state and resource capacities,
U_θ : Resource utilization threshold for hosts.
Output: $\mathcal{B} = (V_i, H_j)$, allocation of VMs to hosts.

```
1  B ← ∅ ;                                          // Initialize empty allocation set
2  for each H_j ∈ H do
3  |  H_j.Utilization ← 0 ;                          // Initialize host utilization
4  end
5  Vsort ← Sort(V, decreasing) ;                    // Sort VMs by resource product
6  for each V_i ∈ Vsort do
7  |  for each H_j ∈ H do
8  |  |  if Fits(V_i, H_j) then
9  |  |  |  if H_j.State = OFF then
10 |  |  |  |  H_j.State ← ON ;                     // Power on the host
11 |  |  |  end
12 |  |  |  Load ← CalculateLoad(H_j, V_i) ;       // Load after allocation
13 |  |  |  if Load < U_θ then
14 |  |  |  |  B ← B ∪ (V_i, H_j) ;                // Add to allocation set
15 |  |  |  |  H_j.Utilization ← UpdateUtilization(H_j, V_i) ;   // Update host
   |  |  |  |   utilization
16 |  |  |  |  break
17 |  |  |  end
18 |  |  end
19 |  end
20 end
21 for each H_j ∈ H do
22 |  if H_j.State = ON ∧ H_j.Utilization = 0 then
23 |  |  H_j.State ← OFF ;                          // Power off unused hosts
24 |  end
25 end
26 return B
```

This threshold mechanism prevents resource contention and potential SLA violations. When conditions are satisfied, VM-host assignment is added to allocation set (`line 14`), host utilization is updated (`line 15`), and the algorithm proceeds to the next VM. After placing all VMs, final optimization identifies and powers off active hosts with zero utilization (`lines 21-24`). This reduces energy consumption

by eliminating unnecessary power usage.

3.6.1.2 Enhancements and Limitations of MBFD

The `MBFD` algorithm effectively handles multi-dimensional resource constraints while implementing energy-efficient placement decisions. By prioritizing resource-intensive VMs and incorporating utilization thresholds, it achieves balance between resource efficiency and performance guarantees.

The algorithm incorporates key modifications to standard BFD for cloud data center operations: (1) *Multi-dimensional resource consideration* - simultaneously evaluates CPU and memory constraints using their product as comprehensive metric, unlike traditional single-dimension BFD; (2) *Power state management* - explicitly manages physical machine power states, activating when needed and deactivating idle machines for energy conservation; (3) *Overload prevention* - threshold-based constraints prevent resource over-utilization by ensuring aggregate host load remains below predefined thresholds, reducing SLA violation risks. The threshold parameter serves as tunable safety margin balancing aggressive consolidation (higher energy efficiency) and conservative resource allocation (lower SLA violation risk). Cloud operators can adjust this threshold to control tradeoffs based on specific requirements and risk tolerance.

Despite effectiveness for multi-dimensional resource-constrained VM placement, several inherent limitations affect performance in complex cloud environments. Heuristic approaches employ static decision-making frameworks with fixed sorting criteria and placement rules, limiting adaptability to dynamic workload characteristics and changing datacenter conditions. These algorithms struggle to effectively balance multiple competing objectives simultaneously, such as energy efficiency, resource utilization uniformity, and performance guarantees. The rigid structure challenges incorporation of multiple objectives with appropriate weighting factors.

Furthermore, heuristic methods lack formal theoretical foundation for optimality

guarantees, making it difficult to quantify solution proximity to theoretical optimum across different datacenter configurations and workload patterns. These limitations suggest need for sophisticated approaches that capture competitive resource allocation nature, model dynamic system interactions, and adapt to changing cloud environment conditions while optimizing collective outcomes.

3.6.2 Game-Theoretic Approach for VM Placement

The game-theoretic VM placement strategy formulates resource allocation as a strategic game where VMs compete for finite physical resources to minimize energy consumption. Unlike heuristic approaches relying on fixed rules, this strategy leverages game theory principles to achieve energy efficiency while maintaining efficient resource utilization. This approach is particularly effective for dynamic cloud environments with fluctuating resource demands, enabling adaptive allocation that optimizes energy efficiency without compromising performance.

Reducing DC energy consumption requires minimizing active hosts. This is achieved by placing VMs to maximize utilization evenly across all resource dimensions without causing host overload. The solution represents competition among VMs toward efficient resource utilization through strategic placement. VM allocation depends not only on individual placement but also on other VM placements. This clearly corresponds to a strategic game where players engage in conflict and collectively decide outcomes. A game involves players, actions, and utility functions calculating payoffs. Each player maximizes payoff to achieve optimal outcomes. For VM placement, hosts with minimal resource wastage represent the preferred outcome.

3.6.2.1 Game-Based VM Placement Formulation

In the Game-Based VM Placement approach, VMs are considered players and their host assignments are possible actions or strategies. The placement problem is modeled as an M-player strategic game $\mathsf{G} = \{\mathcal{P}, \mathcal{A}, \mathcal{J}\}$ where,

- $\mathcal{P} = \mathcal{V}^t$, is a set of N^t players
- $\mathcal{A} = a_1 \times a_2 \times ... \times a_{N^t}$, is the set of action profile
- $\mathcal{J} = \{J_1(b), J_2(b), ..., J_{N^t}(b)\}$, is the set of payoff function

Here, a_j is the action set of player $V_j \in \mathcal{P}$, i.e., $a_j \subseteq \mathcal{H}$. Action profile $b \in \mathcal{A}$ is defined as $b = (b_1, b_2, \ldots, b_{N^t})$, where $b_j \in a_j$ is a specific action of player V_j. Each player chooses actions independently, but payoff depends on both own action and other's actions. For V_j, payoff is determined by function $J_j(b)$.

The optimal placement scheme requires an action profile from which players/VMs cannot improve payoff by changing selected actions/hosts. The solution must be a Nash equilibrium, where no VM can improve its situation by unilaterally changing assigned host. $J_j(b)$ for V_j is defined as:

$$J_j(b) = \frac{\lambda_b^t}{N^t} \tag{3.16}$$

where λ_b^t represents payoff for action profile b at time t. The action profile yielding maximum payoff represents the desired solution. If $b^{max} = (b_1^{max}, b_2^{max}, \ldots, b_{N^t}^{max})$ specifies that profile, then $\forall V_j \in \mathcal{V}^t : J(b_j^{max}, b_{-j}^{max}) \geq J(q_j, q_{-j}^{max})$, where b_{-j}^{max} represents actions of all players from profile b^{max} except player V_j.

To operationalize this game-theoretic formulation and compute the Nash equilibrium that represents optimal VM placement, the Game-Based Virtual Machine Placement (GB-VMP) algorithm is developed. GB-VMP systematically evaluates all possible action profiles to identify the placement configuration that maximizes the collective payoff while preventing resource overload. The following section details the algorithmic implementation of this multi-objective optimization approach.

GB-VMP: Optimal placement of VMs at time t.

Input: $\mathcal{P} : V_1, .., V_{N^t}$, set of VMs to be placed at time t,
$\mathcal{A} : \prod_{j=1}^{M} a_j$,set of all possible action profiles,
$\mathcal{R} : r_1, .., r_{\mathtt{d}}$,set of resource types,
U_θ : resource utilization threshold.

Output: Placement of VMs to hosts.

1 **for** each $b \in \mathcal{A}$ **do**
2 $\quad \mathcal{O}_b^t \leftarrow 1$; `// Initialize overload indicator`
3 $\quad \Upsilon_b^t \leftarrow \sum_{k=1}^{\mathtt{d}} \widetilde{DC_{r_k}^t}$; `// Total normalized utilization across all resources`
4 $\quad \delta_{\mathtt{d}}^t \leftarrow \sum_{k=1}^{\mathtt{d}} \left(\max_{r_k} \widetilde{DC_{r_k}^t} - \widetilde{DC_{r_k}^t} \right)$; `// Utilization gap`
5 $\quad$ **for** each $a_j \in b$ **do**
6 $\quad\quad$ **for** each $r_k \in \mathcal{R}$ **do**
7 $\quad\quad\quad$ **if** $DC_{r_k}^t \geq U_\theta$ **then**
8 $\quad\quad\quad\quad \mathcal{O}_b^t \leftarrow 0$; `// Mark action profile as overloaded`
9 $\quad\quad\quad\quad$ **break**
10 $\quad\quad$ **end**
11 $\quad$ **end**
12 $\quad \lambda_b^t \leftarrow \left(\Upsilon_b^t - \frac{\delta_{\lceil}^t}{\mathtt{d}} \right) \mathcal{O}_b^t$; `// Compute payoff`
13 **end**
14 $b^{opt} \leftarrow \text{argmax}_b \, \lambda_b^t$; `// Select best action profile`
15 **for** each $a_j \in b^{opt}$ **do**
16 $\quad$ **if** a_j *is in Sleep state* **then**
17 $\quad\quad$ Put a_j into Active state ; `// Wake up the host`
18 $\quad$ Assign V_j to a_j.
19 **end**

3.6.2.2 Optimization of Multi-Resource Utilization

Given the set of VMs (players) and their possible host assignments (actions), GB-VMP systematically explores the action space to identify the Nash equilibrium that optimizes resource allocation. The key innovation in the GB-VMP approach is the calculation of the payoff function that balances multiple objectives: maximizing overall resource utilization, achieving uniform utilization across different resource dimensions, and preventing resource overload. For each action profile b, GB-VMP first calculates the payoff λ_b^t (`line 1-13`) as:

$$\lambda_b^t = \left(\Upsilon_b^t - \frac{\delta_{\mathsf{d}}^t}{\mathsf{d}}\right) \mathcal{O}_b^t \tag{3.17}$$

where, Υ_b^t represents the total utilization of the DC at time t for the action profile b and δ_{d}^t is the utilization gap of the DC at that time across d resource dimensions. Therefore, the total normalized utilization Υ_b^t is obtained as $\sum_{k=1}^{\mathsf{d}} \widetilde{DC_{r_k}^t}$ (`line 3`), where $\widetilde{DC_{r_k}^t}$ is the DC's normalized utilization for r_k at time t. Now, δ_{d}^t is computed (`line 4`) for a multi-resource environment as:

$$\delta_{\mathsf{d}}^t = \sum_{k=1}^{\mathsf{d}} \left(\max_{r_k} \widetilde{DC_{r_k}^t} - \widetilde{DC_{r_k}^t}\right) \tag{3.18}$$

Here, δ_{d}^t signifies the uniformity among the utilization of different resources. A smaller value indicates more balanced utilization across all resource dimensions, which is desirable for efficient resource usage and reduced energy consumption. A multiplier $\mathcal{O}_b^t$ is then used in Eq.3.17 in order to associate the overload status of the DC with each of the action profiles. The value of $\mathcal{O}_b^t$ is 1, if the DC is in load-balanced state for b at time t, otherwise it is set to 0. A data center is therefore considered overloaded if any of its constituent hosts becomes overloaded, formally defined as:

$$\mathcal{O}_b^t = \begin{cases} 1, & \text{if } \exists H_i \in \mathcal{H}, \exists r_k \in \mathcal{R} : \mathcal{U}_{ir_k}^t \geq U_\theta \\ 0, & \text{otherwise} \end{cases}$$

This ensures that solutions causing resource overload are automatically eliminated from consideration. GB-VMP initially sets the value of $\mathcal{O}_b^t$ as 1 (`line 2`) and checks for the occurrence of any overload (from `line 5` to `line 11`). The action profile for which λ_b^t is having the highest value (`line 14`) constitutes the equilibrium state. Finally, for each of the players, their corresponding actions or PMs are checked to see if they are in sleep state (`line 15` to `19`), so that they can be put to active state and accomplish the final assignment.

3.7 Performance Evaluation

This section evaluates the proposed workload prediction and VM placement solutions through systematic analysis. The evaluation first assesses machine learning-based prediction accuracy against state-of-the-art approaches. Subsequently, integrated performance is examined by testing various combinations of prediction techniques with placement algorithms to form unified resource management solutions.

The evaluation focuses on CPU and memory resources using publicly available workload traces. Homogeneous datacenter configurations are employed to eliminate hardware heterogeneity variables. Comparative analysis identifies specific conditions where prediction-placement combinations achieve superior performance. The methodology ensures comprehensive evaluation across diverse operational scenarios.

3.7.1 Experimental Setup

A Python 3.0-based simulation framework was developed for evaluation due to the impracticality of large-scale real cloud experiments. The implementation executed on Intel(R) Core(TM) i5-9400f CPU @ 2.90 GHz with 16 GB RAM and 64-bit Windows 10 Operating System. The experimental design ensures reproducibility across diverse operating scenarios. Methodologically sound comparisons with state-of-the-art approaches validate result accuracy. The framework covers comprehensive evaluation scenarios to establish solution effectiveness under varying datacenter conditions.

3.7.1.1 Data Center Configuration

Homogeneous data center configurations maintain experimental integrity and establish controlled evaluation environments. Infrastructure specifications derive from SPECpower benchmark results [119], providing empirical power consumption data across various server configurations.

The experimental setup employs a simulated cluster of 100 homogeneous Dell PowerEdge R6525 servers. Each server features a 64-core processor capable of 2000 MIPS with 128GB RAM. Power consumption characteristics follow SPECpower benchmark data presented in Table 3.2. Server utilization and power consumption relationships follow a linear model from Equation 3.4. The idle state energy fraction (Q) is:

$$Q = \frac{(81.6 \text{ Watt} \times 100)}{404 \text{ Watt}} = 20.2$$

The value 0.202 reflects the idle power consumption of the Dell PowerEdge R6525 server, which is 20.2% of the power consumed when fully utilized. The server power consumption model becomes:

$$P_i^t = 0.202 \cdot H_{max}^i + 0.798 \cdot H_{max}^i \cdot \mathcal{U}_{i.CPU}^t \tag{3.19}$$

This power model enables accurate energy consumption estimation across varying server utilization levels. The model provides a realistic foundation for evaluating energy efficiency across different VM placement strategies.

3.7.1.2 Workload Description

A 350-hour VM resource usage dataset evaluates the approach and tunes framework parameters. The dataset partitions into a 300-hour training set and 50-hour testing set. This partitioning ensures sufficient training data while reserving adequate validation portions.

The evaluation utilizes FastStorage [120], a publicly available workload trace from Bitbrains, a managed hosting and business computation service provider. The trace represents business-critical workloads using identical VMs over several months. Data collection encompasses CPU, memory, network, and disk performance metrics for 1,250 VMs across one operating month at 5-minute intervals. Table 3.1 presents

the metric statistics. These comprehensive workload traces capture real-world resource utilization patterns across diverse application profiles [121]. It represents aggregate descriptive statistics across the full dataset rather than any selected time window. This ensures that the reported values — including mean, minimum, median, maximum, and standard deviation — reflect the complete distribution of resource consumption behavior across diverse workload conditions. The wide gap between requested and actual resource usage is therefore a consistent characteristic observed throughout the entire trace. These traces thus provide a reliable and realistic foundation for evaluating both the prediction accuracy and the resource placement efficiency of the proposed solutions.

TABLE 3.1: Statistics of requested and used resources.

Metrics	Mean	Min	Median	Max	Std Dev
CPU Requested [GHz]	8.9	2.4	5.20	86	11.1
CPU Usage [GHz]	1.4	0.0	0.08	64	4.4
Memory Requested [GB]	10.7	0.0	3.98	511	29.9
Memory Usage [GB]	0.6	0.0	0.10	384	1.8

TABLE 3.2: Power consumption (in Watts) of the server for different CPU utilizations.

Server	0%	20%	40%	60%	80%	100%
Dell PowerEdge R6525	81.6	214	258	305	358	404

3.7.1.3 Performance Metrics

To comprehensively assess the efficacy of the proposed solutions, a diverse set of performance metrics is employed that evaluate both the clustering quality for workload pattern identification and the prediction accuracy of the machine learning models.

Clustering Quality Metrics: For determining the optimal number of clusters and assess the quality of workload pattern categorization, the *Silhouette score* is utilized as an internal validity measure. It evaluates the quality of clustering by

measuring how close an object is to other objects within its own cluster (intra-cluster) as opposed to those in adjacent clusters (inter-cluster). The calculation is based on the full pairwise distance matrix over all data points. For a single data point (a single VM) V_j, its Silhouette value, denoted by $sil(j)$, is calculated as:

$$sil(j) = \frac{a(j) - b(j)}{\max\{a(j), b(j)\}} \tag{3.20}$$

where $a(j)$ is the average DTW distance of V_j to all the other VMs in its own cluster and $b(j)$ is the average DTW distance of V_j to all the VMs in its closest neighboring cluster. The values range from -1 to 1. Values approaching 1 indicate well-clustered data points with significantly smaller intra-cluster than inter-cluster distances. Values approaching -1 suggest poor clustering assignments where data points resemble neighboring clusters more than their own.

Prediction Accuracy Metrics: VM resource usage prediction constitutes a regression problem. Performance evaluation employs two complementary error metrics: *Root-Mean-Square Error (RMSE)* and *Mean Absolute Error (MAE)*.

RMSE quantifies the standard deviation of prediction errors, emphasizing larger errors through quadratic weighting:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T} (D_{jr_k}^t - \hat{D}_{jr_k}^t)^2}{T}} \tag{3.21}$$

MAE measures average error magnitude without directional consideration, providing linear penalty across all error magnitudes:

$$MAE = \frac{1}{T} \sum_{t=1}^{T} |D_{jr_k}^t - \hat{D}_{jr_k}^t| \tag{3.22}$$

In both formulas, T is the total number of predictions evaluated. These metrics enable comprehensive prediction accuracy assessment, with RMSE emphasizing large

prediction error impacts while MAE provides intuitive average error magnitude measures.

3.8 Experimental Results

3.8.1 Findings from Workload Characterization

On the basis of the two primary resource types—CPU and memory—the VMs in the training set were systematically categorized into distinct groups. For each resource type, the AHC algorithm described in Section 3.5.1.1 was applied to identify VMs that exhibit similar temporal resource utilization patterns.

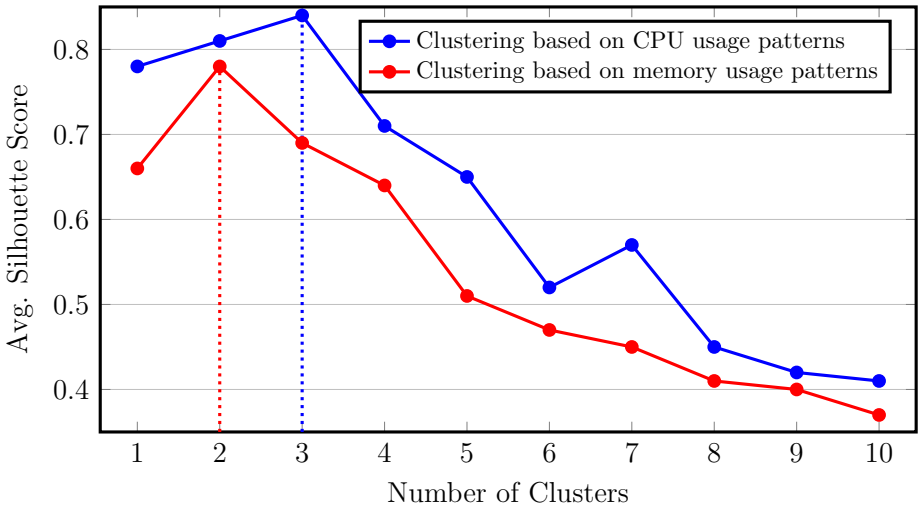

FIGURE 3.2: Silhouette Score for selecting the number of clusters.

Silhouette scores were calculated for varying cluster counts, ranging from $\mathcal{C} = 0$ to $\mathcal{C} = 9$, as illustrated in Figure 3.2. The silhouette score provides quantitative clustering quality measurement. Higher values indicate better-defined clusters with strong intra-cluster similarity and inter-cluster separation. The selection of the optimal number of clusters is critical as it directly impacts the granularity and accuracy of subsequent workload prediction models.

TABLE 3.3: CPU and Memory Utilization Patterns by Cluster

Resource Type	Cluster	VM Count (%)	Min Value	Max Value
CPU (GHz)	1	1,090 (87.2%)	1.53	3.30
	2	83 (6.6%)	0.5	101
	3	77 (6.2%)	13.8	81.33
Memory (GB)	1	1,070 (85.6%)	0.5	7.9
	2	180 (14.4%)	0.177	0.278

CPU Utilization Pattern Analysis: Silhouette score analysis determined three as the optimal cluster number for CPU utilization patterns, corresponding to the highest silhouette value. The 1,250 VMs partitioned into three distinct clusters, each representing characteristic CPU utilization profiles.

Cluster 1 comprises 1,090 VMs (87.2% of total) exhibiting relatively stable and moderate CPU utilization patterns. Average CPU usage ranged from 1.53 to 3.30 GHz over the observation period as detailed in Table 3.3. Cluster 2 consists of 83 VMs (6.6% of total) displaying significantly variable CPU utilization. Hourly average CPU usage fluctuated between 0.5 and 101 GHz. Cluster 3 represents the smallest group with 77 VMs (6.2% of total) demonstrating consistently high CPU utilization with pronounced variability. Hourly average CPU usage ranged from 13.8 to 81.33 GHz.

Memory Utilization Pattern Analysis: AHC algorithm application to memory utilization data yielded two distinct clusters as optimal configuration based on silhouette score analysis. This bimodal distribution reveals two distinct memory utilization patterns among dataset VMs.

Cluster 1 represents the predominant group with 1,070 VMs (85.6% of total). This cluster displays consistent and moderate memory utilization profiles. Table 3.3 illustrates hourly average memory consumption fluctuating between 0.5 and 7.9 GB throughout observation periods. Cluster 2 consists of 180 VMs (14.4% of total) exhibiting notably lower memory utilization characteristics. Hourly average memory

usage ranges from 0.177 to 0.278 GB.

3.8.2 Performance Analysis of Cluster-specific ML Model

After classifying VMs into distinct resource usage clusters, the algorithm from Section 3.5.1.2 was implemented to identify optimal ML models for predicting future resource consumption. The analysis used observation window (O_w) values from 5 to 20 and set the prediction window (P_w) at 5. With data collected at 5-minute intervals, this represents prediction horizons of 25 minutes and observation periods ranging from 25 to 100 minutes. Tables 3.4 and 3.5 report the RMSE and MSE for various ML models on test data across all CPU and memory usage clusters.

ML models for CPU Utilization Prediction: CPU utilization prediction presents challenges due to inherently variable and bursty workloads in cloud environments. Table 3.4 presents CPU utilization prediction metrics across clusters, demonstrating performance differentiation based on workload characteristics.

TABLE 3.4: In each cluster, the RMSE and MAE for estimating CPU consumption using different ML methods.

	ML Methods	**Observation Window Size (O_w)**	**RMSE**	**MAE**
Cluster 1	LR	6	6.15	5.48
	k-NN (k=5)	10	5.34	4.80
	DT	15	5.11	3.66
	SVM	8	5.89	4.23
	GB	10	4.87	3.12
Cluster 2	LR	9	7.42	5.39
	k-NN (k=5)	10	6.55	5.08
	DT	17	4.65	3.43
	SVM	8	4.91	3.87
	GB	14	5.25	4.55
Cluster 3	LR	10	6.56	5.98
	k-NN (k=5)	11	6.22	5.87
	DT	14	5.94	4.51
	SVM	12	5.24	4.48
	GB	10	6.75	5.92

Cluster 1 achieves minimum prediction error with Gradient Boosting (GB) at $O_w = 10$. GB's ensemble approach effectively handles moderate variability with occasional spikes characteristic of these workloads. Cluster 2 workloads exhibit step-like utilization changes best predicted by Decision Tree with $O_w = 17$. This window size provides sufficient historical context for identifying recurring patterns. The model's hierarchical splitting criteria align well with step-like utilization changes. Cluster 3 achieves optimal results with Support Vector Machine at $O_w = 12$. The model's kernel transformations effectively capture complex, non-linear relationships between historical and future utilization patterns present in these workloads.

ML models for Memory Utilization Prediction: The same algorithm set was evaluated across two identified memory usage clusters. Table 3.5 presents distinct optimal configurations for each cluster. Experimental results indicate memory utilization patterns demonstrate higher predictability than CPU utilization. Lower overall error rates across all algorithms reflect this enhanced predictability.

TABLE 3.5: In each cluster, the RMSE and MAE for estimating memory consumption using different ML methods.

	ML Methods	**Observation Window Size (O_w)**	**RMSE**	**MAE**
Cluster 1	LR	12	8.56	7.88
	k-NN (k=5)	15	7.34	6.77
	DT	17	7.04	5.84
	SVM	11	5.53	4.15
	GB	14	6.38	5.52
Cluster 2	LR	7	5.29	4.60
	k-NN (k=5)	10	5.88	4.67
	DT	9	5.71	4.76
	SVM	11	5.04	3.80
	GB	9	4.92	3.56

Cluster 1 exhibits stable resource consumption patterns. Support Vector Machine ($O_w = 11$) delivered superior performance with minimum RMSE of 0.018 and

TABLE 3.6: In each cluster, the RMSE and MAE for estimating both the CPU and memory consumption using LSTM and GRU.

		Methods	Observation Window Size (O_w)	Hidden Layer Size (H_{size})	RMSE	MAE
CPU usage	**Cluster 1**	LSTM	35	128	10.28	8.42
		GRU	34	128	11.06	9.63
	Cluster 2	LSTM	37	64	9.49	7.95
		GRU	33	128	10.63	9.22
	Cluster 3	LSTM	35	128	12.31	10.89
		GRU	34	64	14.72	13.07
Memory usage	**Cluster 1**	LSTM	33	64	11.03	9.66
		GRU	31	64	11.68	10.15
	Cluster 2	LSTM	28	128	9.85	8.36
		GRU	26	128	11.44	9.58

MAE of 0.014. The algorithm effectively captures gradual transitions and consistent memory allocation behavior. Cluster 2 workloads demonstrate variable memory utilization characteristics. Gradient Boosting ($O_w = 9$) achieved most accurate predictions with RMSE of 0.024 and MAE of 0.019, as highlighted in the table. This algorithm's proficiency with non-linear relationships proves particularly effective for dynamic workloads. The diversity in optimal configurations underscores cluster-specific prediction approach importance. No single model-window combination delivers optimal results across all workload patterns.

Evaluation of the Workload Prediction Framework: The proposed CPU and memory prediction strategy was compared against Long Short-Term Memory (LSTM) [30] and Gated Recurrent Unit Encoder-Decoder (GRUED) [31] approaches. Grid search determined optimal parameters for both baselines. Search parameters included observation window sizes, O_w from 5 to 40 and hidden layer sizes (H_L) from $\{8, 16, 32, 64, 128\}$. GRUED encoder and decoder hidden layer sizes remained equal for simplicity. Both methods used 100 iterations with 0.05 learning rate.

Table 3.6 presents optimal O_w and H_L combinations across all clusters. The selected ML models consistently outperformed both methods. In particular, LSTM

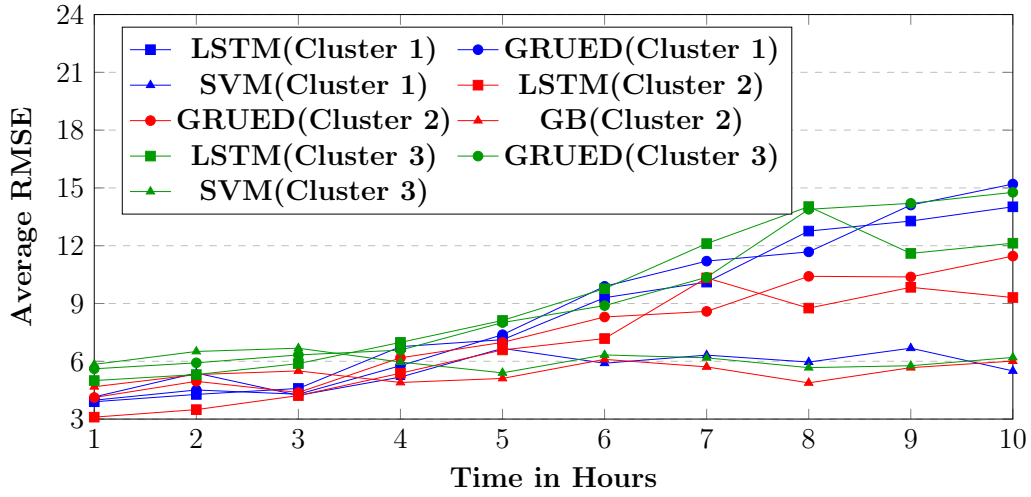

FIGURE 3.3: Hourly avg. RMSE of different methods for estimating CPU consumption in each cluster during the first 10 hours of testing period.

and GRUED required larger observation windows than proposed models, demonstrating superior pattern recognition efficiency.

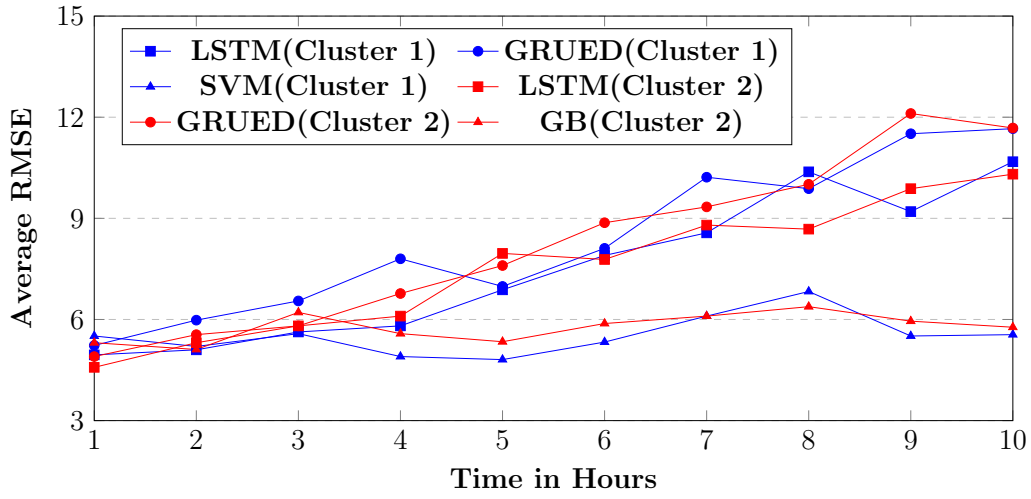

FIGURE 3.4: Hourly avg. RMSE of different methods for estimating memory consumption in each cluster during the first 10 hours of testing period.

Figures 3.3 and 3.4 show hourly average RMSE changes over the first 10 testing hours. LSTM and GRUED achieved initial higher accuracy but degraded over

time due to failure adapting to long-term usage patterns. In contrast, the proposed approach maintains consistent accuracy through model re-training after each prediction window (P_w), adapting to evolving resource usage patterns.

3.9 Comparative Analysis of Integrated Frameworks

This section evaluates integrated performance of proposed components as unified resource management solutions. The assessment examines various combinations of workload prediction techniques with VM placement algorithms in cloud DCs.

The evaluation systematically combines proposed workload prediction methods (ML-based and Stochastic approaches) with two proposed VM placement algorithms (Heuristic-Based MBFD and Game-Based GB-VMP). Comprehensive comparison includes evaluation with existing placement algorithms: BFD [16] and IRB-VMP [122]. All schemes maintain resource utilization thresholds at 90% host capacity to balance utilization and performance stability. The evaluation focuses on resource utilization efficiency (CPU and memory) and energy consumption as critical DC management factors.

3.9.1 Analysis of Resource Utilization Efficiency

This section analyzes resource utilization efficiency across different workload prediction and VM placement combinations. The analysis evaluates CPU utilization, memory utilization, and active host count as key performance metrics. Figure 3.5 and 3.6 display temporal distributions with data aggregated into 6-hour intervals. This representation reveals performance patterns throughout simulation periods. The analysis demonstrates how different approaches adapt to workload fluctuations during peak and off-peak periods under varying datacenter conditions.

CPU Utilization Analysis: Among practical combinations, Stochastic prediction with GB-VMP demonstrated superior performance, maintaining consistently high utilization above 90% throughout the simulation period, ranging from 88.02% to 93.23%. This effectiveness stems from Stochastic prediction's ability to capture workload variability while the game-theoretic approach optimizes placement based on resource competition. Figure 3.5 shows this combination maintaining robust performance even during significant workload fluctuations, particularly evident in the 12-16 hour interval.

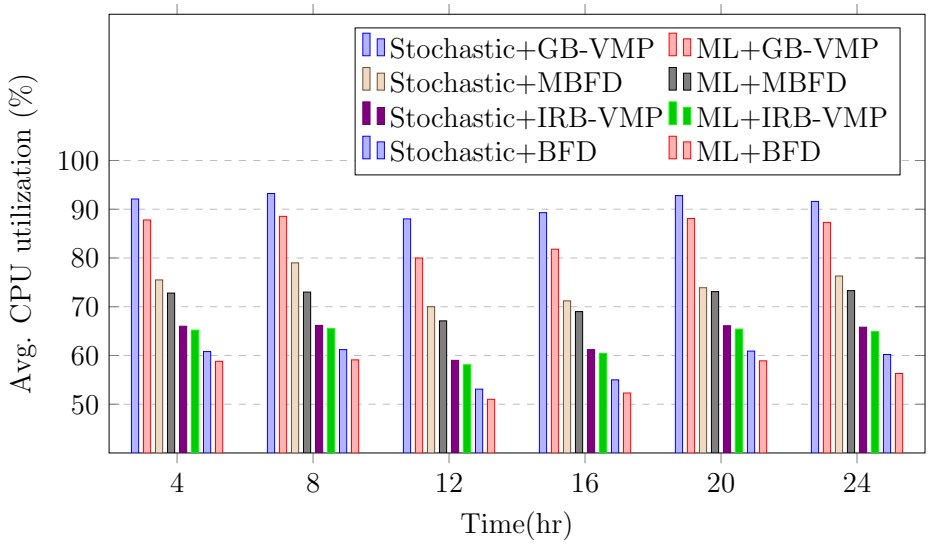

FIGURE 3.5: CPU utilization over the simulation period.

ML-based prediction with GB-VMP followed with competitive performance ranging from 80% to 88.54%. This demonstrates game-theoretic placement algorithm effectiveness regardless of prediction technique used. Stochastic prediction with MBFD achieved moderate performance with fluctuation ranging from 70% to 79%. ML-based prediction with MBFD showed more stable but lower performance ranging from 67% to 73.3%. The combinations with IRB-VMP showed constrained performance. Stochastic+IRB-VMP achieved 59% to 66.17% while ML-based+IRB-VMP reached 58.1% to 65.5%. BFD combinations demonstrated lowest efficiency.

Stochastic+BFD ranged from 53.1% to 61.2% and ML-based+BFD achieved 51% to 59.1%.

Temporal analysis reveals significant fluctuations during the 12-16 hour period across all combinations. Each approach experienced notable drops within respective performance ranges. This period represents the most challenging workload conditions where superior combinations maintain advantages despite increased resource demands. Small utilization gaps exist for combinations using specific VM placement approaches with different prediction techniques. Utilization varies more significantly based on placement approaches rather than prediction approaches. This highlights the critical role of placement algorithms in overall system performance.

Memory Utilization Analysis: Memory utilization analysis reveals a similar performance hierarchy (Figure 3.6). Stochastic+GB-VMP maintained the highest memory utilization (58.1%-61.7%), demonstrating effective memory resource allocation through its game-theoretic optimization approach. ML+GB-VMP achieved competitive performance at 57.3%-59.5%, while the MBFD-based combinations showed moderate efficiency with Stochastic+MBFD at 53.8%-55.9% and ML+MBFD at 52.1%-53.5%. The IRB-VMP combinations exhibited constrained performance, with Stochastic+IRB-VMP reaching 51.2%-52.3% and ML+IRB-VMP at 50.7%-51.8%. BFD-based approaches showed the lowest memory utilization, with Stochastic+BFD at 50.1%-51.0% and ML+BFD at 50.0%-50.8%.

Lower memory utilization compared to CPU utilization indicates memory resources are typically less constrained in evaluated workloads. This suggests CPU-intensive applications dominate the test scenarios. The temporal patterns mirror CPU utilization observations with similar fluctuations during the 12-16 hour period. This reinforces consistent performance characteristics of each combination across different resource types. The consistency validates placement algorithm robustness in managing heterogeneous resource requirements.

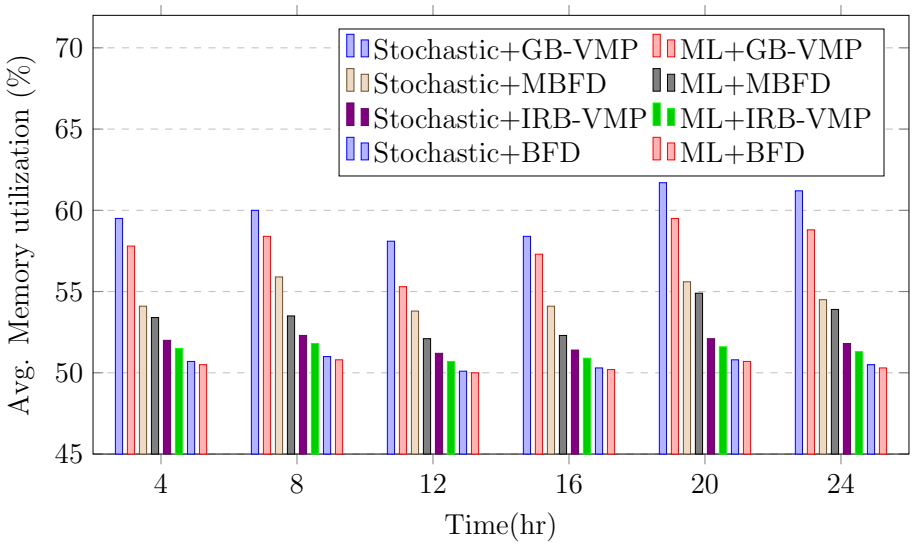

FIGURE 3.6: Memory utilization over the simulation period.

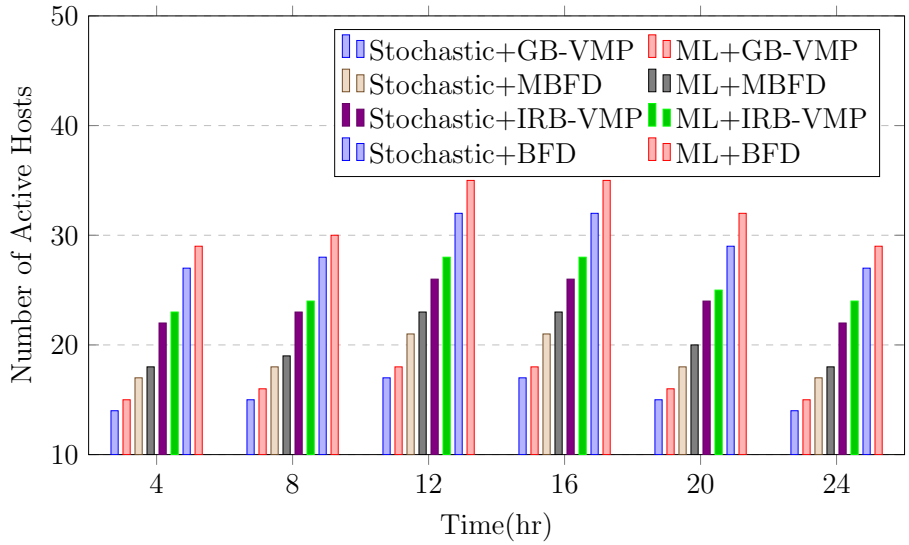

FIGURE 3.7: Number of active hosts over the simulation period.

3.9.2 Analysis of Active Host Counts

Active host counts directly correlate with resource utilization efficiency. Among practical combinations, Stochastic prediction with GB-VMP demonstrated most efficient host utilization, requiring an average of 15.2 active hosts throughout simulation periods. This efficiency stems from the stochastic model's superior workload

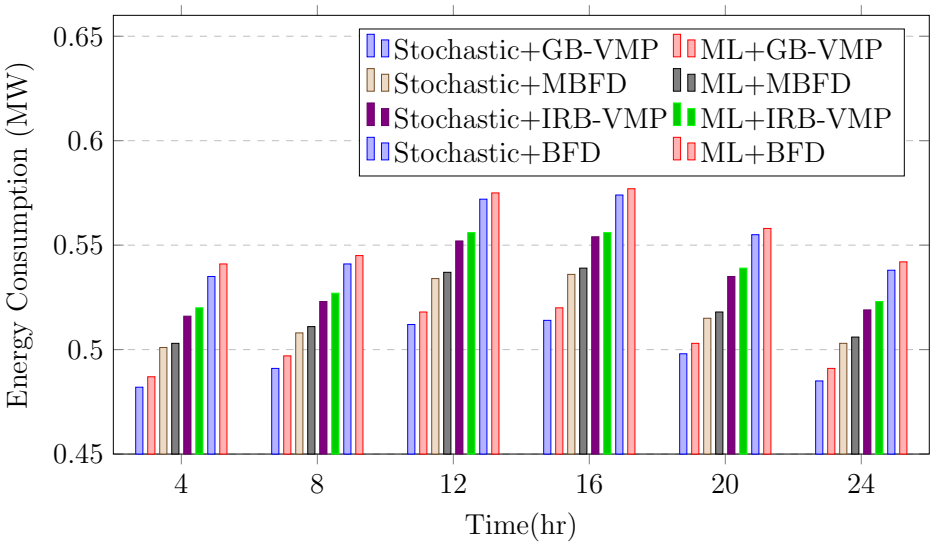

FIGURE 3.8: Energy consumption over the simulation period.

uncertainty handling combined with GB-VMP's game-theoretic optimization. The approach strategically positions VMs to minimize host fragmentation. ML-based prediction with GB-VMP followed closely with 16.4 average active hosts. Stochastic prediction with MBFD required 18.7 hosts while ML-based prediction with MBFD needed 20.1 hosts. IRB-VMP combinations showed higher resource requirements. Stochastic+IRB-VMP required 23.8 hosts and ML+IRB-VMP needed 25.3 active hosts on average. BFD combinations demonstrated lowest host utilization efficiency. Stochastic+BFD required 28.9 hosts and ML+BFD needed 31.4 active hosts on average.

Temporal analysis reveals host requirements increased during 12-16 hour periods across all combinations. This reflects intensified resource demands during peak workload conditions. The behavior demonstrates challenging resource management under elevated workload intensities. During off-peak periods, all combinations achieved more effective host utilization with reduced active host counts. Consistent performance hierarchy across different time intervals validates the robustness of each combination's host utilization characteristics.

3.9.3 Analysis of Energy Consumption Efficiency

Energy efficiency patterns directly correlate with active host counts and closely mirror host utilization results. Stochastic prediction with GB-VMP achieved the lowest average energy consumption at 0.4952 MW. This superior efficiency results from exceptional host consolidation capabilities that directly reduce power consumption across datacenter infrastructure. Figure 3.8 illustrates energy consumption patterns at 6-hour intervals throughout simulation periods. Stochastic+GB-VMP maintains consistently lower energy usage across all time intervals with particularly significant savings during high-workload periods (12-16 hour interval).

ML-based prediction with GB-VMP follows with 0.5018 MW average consumption, demonstrating game-theoretic placement effectiveness in minimizing energy overhead. Stochastic prediction with MBFD consumed 0.5145 MW average while ML-based prediction with MBFD used 0.5237 MW. IRB-VMP combinations showed higher energy usage: Stochastic+IRB-VMP (0.5316 MW) and ML+IRB-VMP (0.5403 MW). BFD combinations demonstrated highest energy consumption with Stochastic+BFD (0.5498 MW) and ML+BFD (0.5612 MW) averages.

3.10 Summary

This chapter presented a comprehensive proactive resource management framework that integrates advanced workload prediction with intelligent VM placement algorithms to address critical cloud resource management challenges. The framework successfully characterized VM workloads into distinct clusters (3 CPU patterns, 2 memory patterns) and developed cluster-specific machine learning models that outperformed state-of-the-art LSTM and GRU approaches. The CS-VMPs algorithm optimally matched prediction models with observation windows, while the

statistical-stochastic framework provided probabilistic resource consumption modeling for risk-aware allocation decisions.

Two complementary strategies were developed: the heuristic-based MBFD algorithm for scalable multi-dimensional resource allocation and the game-theoretic GB-VMP approach for Nash equilibrium solutions. The integrated stochastic prediction with game-theoretic placement achieved optimal performance with only 15.2 active hosts while maintaining $> 90\%$ CPU and $\approx 60\%$ memory utilization. Comprehensive evaluation using real-world traces demonstrated up to 12% energy consumption reduction compared to existing methods, superior resource consolidation efficiency, and consistent performance improvements across diverse workload scenarios. The modular framework design enables flexible deployment with measurable contributions from each component.

Despite these achievements, the proactive VM placement framework addresses only the initial phase of cloud resource management lifecycle. The primary limitation lies in static placement decisions that become suboptimal as workload patterns evolve during VM operation. Real-world cloud applications exhibit dynamic resource consumption patterns with varying user loads, fluctuating computational requirements, and temporal demand variations unpredictable at placement time. Sophisticated prediction models cannot capture the full spectrum of runtime workload changes, leading to resource deficit conditions where allocated resources become insufficient to meet collective VM demands, directly impacting service quality and causing SLA violations.

The framework lacks continuous optimization mechanisms for runtime resource rebalancing when hosts become overloaded or underutilized. Experimental analysis reveals that proactive placement significantly reduces initial resource imbalances, yet inherent cloud workload variability requires continuous management strategies.

The stochastic workload modeling framework and multi-dimensional resource optimization techniques provide essential building blocks for sophisticated load balancing solutions. However, transitioning from static optimization to dynamic resource management requires addressing SLA-aware migration decisions, overload detection mechanisms, and destination host selection strategies. This establishes the critical need for dynamic load balancing mechanisms complementing proactive placement, setting the foundation for comprehensive resource management solutions maintaining optimal performance across the entire cloud operation spectrum.

Chapter 4

QoS-Aware Load Balancing in Dynamic Cloud

4.1 Introduction

Building upon the proactive VM placement framework established in Chapter 3, this chapter addresses the critical challenge of maintaining optimal resource allocation during VM operational lifecycle. While the predictive placement strategies achieved significant improvements in initial energy efficiency and resource utilization, the experimental analysis revealed fundamental limitations in dynamic cloud environments where workload patterns evolve continuously post-deployment [123].

The transition from static optimization to continuous resource management presents complex challenges that extend beyond initial placement decisions. Resource deficit conditions emerge when aggregate VM demands exceed host capacities due to workload evolution, manifesting across multiple dimensions—CPU, memory, and network bandwidth [124]. Contemporary cloud workloads exhibit unpredictable temporal variations including traffic surges, computational bursts, and varying resource demands that create cascading performance effects, leading to service degradation and

potential SLA violations [125].

Existing load balancing solutions can be broadly categorized into reactive and proactive approaches, each with distinct limitations. Reactive methods respond to overload situations only after they manifest, suffering from inherent detection delays that result in prolonged performance degradation. These approaches typically rely on deterministic threshold-based triggers that cannot adequately capture the uncertainty inherent in dynamic workload patterns, leading to either overly conservative resource under-utilization or aggressive strategies that risk frequent SLA violations [126]. The migration decision complexity requires sophisticated analysis of multi-dimensional resource constraints, application-specific QoS requirements[127], and system stability considerations that current reactive approaches handle inadequately.

Proactive approaches attempt to predict future overload conditions but often generate excessive migrations due to prediction inaccuracies, resulting in unnecessary system disruption and performance overhead. More critically, the absence of SLA-aware migration strategies across existing solutions results in decisions that prioritize immediate load relief without considering long-term service quality implications [128].

This chapter introduces a comprehensive SLA-aware load balancing (SLA-LB) framework addressing these limitations through three key innovations:

- **Stochastic Overload Detection**: Leverages the statistical-stochastic forecasting framework from Chapter 3 to implement probabilistic resource consumption modeling that anticipates overload conditions before SLA violations occur, replacing reactive thresholds with proactive risk assessment.

- **Intelligent Migration Optimization**: Addresses VM selection and destination placement through multi-criteria algorithms that minimize migration

overhead while preventing future resource deficits across multiple dimensions, incorporating SLA violation tracking for cumulative service quality impact assessment.

- **Seamless Framework Integration**: Builds upon Chapter 3's stochastic workload modeling, extending probabilistic analysis from initial placement to continuous operational management, maintaining energy efficiency and resource utilization benefits while adapting to evolving conditions.

The significance of this research lies in transforming reactive resource deficit management into proactive load balancing that maintains service quality while optimizing resource utilization. By extending predictive capabilities to continuous operational management, this chapter completes the transition from static optimization to comprehensive adaptive resource management.

4.2 System Models

The system model forms the foundation for the SLA-aware load balancing approach. It formally defines the components of cloud infrastructure, their relationships, and the metrics used to evaluate system performance. The model comprises three key components: *the Resource Utilization Model*, which defines how resources are allocated and consumed; *the Migration Overhead Model*, which quantifies the cost of VM migrations; and *the SLA Violation Model*, which establishes the metrics for measuring service level compliance. Each component provides a mathematical framework that guides load balancing decisions to ensure both efficient resource utilization and SLA compliance.

4.2.1 Resource Utilization Model

The cloud DC is modeled as a set $\mathcal{H}$ of M hosts, $\mathcal{H} = \{H_i \mid i = 1 \text{ to } M\}$, and a set $\mathcal{V}^t$ of N^t VMs, $\mathcal{V}^t = \{V_j \mid j = 1 \text{ to } N^t\}$. The allocation of VMs to hosts is

represented by the VM placement matrix $\mathcal{X}^t = [X_{ij}^t]_{M \times N^t}$, where X_{ij} is a binary indicator variable:

$$X_{ij}^t = \begin{cases} 1, & \text{if } V_j \text{ is allocated to } H_i \text{ at } t^{th} \text{ time instant} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Each host possesses multiple resources represented by the set $\mathcal{R} = \{r_k \mid k = 1 \text{ to } \mathsf{d}\}$, where d is the number of resource types (typically CPU, memory, network bandwidth, and storage). The resource profile of each host H_i is defined by a capacity vector $H_i = \langle C_{ir_k} \mid \forall r_k \in \mathcal{R} \rangle$, where C_{ir_k} quantifies the maximum available capacity of resource r_k on host H_i.

VM resource demands $D_{jr_k}^t$ fluctuate based on application behavior, with temporal variability posing significant challenges for maintaining SLA compliance and efficient resource utilization. To avoid needless migrations triggered by transient spikes, the average demand during time period Δt, denoted by $\overline{D}_{jr_k}^t$, is used:

$$\overline{D}_{jr_k}^t = \frac{1}{\Delta t} \int_{t-\Delta t}^{t} D_{jr_k}^t dt \tag{4.2}$$

where Δt is an adaptive value depending on the desired monitoring granularity.

The utilization of resource r_k in host H_i at time t (denoted by $\mathcal{U}_{ir_k}^t$) is defined as the ratio between the aggregated resource consumption of all allocated VMs and the host capacity:

$$\mathcal{U}_{ir_k}^t = \frac{\sum_j X_{ij}^t \times \overline{D}_{jr_k}^t}{C_{ir_k}} \tag{4.3}$$

This model serves as the foundation for detecting overloaded hosts and triggering VM migration decisions when resource utilization exceeds predefined SLA thresholds. The time-dependent nature enables dynamic adaptation to changing workload

conditions while maintaining service quality guarantees, forming the basis for overload detection detailed in the next section.

4.2.2 Overload Detection Model

The overload detection mechanism is based on a probabilistic approach that leverages stochastic resource usage prediction. As discussed in Section 3.5.2, this approach models resource consumption as a probabilistic process rather than relying on deterministic point predictions. By generating full predictive distributions instead of single values, this method enables quantifiable risk assessment and supports more resilient resource allocation decisions.

Building upon this probabilistic foundation, the proposed load balancing approach periodically checks the resource consumption of each host H_i for each resource type $r_k \in \mathcal{R}$ to determine if it is overloaded. Traditional deterministic approaches consider a host as overloaded if the aggregated resource demand exceeds a predefined threshold. In contrast, this chapter determines an overloaded host by checking whether the probability of overloading exceeds the threshold θ for each resource type, providing a more nuanced view of resource utilization patterns and potential risks.

For a host H_i at time t, the probability of overload for resource type $r_k \in \mathcal{R}$ is defined as:

$$\Pr^t_{\text{over}}(H_i)^{r_k} = 1 - \Pr\left(L^t_{r_k}(H_i) \leq C_{ir_k}\right) \tag{4.4}$$

where $L^t_{r_k}(H_i) = \sum_j X^t_{ij} \cdot \overline{D}^t_{jr_k}$ is the load of H_i for resource r_k by the VMs running on it at time t.

Based on the stochastic modeling framework, the overload probability can be expressed using the cumulative distribution function:

$$\Pr^t_{\text{over}}(H_i)^{r_k} = 1 - \Phi\left(\frac{C_{ir_k} - \mu^t_{r_k}(H_i)}{\sigma^t_{r_k}(H_i)}\right) \tag{4.5}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function, $\mu^t_{r_k}(H_i) = \sum_j X^t_{ij} \cdot \mu^t_{r_k}(V_j)$, and $\sigma^t_{r_k}(H_i) = \sqrt{\sum_j \left[X^t_{ij} \cdot \sigma^t_{r_k}(V_j)\right]^2}$. Here, $\mu^t_{r_k}(V_j)$ and $\sigma^t_{r_k}(V_j)$ are the mean and standard deviation of V_j's resource demand for resource r_k at time t, respectively.

Using this probabilistic framework, a host H_i is considered overloaded at time t if the overload probability for any of its resources exceeds the threshold θ:

$$\exists r_k \in \mathcal{R} : \Pr^t_{\text{over}}(H_i)^{r_k} > \theta \tag{4.6}$$

Resources satisfying this condition are termed *overutilized resources*. To track these dynamic changes, the set of overutilized resources by host H_i at time t is denoted as λ^t_i, while the set of non-overutilized resources is denoted as Υ^t_i, such that $\lambda^t_i \cup \Upsilon^t_i = \mathcal{R}$. This overload detection model enables proactive identification of potential SLA violations before they occur, allowing the system to initiate migration decisions based on risk assessment rather than reactive threshold-based triggers.

4.2.3 Migration Overhead Model

Resource demand of modern applications is dynamic in nature, leading cloud resources to constant fluctuations. In this situation, it is critical to choose appropriate VMs for migration. Moreover, live VM migration consumes extra computing resources and causes service suspension [129], affecting quality of service. Therefore, reducing the total number of VM migrations is essential. The proposed approach selects only those VMs for migration which have a negative impact on the SLA upon resource overloading, as exceeding a host's overload probability threshold does not necessarily indicate migration necessity. The selection method is described in detail in later sections.

The mapping relation between hosts and VMs before migration at time t is denoted by the placement matrix $\mathcal{X}^t = [X^t_{ij}]_{M \times N^t}$, and after migration as $\mathcal{X}^{t^*} =$

$[X_{ij}^{t^*}]_{M \times N^t}$. A VM V_j is considered migrated from source host H_s to destination host H_d if $X_{sj}^t \cdot X_{dj}^{t^*} = 1$ and $s \neq d$. Therefore, the set of migrating VMs at time t, denoted by $\mathcal{S}_{Mig}^t$, is:

$$\mathcal{S}_{Mig}^t = \left\{ V_h \mid \sum_{s=1}^{M} \sum_{d=1}^{M} X_{sh}^t \cdot X_{dh}^{t^*} = 1 \right\} \tag{4.7}$$

The objective is to minimize $|\mathcal{S}_{Mig}^t|$, where $|.|$ represents the cardinality of a set. This minimization ensures that migration overhead is kept to a minimum while maintaining required SLA compliance and resource utilization efficiency. By strategically selecting VMs with the most significant impact on SLA violations, the migration overhead model balances the trade-off between system stability and service quality preservation.

4.3 Problem Description

This section formulates the stochastic load balancing problem in cloud systems, where the primary challenge lies in mitigating resource overloads through strategic VM migrations under uncertain workload conditions. The formulation models the probabilistic behavior of workloads and establishes constraints ensuring system stability. The problem encompasses optimal VM selection for migration, suitable destination host determination, and migration timing decisions to maintain resource balance across the infrastructure.

4.3.1 Problem Statement

In a cloud system, the set of all overloaded hosts at time instant t is represented by $\mathcal{H}_{ovr}^t$ where $\mathcal{H}_{ovr}^t \subset \mathcal{H}$. Given the set of overloaded hosts $\mathcal{H}_{ovr}^t$, the objective is to select a VM V_j from a source host $H_s \in \mathcal{O}$ and identify a destination host $H_d \in \mathcal{H} \setminus H_s$ to migrate V_j in order to reduce the load from H_s.

As per the overload definition, not all resources of a host must be overutilized for it to be classified as overloaded. Some resources may not be overutilized, and their utilization should not be reduced drastically after migrations. Moreover, the migrating VM's final placement plays an important role in efficient resource utilization across the DC. Therefore, migration decisions must simultaneously minimize the number of VM migrations and maximize the utilization of all resources $r_k \in \mathcal{R}$. Let the utilization of resource r_k in host H_i after migration completion be denoted as $\mathcal{U}_{ir_k}^{t^*}$:

$$\mathcal{U}_{ir_k}^{t^*} = \frac{\sum_j X_{ij}^{t^*} \times \overline{D}_{jr_k}^{t}}{C_{ir_k}} \tag{4.8}$$

Due to inherent workload variability, the load balancing approach performs VM migrations such that the total resource requirements of VMs for each resource type r_k on each host H_i at any time t does not exceed the capacity C_{ir_k} with high probability $1-\theta$:

$$\Pr\left(L_{r_k}^{t^*}(H_i) \leq C_{ir_k}\right) > 1-\theta \tag{4.9}$$

where $L_{r_k}^{t^*}(H_i) = \sum_j X_{ij}^{t^*} \cdot \overline{D}_{jr_k}^{t}$ is the load of H_i for resource r_k at time t after migration.

The multi-objective load balancing problem can therefore be formulated as the following optimization problem: Objective:

$$\text{Minimize} \quad |\mathcal{S}_{Mig}^{t}| \qquad \text{and} \tag{4.10}$$

$$\text{Maximize} \quad \sum_i \mathcal{U}_{ir_k}^{t*}, \quad (\forall r_k \in \mathcal{R}, \forall H_i \in \mathcal{H}) \tag{4.11}$$

Subject to the following constraints:

$$\Pr\left(L_{r_k}^{t^*} \leq C_{ir_k}\right) > 1-\theta, \quad (\forall r_k \in \mathcal{R}, \forall H_i \in \mathcal{H}) \tag{4.12}$$

$$X_{ij}^{t*} \in 0, 1, \quad (\forall i, j) \tag{4.13}$$

The VM migration problem for load balancing is NP-hard, as it can be regarded as a variant of the stochastic knapsack problem. The computational complexity is further amplified by the large number of VMs and hosts in modern DCs. Due to this complexity, practical solutions typically rely on heuristic approaches. Therefore, this work develops a heuristic-based framework to solve the stochastic load balancing problem while maintaining computational efficiency and near-optimal performance.

4.4 SLA-Aware Load Balancing Framework

This section presents the practical framework for implementing the SLA-aware load balancing (**SLA-LB**) approach. The framework addresses three fundamental questions in VM migration-based load balancing: *when* to migrate VMs, *which* VMs to migrate, and *where* to migrate them. The proposed solution integrates probabilistic overload detection with intelligent VM selection and placement strategies to achieve efficient resource utilization while maintaining SLA compliance.

The SLA-LB framework operates on the principle that VM migration should be triggered only when a host becomes overloaded according to the probabilistic overload detection model defined in Section 4.2.2. The primary objective is to ensure that each host maintains its aggregate VM resource requirements within capacity limits for all resource types, while achieving rapid convergence toward a load-balanced state and efficient resource utilization.

4.4.1 Architectural Components and Information Flow

The SLA-LB framework adopts a centralized approach where the load balancer operates from a central server, enabling global optimization decisions across the DC infrastructure. Figure 4.1 illustrates the comprehensive architecture and information flow among interconnected modules.

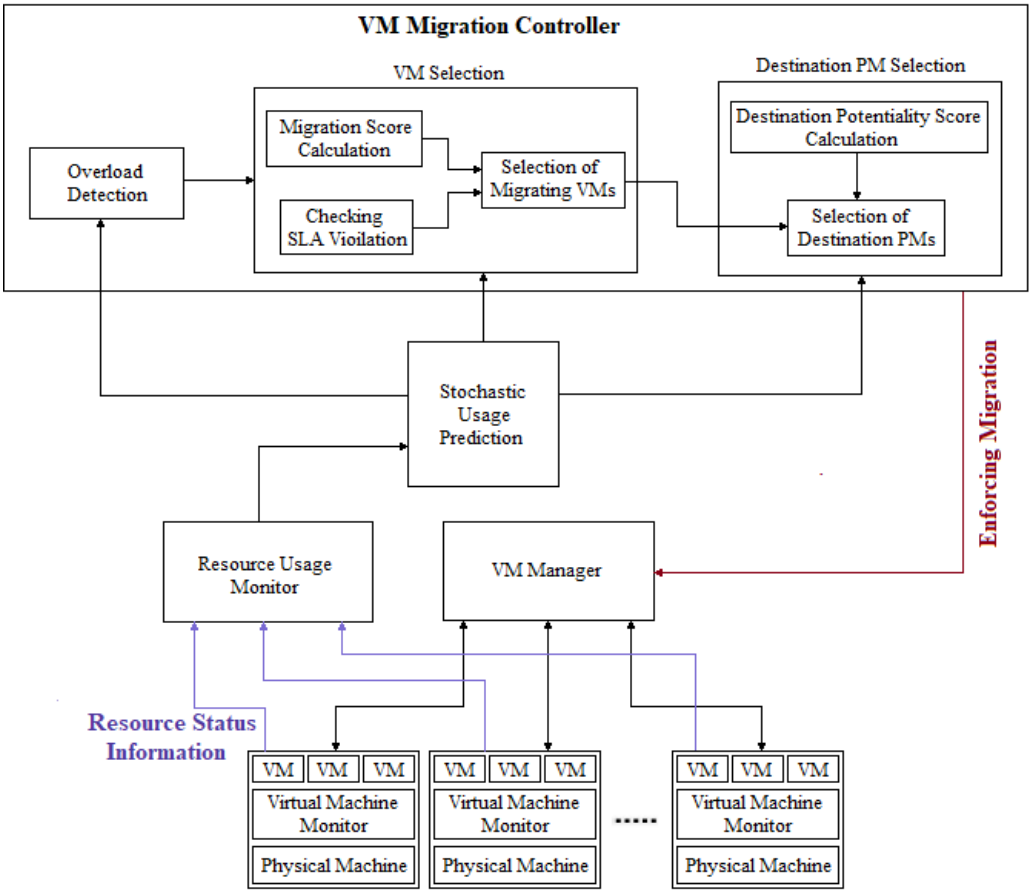

FIGURE 4.1: Framework of the SLA-Aware Load Balancing Approach

The framework comprises several specialized modules that work in coordination to achieve optimal load balancing:

1. *VM Manager*: This module serves as the interface between the load balancing framework and the virtualization infrastructure. It manages virtual environments and facilitates live VM migrations with minimal service interruption. The VM Manager executes migration commands from the VM Migration Controller while ensuring service quality during transitions.

2. *Resource Monitor*: Operating as the data collection component, the Resource Monitor continuously tracks resource usage statistics for each active host and

VM. It periodically samples resource consumption metrics including CPU utilization, memory usage, network bandwidth consumption, and storage I/O patterns. These statistics are systematically collected and forwarded to the Prediction Module for analysis.

3. *Prediction Module*: This critical component processes historical resource usage statistics from the Resource Monitor to estimate probabilistic distribution parameters of resource consumption for each VM. Leveraging the statistical-stochastic forecasting framework from Section 3.5.2, this module generates predictive distributions that capture uncertainty in workload patterns. The framework enables modeling resource consumption as a probabilistic process rather than deterministic point predictions, providing full predictive distributions. The estimated parameters include mean resource consumption rates and their associated variances, essential for probabilistic overload detection.

4. *Overload Detection Engine*: Building upon predictions from the Prediction Module, this engine implements the probabilistic overload detection mechanism described in Section 4.2.2. It continuously evaluates the overload probability for each resource type on every host using Eq. 4.6 and identifies hosts requiring load redistribution based on the threshold criterion.

5. *VM Selection and Placement Optimizer*: When overloaded hosts are detected, this module addresses the dual challenge of selecting appropriate VMs for migration and identifying suitable destination hosts. The selection process considers each VM's impact on the overload condition. The placement strategy ensures migrations contribute to overall system balance and resource efficiency.

6. *VM Migration Controller*: Serving as the coordination hub, this controller receives optimization decisions from the VM Selection and Placement Optimizer

and translates them into actionable migration commands. It orchestrates the migration process by instructing the VM Manager to execute specific VM relocations. It maintains awareness of system-wide migration activities to prevent conflicts.

The information flow follows a continuous cycle: resource usage data flows from the *Monitor* to the *Prediction Module*, generating probabilistic estimates for the *Overload Detection Engine*. Upon identifying overload conditions, the *VM Selection and Placement Optimizer* determines migration strategies executed by the *VM Migration Controller* and *VM Manager*. This cyclical process ensures continuous adaptation while maintaining system stability and SLA compliance. The centralized architecture enables global optimization decisions considering the entire DC state.

Having established the framework architecture, this chapter now addresses the most critical component: the VM Selection and Placement Optimizer. This component tackles the challenging aspects of determining which VMs to migrate and where to place them for optimal resource distribution while maintaining SLA compliance.

4.5 Selection of Migrating VMs to Relieve Load

The VM selection module determines which VMs should migrate from overloaded hosts to achieve effective load redistribution while avoiding unnecessary migrations from transient fluctuations. This module comprises three interconnected components working sequentially to identify optimal migration candidates and minimize system disruptions.

The selection process initiates only after confirming persistent overload conditions, distinguishing them from temporary spikes. Upon confirmation by the Overload Detection Engine, all VMs on the overloaded host are evaluated for migration

suitability. The methodology applies SLA-aware filtering to reduce unnecessary migrations, then implements a comprehensive ranking algorithm considering multiple criteria and temporal stability of resource demands.

4.5.1 Resource-Intensity-Aware VM Selection

The VM selection process identifies VMs whose migration provides rapid load relief while maintaining optimal utilization of non-overloaded resources. Consider host H_i predicted as overutilized for resource set λ_i^t at time t. To achieve rapid relief, the process prioritizes VMs with higher consumption of resources in λ_i^t while selecting VMs with low consumption of resources in non-overutilized set Υ_i^t to preserve capacity utilization.

This challenge formulates as a Multi-Criteria Decision Making (MCDM) problem that evaluates multiple alternatives based on various decision criteria. In this formulation, the VMs hosted by the overloaded host serve as the alternatives, while the different resource types constitute the decision criteria. The framework employs linear scalarization [130], assigning non-negative weights to each criterion and maximizing their linear combination across alternatives. The Weighted Sum Model (WSM) serves as the foundation for this approach. For each criterion, WSM calculates the weighted sum of all alternative's performance values, identifying the maximum value as the optimal choice.

4.5.1.1 Performance Value Calculation

For host H_i at time t, let $\mathcal{V}_i^t = \{V_j \mid X_{ij}^t = 1\}$ represent the VM set allocated to H_i. For resource $r_k \in \mathcal{R}$, the performance value of VM $V_x \in \mathcal{V}_i^t$ is the probability that H_i will not exceed capacity C_{ir_k} upon removing V_x: $Pr\left(\sum_{V_j \in \mathcal{V}_i^t \setminus \{V_x\}} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)$. This performance value, $PVal_{ir_k}^t(V_x)$, indicates the VM's consumption impact for that resource type. The optimal migration candidate exhibits high performance values for overutilized resources in $\lambda_i(t)$ and low values for non-overutilized resources in Υ_i^t.

To prioritize appropriately, the performance value function differs for each resource category:

$$PVal^t_{ir_k}(V_x) = \begin{cases} Pr\left(\sum_{V_j \in \mathcal{V}^t_i \setminus \{V_x\}} \overline{D}^t_{jr_k} \leq C_{ir_k}\right), & \text{for } k \in \lambda^t_i \\ 1 - Pr\left(\sum_{V_j \in \mathcal{V}^t_i \setminus \{V_x\}} \overline{D}^t_{jr_k} \leq C_{ir_k}\right), & \text{for } k \in \Upsilon^t_i \end{cases}$$

For overutilized resources ($r_k \in \lambda^t_i$), higher $PVal^t_{ir_k}(V_x)$ indicates removing V_x enables rapid load relief. For non-overutilized resources ($r_k \in \Upsilon^t_i$), higher values ensure minimal utilization reduction, preserving efficiency.

Migration Score Calculation: Algorithm `MigScore(`H_i, t`)` implements a Weighted Sum Model (WSM) methodology to calculate migration scores for all VMs hosted on overloaded host H_i. The algorithm quantifies each VM's suitability for migration through a two-level iteration structure that evaluates weighted performance across multiple resource dimensions.

The Migration Score (MS) of VM V_x as a migration candidate from H_i is calculated as:

$$MS^t_{ix} = \sum_{r_k \in \mathcal{R}} \omega^t_{ir_k} \cdot PVal^t_{ir_k}(V_x) \tag{4.14}$$

where $\omega^t_{ir_k}$ represents the relative importance of resource type r_k in host H_i at time t. The weight assignment strategy prioritizes rapid load relief over resource waste reduction. Overutilized resources receive higher weights than non-overutilized ones. Within overutilized resources, higher consumption levels get greater weights. Among non-overutilized resources, lower consumption receives higher weights to minimize wastage. The weight for resource r_k is defined as:

$$\omega^t_{ir_k} = \begin{cases} \frac{1}{1-\mathrm{Pr}^t_{\mathrm{over}}(H_i)^{r_k}}, & \text{for } r_k \in \lambda^t_i \\ 1 - \mathrm{Pr}^t_{\mathrm{over}}(H_i)^{r_k}, & \text{for } r_k \in \Upsilon^t_i \end{cases}$$

MigScore(H_i, t): Migration score calculation for VMs on host H_i at time t.

Input: $\mathcal{V}_i^t$: $\{V_j \mid X_{ij}^t = 1\}$, set of VMs allocated to host H_i,
λ_i^t: set of overutilized resources on host H_i at time t,
Υ_i^t: set of non-overutilized resources on host H_i at time t,
C_{H_i}: $\{C_{ir_k} \mid \forall r_k \in \lambda_i^t \cup \Upsilon_i^t\}$, capacity vector.
Output: $MS_{H_i}^t$: $\{MS_{ix}^t \mid \forall V_x \in \mathcal{V}_i^t\}$, set of migration scores.

$MS_{H_i}^t \leftarrow \emptyset$ `// Initialize migration score set`
for each $V_x \in \mathcal{V}_i^t$ **do**
  $WgtSum \leftarrow 0$ `// Initialize weighted sum accumulator`
  for each $r_k \in \lambda_i^t \cup \Upsilon_i^t$ **do**
    $PVal_{ir_k}^t(V_x) \leftarrow Pr\left(\sum_{V_j \in \mathcal{V}_i^t \setminus \{V_x\}} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)$ `// Performance value`
    if $r_k \in \lambda_i^t$ **then**
      $\omega_{ir_k}^t \leftarrow \left[Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)\right]^{-1}$ `// Weight assignment for overutilized resources`
      $WgtPVal_x^{r_k} \leftarrow \omega_{ir_k}^t \times PVal_{ir_k}^t(V_x)$ `// Weighted performance`
    else
      $\omega_{ir_k}^t \leftarrow Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)$ `// Weight assignment for non-overutilized resources`
      $WgtPVal_x^{r_k} \leftarrow \omega_{ir_k}^t \times (1 - PVal_{ir_k}^t(V_x))$ `// Complement for efficiency`
    end
    $WgtSum \leftarrow WgtSum + WgtPVal_x^{r_k}$ `// Accumulate weighted values`
  end
  $MS_{ix}^t \leftarrow WgtSum$ `// Assign migration score`
  Add MS_{ix}^t to $MS_{H_i}^t$ `// Store in result set`
end
return $MS_{H_i}^t$

This formulation ensures that higher overload probabilities yield higher weights for overutilized resources and lower weights for non-overutilized resources, guaranteeing $\omega_{ir_k}^t > 1$ for $r_k \in \lambda_i^t$ always exceeds $\omega_{ir_k}^t < 1$ for $r_k \in \Upsilon_i^t$. Therefore, the weight equation becomes:

$$\omega_{ir_k}^t = \begin{cases} \left[\Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)\right]^{-1}, & \text{for } r_k \in \lambda_i^t \\ \Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right), & \text{for } r_k \in \Upsilon_i^t \end{cases} \tag{4.15}$$

The outer loop processes each VM $V_x \in \mathcal{V}_i^t$ hosted on target server H_i, while the inner loop evaluates weighted performance across all resource types in $\lambda_i^t \cup \Upsilon_i^t$. For each VM-resource pair, the algorithm computes performance value $PVal_{ir_k}^t(V_x)$ as the probability that H_i will not exceed capacity C_{ir_k} upon VM removal (`line 5`).

Weight assignment differentiates between resource categories: overutilized resources receive inverse probability weights (`line 8`), while non-overutilized resources receive direct probability weights (`line 11`). Weighted performance values are calculated using the original performance value for overutilized resources (`line 9`) and its complement for non-overutilized resources (`line 12`), prioritizing minimal resource wastage. The migration score is obtained by accumulating weighted performance values across resource dimensions (`line 14`), with the complete set returned for VM selection (`lines 16-17`). This ensures VMs with higher consumption of overutilized resources and lower consumption of efficiently utilized resources receive higher migration scores.

4.5.2 Reduction of Unnecessary Migrations

Modern cloud applications exhibit dynamic workload patterns with frequent transient resource demand spikes—sharp, temporary increases followed by returns to baseline levels. During such spikes, hosts may temporarily become overloaded then return to normal or underloaded states once spikes subside. This temporal variability creates complex decision-making for migration mechanisms. While migration scores identify suitable VM candidates, the challenge is determining when migration is genuinely necessary versus an overreaction to temporary conditions. Triggering migration solely when instantaneous overload probability exceeds the threshold (i.e., $\Pr_{\text{over}}^t(H_i)^{r_k} > \theta$ for $r_k \in \lambda_i^t$) may cause unnecessary overhead and system disruption.

The fundamental challenge is temporal: at time t, condition $\Pr_{\text{over}}^t(H_i)^{r_k} > \theta$ for resource r_k provides no assurance this probability will persist. The framework

requires a mechanism to distinguish transient spikes that resolve naturally from sustained overloads necessitating migration.

To address this, SLA-LB incorporates SLA-aware VM selection that triggers migration only when VMs experience Service Level Objective (SLO) violations. An SLO, defined within the SLA, represents measurable QoS characteristics acceptable to both providers and customers. Here, an SLO requires at least ξ percent (where $\xi \in [0, 100]$) of a VM's resource requirements be satisfied throughout its operational lifetime.

4.5.2.1 Cumulative SLO Violation Tracking

The framework monitors resource allocation at regular intervals of δt time units, where $\delta t = t_{c+1} - t_c$ represents the monitoring interval between consecutive time points t_c and t_{c+1}. For a VM V_x with start time t_s^x, the cumulative amount of r_k that could not be allocated during overloading from t_s^x to t_c is denoted as $Cu_{r_k}^x(t_c)$. This metric quantifies the accumulated resource deficit:

$$Cu_{r_k}^x(t_c) = \sum_{t=t_s^x}^{t_c} \gamma_{r_k}^x(t), \quad r_k \in \mathcal{R} \tag{4.16}$$

where $\gamma_{r_k}^x(t)$ represents the percentage of unallocated resource r_k for V_x at time t.

SLO Violation Assessment at Current Time t_c: Consider a VM $V_x \in \mathcal{V}_i^{t_c}$, where $\mathcal{V}_i^{t_c}$ denotes the set of VMs hosted by H_i at time t_c. The value of $\gamma_{r_k}^x(t_c)$ in Eq. 4.16 equals zero when H_i operates within capacity limits. When H_i experiences overload for resource type $r_k \in \lambda_i^{t_c}$, computing $\gamma_{r_k}^x(t_c)$ requires determining the total resource deficit.

The aggregate overloaded resource amount of type r_k in host H_i at time t_c, denoted as $ORA_{r_k}^{t_c}(H_i)$, represents the total demand exceeding the host's capacity:

$$ORA_{r_k}^{t_c}(H_i) = \sum_{V_j \in \mathcal{V}_i^{t_c}} \overline{D}_{jr_k}^{t_c} - C_{ir_k} \tag{4.17}$$

During overload, all VMs in $\mathcal{V}_i^{t_c}$ experience resource scarcity for type r_k. For any VM $V_j \in \mathcal{V}_i^{t_c}$, the unallocated resource amount equals $\omega_{jr_k}^{t_c} \times ORA_{r_k}^{t_c}(H_i)$, where $\omega_{jr_k}^{t_c} \in (0, 1)$ represents the fraction of total overloaded resource that cannot be allocated to V_j. The constraint $\sum_{V_j \in \mathcal{V}_i^{t_c}} \omega_{jr_k}^{t_c} = 1$ ensures the entire overloaded amount is distributed among affected VMs. The fraction $\omega_{xr_k}^{t_c}$ for VM V_x is determined proportionally based on its demand relative to total demand:

$$\omega_{xr_k}^{t_c} = \frac{\overline{D}_{xr_k}^{t_c}}{\sum_{V_j \in \mathcal{V}_i^{t_c}} \overline{D}_{jr_k}^{t_c}} \tag{4.18}$$

Consequently, the percentage of unallocated resource r_k for VM V_x at time t_c is:

$$\gamma_{r_k}^{x}(t_c) = \frac{\omega_{xr_k}^{t_c} \times ORA_{r_k}^{t_c}(H_i)}{\overline{D}_{xr_k}^{t_c}} \times 100 \tag{4.19}$$

SLO Violation Prediction at Future Time t_{c+1}: The direct application of Equation 4.19 for future time t_{c+1} is not feasible due to the unavailability of individual VM resource consumption values. Instead, the framework employs the stochastic prediction module to estimate aggregate resource consumption patterns. The individual VM demand $\overline{D}_{xr_k}^{t_{c+1}}$ is derived through the difference between total aggregate consumption and consumption excluding VM V_x:

$$\overline{D}_{xr_k}^{t_{c+1}} = \sum_{V_j \in \mathcal{V}_i^{t_{c+1}}} \overline{D}_{jr_k}^{t_{c+1}} - \sum_{V_j \in \mathcal{V}_i^{t_{c+1}} \setminus V_x} \overline{D}_{jr_k}^{t_{c+1}} \tag{4.20}$$

The fraction of overloaded resource allocated to VM V_x at time t_{c+1} is then computed as:

$$\omega_{xr_k}^{t_{c+1}} = \frac{\overline{D}_{xr_k}^{t_{c+1}}}{\sum_{V_j \in \mathcal{V}_i^{t_{c+1}}} \overline{D}_{jr_k}^{t_{c+1}}} \tag{4.21}$$

Subsequently, the percentage of unallocated resource for VM V_x at time t_{c+1} is determined by:

$$\gamma_{r_k}^{x}(t_{c+1}) = \frac{\omega_{xr_k}^{t_{c+1}} \times ORA_{r_k}^{t_{c+1}}(H_i)}{\overline{D}_{xr_k}^{t_{c+1}}} \times 100 \tag{4.22}$$

Migration Candidate Selection Decision: The framework employs a binary decision variable $MC^{t_{c+1}}(V_x)$ to determine whether VM V_x qualifies as a migration candidate at time t_{c+1}. The decision criterion evaluates whether the cumulative SLO violation would exceed the acceptable threshold:

$$MC^{t_{c+1}}(V_x) = \begin{cases} 1 & \text{if } \left(Cu_{r_k}^{x}(t_c) + \gamma_{r_k}^{x}(t_{c+1})\right) > 1 - \xi \\ 0 & \text{otherwise} \end{cases} \tag{4.23}$$

This formulation ensures that migration is triggered only when the projected SLO violation would breach the acceptable service level, thereby preventing unnecessary migrations due to transient workload spikes.

4.5.2.2 Stochastic Estimation of SLO Violation

Algorithm `SLAViolation(`H_i, t`)` implements a systematic framework for identifying migration candidates based on cumulative SLO violations. The algorithm employs an iterative probabilistic approach to estimate aggregate resource consumption and evaluate each VM's contribution to resource overloading.

The estimation process uses iterative probability calculations to determine aggregate resource consumption $\sum_{V_j \in \mathcal{V}_i^{t_{c+1}}} \overline{D}_{jr_k}^{t_{c+1}}$. At each iteration α, the algorithm calculates the probability that total consumption of resource type $r_k \in \lambda_i^{t_{c+1}}$ exceeds threshold $(C_{ir_k} + \alpha \cdot \Delta_k)$, where $\Delta_k = \eta \cdot C_{ir_k}$ represents a fractional increment with

SLAViolation(H_i, t): Selection of migration candidates based on SLO violation

Input: $\mathcal{V}_i^t$: $\{V_j \mid X_{ij}^t = 1\}$, set of VMs allocated to host H_i,
λ_i^t: set of overutilized resources on host H_i at time t,
η, θ, ξ: system parameters, C_{H_i}: $\{C_{ir_k} \mid \forall r_k \in \lambda_i^t\}$, capacity vector,
$Cu^x(t-1)$: $\{Cu_{r_k}^x(t-1) \mid \forall r_k \in \lambda_i^t\}$, historical resource deficit.
Output: $MC_{H_i}^t$: $\{V_x \mid \forall V_x : MC^t(V_x) = 1\}$, migration candidates.

$MC_{H_i}^t \leftarrow \emptyset$ // Initialize migration candidate set
for each $V_x \in \mathcal{V}_i^t$ **do**
  for each $r_k \in \lambda_i^t$ **do**
    $\alpha \leftarrow 0$ // Initialize scaling factor
    $\alpha' \leftarrow 0$ // Initialize scaling factor for VM removal
    do
      $\alpha \leftarrow \alpha + 1$ // Increment scaling factor
      $Pb_i^{r_k} \leftarrow Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t > C_{ir_k}(1 + \alpha \cdot \eta)\right)$ // Update probability
    while $Pb_i^{r_k} > \theta$ // Find resource consumption with all VMs;
    $RC_i^{r_k} \leftarrow C_{ir_k}(1 + (\alpha - 1) \cdot \eta)$ // Total resource consumption
    do
      $\alpha' \leftarrow \alpha' + 1$ // Increment scaling factor
      $Pb_{i-x}^{r_k} \leftarrow Pr\left(\sum_{V_j \in \mathcal{V}_i^t \setminus \{V_x\}} \overline{D}_{jr_k}^t > C_{ir_k}(1 + \alpha' \cdot \eta)\right)$ // Update probability
    while $Pb_{i-x}^{r_k} > \theta$ // Find consumption without VM V_x;
    $RC_{i-x}^{r_k} \leftarrow C_{ir_k}(1 + (\alpha' - 1) \cdot \eta)$ // Consumption without VM
    $ORA^{r_k} \leftarrow RC_i^{r_k} - C_{ir_k}$ // Overloaded resource amount
    $RC_x^{r_k} \leftarrow RC_i^{r_k} - RC_{i-x}^{r_k}$ // VM's resource consumption
    $\omega_x^{r_k} \leftarrow RC_x^{r_k} / RC_i^{r_k}$ // VM's fractional contribution
    $\gamma_{r_k}^x \leftarrow [(\omega_x^{r_k} \times ORA^{r_k}) / RC_x^{r_k}] \times 100$ // Percentage violation
    if $Cu_{r_k}^x(t-1) + \gamma_{r_k}^x > 1 - \xi$ // Check violation threshold **then**
      Add V_x to $MC_{H_i}^t$ // Select as migration candidate
      break // Move to next VM
    end
  end
end
return $MC_{H_i}^t$

$\eta \in [0, 1]$. The probability at iteration α is:

$$\Pr^{r_k}(\mathcal{V}_i^{t_{c+1}})^{\alpha} = \Pr\left(\sum_{V_j \in \mathcal{V}_i^{t_{c+1}}} \overline{D}_{jr_k}^{t_{c+1}} > C_{ir_k}(1 + \alpha \cdot \eta)\right) \tag{4.24}$$

When $\Pr^{r_k}(\mathcal{V}_i^{t_{c+1}})^{\alpha} \leq \theta$ holds at iteration q, the estimated aggregate consumption is assigned as $C_{ir_k}(1 + (q-1) \cdot \eta)$. Similarly, for estimating consumption excluding VM V_x, the algorithm applies the same procedure:

$$\Pr^{r_k}(\mathcal{V}_i^{t_{c+1}} \setminus V_x)^{\alpha} = \Pr\left(\sum_{V_j \in \mathcal{V}_i^{t_{c+1}} \setminus V_x} \overline{D}_{jr_k}^{t_{c+1}} > C_{ir_k}(1 + \alpha \cdot \eta)\right) \tag{4.25}$$

The parameter η balances estimation accuracy and computational efficiency—larger values compromise precision while smaller values increase iterations.

The algorithm operates through nested iterations where the outer loop examines each VM $V_x \in \mathcal{V}_i^t$ while the inner loop processes each overutilized resource $r_k \in \lambda_i^t$. For each VM-resource pair, scaling factors are determined through iterative probability calculations (`lines 6-9, 11-14`), ensuring computed consumption values reflect realistic overload scenarios. Total resource consumption $RC_i^{r_k}$ and consumption excluding target VM $RC_{i-x}^{r_k}$ are calculated (`lines 10, 15`), enabling precise assessment of each VM's contribution.

The SLO violation percentage $\gamma_{r_k}^x$ is computed (`line 19`) based on VM's fractional contribution $\omega_x^{r_k}$ to overloaded resource amount ORA^{r_k} (`lines 16-18`). Cumulative violation tracking combines current violation with historical deficit $Cu_{r_k}^x(t-1)$, comparing against permissible threshold $1 - \xi$ (`lines 20-23`). VMs exceeding this threshold are immediately selected as migration candidates, with the algorithm proceeding to the next VM for computational efficiency.

4.5.3 Selection of Final Migrating VMs

Algorithm `MigratingVMs`(H_i, t) integrates outputs from migration score calculation and SLA violation analysis to determine the optimal set of VMs for migration. It operates through three sequential phases: aggregation of SLA-compliant candidates with their migration scores, priority-based sorting, and iterative threshold-based

MigratingVMs(H_i, t): Selection of migrating VMs from host H_i at time t.

Input: S_i^t: $\{V_j \mid X_{ij}^t = 1\}$, set of VMs allocated to host H_i at time t,
λ_i^t: set of overutilized resources on host H_i at time t,
θ: overload probability threshold for load relief confirmation,
$MS_{H_i}^t$: $\{MS_{ix}^t \mid \forall V_x \in S_i^t\}$, migration scores for all VMs on H_i,
$MC_{H_i}^t$: set of migration candidates based on SLA violation analysis,
C_{H_i}: $\{C_{ir_k} \mid \forall r_k \in \lambda_i^t\}$, capacity vector for overutilized resources.

Output: $FM_{H_i}^t$: Final set of migrating VMs from H_i at time t.

1 $FM_{H_i}^t \leftarrow \emptyset$ `// Initialize final migration set`
2 $SLOV_{H_i}^t \leftarrow \emptyset$ `// Initialize SLA-filtered candidate set`
3 $LoadRelieved \leftarrow 0$ `// Initialize load relief status`
4 **for** each $V_x \in MC_{H_i}^t$ **do**
5 | Add (V_x, MS_{ix}^t) to $SLOV_{H_i}^t$ `// Pair VM with migration score`
6 **end**
7 Sorted.$SLOV_{H_i}^t \leftarrow \text{Sort}(SLOV_{H_i}^t, MS_{ix}^t, \text{Descending})$ `// Priority-based ordering`
8 **for** each $(V_x, MS_{ix}^t) \in$ Sorted.$SLOV_{H_i}^t$ **do**
9 | Add V_x to $FM_{H_i}^t$ `// Tentatively select VM`
10 | **for** each $r_k \in \lambda_i^t$ **do**
11 | | $OverloadProb \leftarrow 1 - Pr\left(\sum_{V_j \in S_i^t \setminus FM_{H_i}^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)$ `// Remaining overload probability`
12 | | **if** $OverloadProb \leq \theta$ **then**
13 | | | **if** r_k *is the last element in* λ_i^t **then**
14 | | | | $LoadRelieved \leftarrow true$ `// All resources adequately relieved`
15 | | | **end**
16 | | **else**
17 | | | **break** `// Insufficient relief`
18 | | **end**
19 | **end**
20 | **if** $LoadRelieved$ is 1 **then**
21 | | **break** `// Sufficient VMs selected`
22 | **end**
23 **end**
24 **return** $FM_{H_i}^t$

selection.

The aggregation phase (`lines 4-6`) combines SLA-compliant candidates with their migration scores. The sorting phase (`line 7`) arranges candidates in descending order of migration scores, creating the prioritized list `Sorted.`$SLOV_{H_i}^t$.

The core selection phase (`lines 8-23`) implements iterative threshold-based evaluation. The algorithm selects VMs from the sorted set and checks whether overload probability falls below θ for each overutilized resource (`lines 10-19`). When overload probability exceeds θ for any resource, the algorithm breaks the inner loop and evaluates the next VM. Selection continues until H_i becomes non-overloaded. The `LoadRelieved` variable indicates overload status, set to true only when overload probability for all overutilized resources falls below θ (`lines 13-15`). Upon satisfying this condition, the algorithm terminates and returns $FM^t_{H_i}$. This systematic approach with early termination prevents over-migration while ensuring complete overload resolution across all resource dimensions.

4.6 Placement of Migrating VMs for Optimized Load Distribution

The VM placement module represents the final critical component of the SLA-LB framework, determining optimal destination hosts for selected VMs. This module operates on the principle that effective placement must simultaneously achieve load relief for overloaded hosts while maximizing overall DC resource utilization without creating new overload conditions.

The VM placement process is structured into two interconnected phases (shown in Fig. 4.1). The first phase evaluates hosting capability of all potential destination hosts using probabilistic assessment techniques. The second phase implements an intelligent selection algorithm that identifies the most appropriate destination host based on comprehensive capability scores while ensuring system stability and preventing future overload conditions.

4.6.1 Capability Assessment of Destination Hosts

The destination host selection constitutes a strategic assignment problem mapping migrating VMs from overloaded source hosts to alternative hosts within the DC. The objective is to identify destinations that maximize resource utilization while preventing overload conditions in the near future. Let, the complete set of migrating VMs from all overloaded hosts at time t is denoted as $\mathcal{N}^t_{Mig}$:

$$\mathcal{N}^t_{Mig} = \bigcup_{H_i \in \mathcal{H}^t_{ovr}} \mathcal{N}^{it}_{Mig} \tag{4.26}$$

where $\mathcal{N}^{it}_{Mig}$ represents VMs selected for migration from overloaded host H_i at time t.

Further, assume that $\overline{\mathcal{H}^t_{ovr}}$ represents the set of non-overloaded hosts at time t. Therefore, for each overloaded host $H_i \in \mathcal{H}^t_{ovr}$, the set of potential destination hosts $\mathcal{D}^t_{H_i}$ encompasses all hosts in the DC, including both overloaded and non-overloaded hosts: $\mathcal{D}^t_{H_i} = \mathcal{H}^t_{ovr} \cup \overline{\mathcal{H}^t_{ovr}}$. When considering overloaded hosts as potential destinations, the framework excludes their migrating VMs from capacity calculations, treating them as non-overloaded entities. This recognizes that overloaded hosts will achieve load relief through migration of their selected VMs, making their remaining capacity available for hosting additional workloads. Formally, for an overloaded host $H_d \in \mathcal{H}^t_{ovr}$ being considered as a destination, the effective VM set becomes $\mathcal{V}^t_d \backslash \mathcal{N}^t_{Mig}$, where $\mathcal{V}^t_d$ represents VMs hosted by H_d at time t. This ensures the framework capitalizes on capacity that will become available post-migration, promoting efficient resource utilization.

4.6.1.1 Performance Value Calculation for Destination Hosts

The capability assessment employs the Weighted Sum Model (WSM) methodology, where potential destination hosts serve as alternatives and resource types constitute

decision criteria. For a potential destination host $H_d \in \mathcal{D}^t_{H_i}$, the performance value for resource type r_k at time t is the probability that H_d will not exceed capacity C_{dr_k} considering its current effective workload:

$$PVal^t_{r_k}(H_d) = \Pr\left(\sum_{V_j \in \mathcal{V}^t_d \setminus \mathcal{N}^t_{Mig}} \overline{D}^t_{jr_k} \leq C_{dr_k}\right), \quad r_k \in \mathcal{R} \tag{4.27}$$

The performance value $PVal^t_{r_k}(H_d)$ indicates host H_d's remaining capacity for resource type r_k. Higher values indicate greater available capacity, making the host more suitable for accommodating additional workloads. This probabilistic approach accounts for uncertainty in resource consumption patterns while providing a robust foundation for placement decisions.

4.6.1.2 Suitability of Destination Hosts

Algorithm `DPScore`(H_i, t) systematically evaluates destination host suitability using a multi-criteria decision-making approach. The algorithm calculates destination potentiality scores for all potential destination hosts, enabling identification of optimal placement targets that maximize resource utilization while ensuring effective load redistribution from overloaded sources.

The algorithm prioritizes destination hosts based on capacity availability patterns, particularly focusing on resources critically overutilized in the source host. Suitable destination hosts for VMs in $\mathcal{N}^{it}_{Mig}$ should demonstrate relatively high performance values for resources within the overutilized set λ^t_i compared to the non-overutilized set Υ^t_i, ensuring adequate capacity for resource types causing overload conditions.

The Destination Potentiality Score (DPS) for host H_d as a potential destination for VMs migrating from overloaded host H_i is:

$$DPS_{id}^{t} = \sum_{r_k \in \mathcal{R}} \omega_{ir_k}^{t} \cdot PVal_{r_k}^{t}(H_d) \tag{4.28}$$

where $\omega_{ir_k}^{t}$ represents the weight from VM selection phase (Section 4.5), indicating both migration priority from H_i and consideration priority for available capacity when selecting destinations. This ensures consistency between VM selection and placement phases.

The algorithm operates through a two-level iteration structure. The outer loop processes each potential destination host $H_d \in \mathcal{D}_{H_i}^{t} \setminus \{H_i\}$, while the inner loop evaluates weighted performance across all resource types in $\lambda_i^t \cup \Upsilon_i^t$. For each destination-resource pair, the algorithm determines appropriate weights based on whether the resource belongs to the overutilized (`lines 5-6`) or non-overutilized (`lines 7-8`) set of the source host. The performance value $PVal_{r_k}^{t}(H_d)$ is computed as the probability that destination host H_d will not exceed capacity C_{dr_k} considering its effective workload after excluding migrating VMs (`line 10`). Weighted performance values are accumulated across all resource dimensions (`lines 11-12`) to produce the final destination potentiality score (`line 14`). This ensures VM migration prioritizes destinations that can effectively alleviate resource constraints while maintaining system stability.

4.6.2 Selection of Potential Destination Hosts

Algorithm `DestinationHosts(`H_i, t`)` implements an intelligent destination selection mechanism that maps each migrating VM to its optimal destination host while ensuring system-wide stability and preventing new overload conditions. The algorithm leverages destination potentiality scores from the capability assessment stage to guide placement decisions through systematic evaluation that prioritizes hosts

DPScore(H_i, t): Destination potentiality score calculation for H_i at time t

Input: $\mathcal{V}_i^t$: $\{V_j \mid X_{ij}^t = 1\}$, set of VMs allocated to host H_i
λ_i^t: set of overutilized resources on host H_i at time t
Υ_i^t: set of non-overutilized resources on host H_i at time t
$\mathcal{D}_{H_i}^t$: set of potential destination hosts for H_i
$\mathcal{N}_{Mig}^t$: set of all migrating VMs at time t
$C_{H_1}, \ldots, C_{H_M}$: capacity vectors of all hosts

Output: $DPS_{H_i}^t$: $\{DPS_{id}^t \mid \forall H_d \in \mathcal{D}_{H_i}^t\}$, set of destination potentiality scores

$DPS_{H_i}^t \leftarrow \emptyset$ `// Initialize potentiality score set`
for *each* $H_d \in \mathcal{D}_{H_i}^t \setminus \{H_i\}$ **do**
  $WgtSum \leftarrow 0$ `// Initialize weighted sum accumulator`
  for *each* $r_k \in \lambda_i^t \cup \Upsilon_i^t$ **do**
    if $r_k \in \lambda_i^t$ **then**
      $\omega_{ir_k}^t \leftarrow \left[\Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)\right]^{-1}$ `// Weight assignment for overutilized resources`
    else
      $\omega_{ir_k}^t \leftarrow \Pr\left(\sum_{V_j \in \mathcal{V}_i^t} \overline{D}_{jr_k}^t \leq C_{ir_k}\right)$ `// Weight assignment for non-overutilized resources`
    end
    $PVal_{r_k}^t(H_d) \leftarrow \Pr\left(\sum_{V_j \in \mathcal{V}_d^t \setminus \mathcal{N}_{Mig}^t} \overline{D}_{jr_k}^t \leq C_{dr_k}\right)$ `// Performance value`
    $WgtPVal_d^{r_k} \leftarrow \omega_{ir_k}^t \times PVal_{r_k}^t(H_d)$ `// Weighted performance`
    $WgtSum \leftarrow WgtSum + WgtPVal_d^{r_k}$ `// Accumulate weighted values`
  end
  $DPS_{id}^t \leftarrow WgtSum$ `// Assign destination potentiality score`
  Add DPS_{id}^t to $DPS_{H_i}^t$ `// Store in result set`
end
return $DPS_{H_i}^t$

based on capacity availability and resource suitability.

The algorithm operates on the principle that the most suitable destination host for a migrating VM can accommodate additional workload without violating overload threshold θ for any resource type. This ensures placement decisions resolve existing overload conditions while preventing propagation of overload states across the DC infrastructure. The algorithm receives as input the set of migrating VMs $\mathcal{N}_{Mig}^t$ from the VM selection phase and destination potentiality scores $DPS_{H_i}^t$ from

the capability assessment phase, producing an updated placement matrix $\mathcal{X}^*$ reflecting optimal post-migration VM-to-host assignments.

`DestinationHosts(`H_i, t`)` implements a three-level nested iteration structure. The outermost loop processes each migrating VM from $\mathcal{N}_{Mig}^{it}$, sorting destination potentiality scores in descending order to prioritize capable hosts (`line 4`). The middle loop evaluates each potential destination host in sorted order. The innermost loop systematically evaluates each resource type in the destination host's resource set (`line 7`), calculating overload probability from accommodating the migrating VM (`line 8`). If probability remains within threshold (`line 9`) and all resources are evaluated (`line 10`), the load indicator is set to one (`line 11`). Otherwise, resource evaluation terminates early (`lines 13-14`). When a destination host passes all compatibility tests (`line 17`), the algorithm assigns the host to the migrating VM (`line 18`), updates the placement matrix (`line 19`), recalculates the host's destination potentiality score (`line 20`), updates the score in the candidate set (`line 21`), and proceeds to the next VM. This systematic approach ensures each migrating VM is assigned to the most suitable available destination host while maintaining system-wide load balance and stability.

4.7 Performance Evaluation

This section evaluates the proposed SLA-LB approach through systematic comparative analysis against established state-of-the-art methods. The evaluation assesses the stochastic overload detection mechanism against traditional threshold-based approaches, then examines how the probabilistic resource characterization, VM selection heuristics, and destination host selection algorithms function together as an integrated solution.

The evaluation compares SLA-LB against three prominent approaches: RIAL

DestinationHosts(H_i, t): Destination host selection and placement matrix generation for host H_i at time t

Input: $\mathcal{V}i^t$: $V_j \mid Xij^t = 1$, set of VMs allocated to host H_i
λ_i^t: set of overutilized resources on host H_i at time t
Υ_i^t: set of non-overutilized resources on host H_i at time t
θ: overload probability threshold, $\mathcal{X}$: current placement matrix
$\mathcal{N}_{Mig}^{it}$: set of migrating VMs from host H_i at time t
$DPS_{H_i}^t$: destination potentiality scores for host H_i
$C_{H_1}, \ldots, C_{H_M}$: capacity vectors of all hosts

Output: $\mathcal{X}^*$: updated placement matrix after migration

$Load \leftarrow 0$ // Initialize load assignment indicator
$\mathcal{X}^* \leftarrow \mathcal{X}$ // Initialize output matrix with current placement
for *each* $V_x \in \mathcal{N}_{Mig}^{it}$ **do**
  Sorted.$DPS_{H_i}^t \leftarrow \text{Sort}(H_i, DPS_{H_i}^t, \text{Descending})$ // Prioritize hosts by capability
  for *each* $H_d \in DPS_{H_i}^t$ **do**
    $Load \leftarrow 0$ // Reset load indicator for current destination
    for *each* $r_k \in \lambda_d^t \cup \Upsilon_d^t$ **do**
      $OvrPrb \leftarrow 1 - \Pr\left(\sum_{V_j \in \mathcal{V}_d^t \setminus \mathcal{N}_{Mig}^t \cup \{V_x\}} \overline{D}_{jr_k}^t \leq C_{dr_k}\right)$ // Calculate overload probability
      if $OvrPrb \leq \theta$ **then**
        if r_k *is the last element in* $\lambda_d^t \cup \Upsilon_d^t$ **then**
          $Load \leftarrow 1$ // Mark host as a suitable one
        end
      else
        break // Exit resource loop if θ exceeded
      end
    end
    if $Load = 1$ **then**
      $H_d \leftarrow \text{DestinationHost}(V_x)$ // Make H_d as destination of V_x
      Set $X_{ix}^* = 0$ and $X_{dx}^* = 1$ in $\mathcal{X}^*$ // Update placement matrix
      Recalculate DPS_{id}^t for host H_d // Update DPS
      Update DPS_{id}^t in $DPS_{H_i}^t$ // Refresh score in candidate set
      break // Exit destination loop and process next VM
    end
  end
end
return $\mathcal{X}^*$

[56], Sandpiper [55], and CloudScale [65], representing different paradigms from reactive threshold-based to proactive prediction techniques. A Python 3.0-based simulation framework provides controlled and reproducible conditions while accurately modeling multidimensional resource characteristics of modern DCs.

The evaluation focuses on four critical metrics: number of VM migrations (system stability), number of overloaded hosts (load distribution efficiency), host resource consumption patterns (infrastructure utilization), and total performance degradation due to migrations (service quality impact). Real-world workload traces including PlanetLab [20] and Google Cloud Cluster (GCC) traces [19] ensure practical relevance while maintaining reproducible conditions. The methodology systematically examines how the stochastic approach handles workload uncertainty compared to deterministic methods.

4.7.1 Experimental Setup

The experimental evaluation employs a comprehensive simulation-based approach to assess performance of the proposed SLA-aware stochastic load balancing framework under realistic cloud computing conditions. The simulation infrastructure operates on an Intel(R) Core(TM) i5-9400F CPU @ 2.90 GHz processor with 16 GB RAM running on a 64-bit Windows 10 Operating System, providing adequate computational capacity for large-scale DC simulations while maintaining reasonable execution times for the three-resource environment encompassing CPU, memory, and bandwidth.

4.7.1.1 DC Configuration

The simulated heterogeneous DC comprises 1000 hosts hosting 8000 VMs, reflecting realistic cloud infrastructure scales. The heterogeneity is modeled through two distinct host types based on commercial hardware specifications: The first host type, modeled after the Hitachi HA8000/SS10 (DK1), features a dual-core CPU with 3067 MIPS per core, 8GB memory, and 1000 Mbps network bandwidth. The second host

type, based on HP ProLiant ML110 G5, incorporates a dual-core CPU with 2660 MIPS per core, 4GB memory, and 800 Mbps network bandwidth. The DC maintains an equal distribution with 500 hosts of each type, ensuring balanced heterogeneity.

4.7.1.2 Workload Description

The evaluation incorporates real-world workload patterns through two established trace datasets: PlanetLab and GCC traces. The PlanetLab trace captures CPU utilization measurements from VMs within the PlanetLab platform, recorded at five-minute intervals across 10 randomly selected days in March and April 2011. The GCC trace provides comprehensive CPU and memory utilization data from approximately 11,000 hosts operating within a production cluster over 29 days in May 2011.

To address multidimensional resource requirements, memory and bandwidth utilization patterns are generated using statistical distributions. Five distinct resource utilization groups are defined with specific (mean, variance) parameters: (0.2, 0.05), (0.2, 0.15), (0.3, 0.05), (0.6, 0.10), and (0.6, 0.15). Each VM's memory and bandwidth utilization values are assigned through random selection from these predefined groups, ensuring realistic resource consumption diversity while maintaining statistical consistency.

4.7.1.3 Experimental Methodology

The experimental procedure initiates with random VM allocation across available hosts, establishing a baseline configuration that remains consistent across all evaluated load balancing algorithms to ensure fair comparative analysis. The simulation duration spans 30 hours for both workload traces, with the initial 6 hours designated as training data for parameter estimation and calibration.

During simulation execution, resource utilization status of all hosts undergoes

evaluation at 5-minute intervals, maintaining consistency with trace data granularity. The monitoring system continuously records VM migration frequency, overloaded host identification, and comprehensive resource utilization patterns across all infrastructure components throughout the evaluation period.

4.7.1.4 Parameter Configuration

The statistical parameter estimation employs Exponentially Weighted Moving Average (EWMA) methodology with a span value of 4 and alpha coefficient calculated as $[2/(span+1)]$. This configuration provides exponentially decreasing weights for historical data points, proving particularly effective for short-term prediction in highly dynamic cloud environments. The optimal span value determination involves systematic evaluation across a range from 2 to 10 using the designated test dataset.

The overload probability threshold (θ) is established at 0.95, meaning a host is classified as overloaded when the overload probability for any of its CPU, memory, or bandwidth resources exceeds this threshold. For comparative baseline algorithms (RIAL, Sandpiper, and CloudScale), a resource utilization threshold of 0.75 is employed to maintain consistency with established evaluation methodologies. The probability calculation incorporates the 30 most recent resource consumption observations, providing sufficient historical context while maintaining responsiveness to recent utilization trends. For SLA profiling configuration, the satisfaction threshold (ξ) is set to 90%, requiring that 90% of each VM's total resource requirements must be fulfilled throughout its operational lifetime. The capacity utilization parameter (η) is configured at 0.2, indicating that 20% of each host's total capacity contributes to overload resource amount calculations, ensuring conservative resource management while maintaining system efficiency.

4.8 Experimental Results

The experimental results demonstrate the effectiveness of the stochastic approach in addressing key challenges of cloud resource management. The performance analysis reveals distinct operational characteristics that emerge from probabilistic decision-making compared to traditional threshold-based and prediction-oriented methods. The findings highlight significant improvements in system stability, resource efficiency, and service quality maintenance across diverse workload scenarios. Notable patterns emerge in the temporal behavior of the stochastic framework, particularly in its ability to balance immediate performance requirements with long-term operational objectives.

4.8.1 VM Migration Analysis

The migration frequency analysis reveals significant advantages of the stochastic approach in maintaining system stability while achieving effective load distribution. Figures 4.2 and 4.3 show the total number of VM migrations measured at 6-hour intervals across all evaluated methods for both workload traces. The results demonstrate that SLA-LB consistently outperforms alternative approaches in minimizing unnecessary migrations while maintaining effective load balancing.

The superior performance of SLA-LB stems from its probabilistic VM selection mechanism, which prioritizes VMs that are expected to provide maximum load relief from overloaded hosts with the highest probability. Additionally, the framework constrains VM migrations based on their SLO violation percentage and implements destination host selection strategies that proactively prevent future overloading scenarios. This approach reduces long-term migration requirements compared to reactive methods. The temporal migration patterns reveal interesting characteristics of the SLA-aware approach. During initial simulation periods, most VMs operate

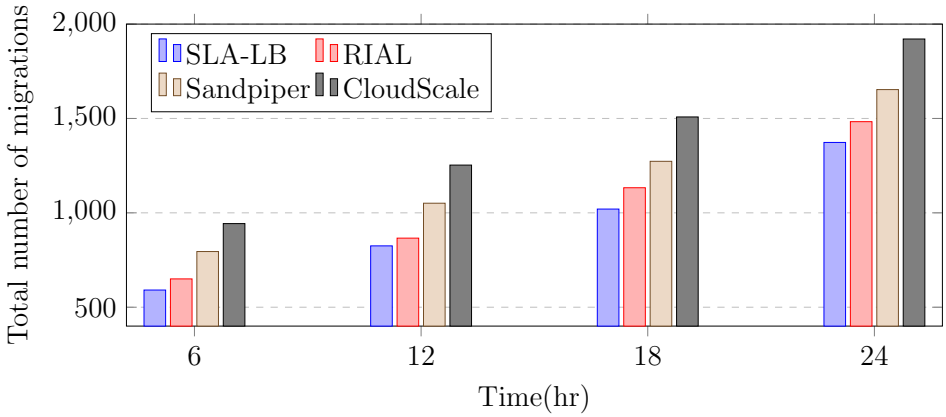

FIGURE 4.2: The number of VM migrations using PlanetLab trace.

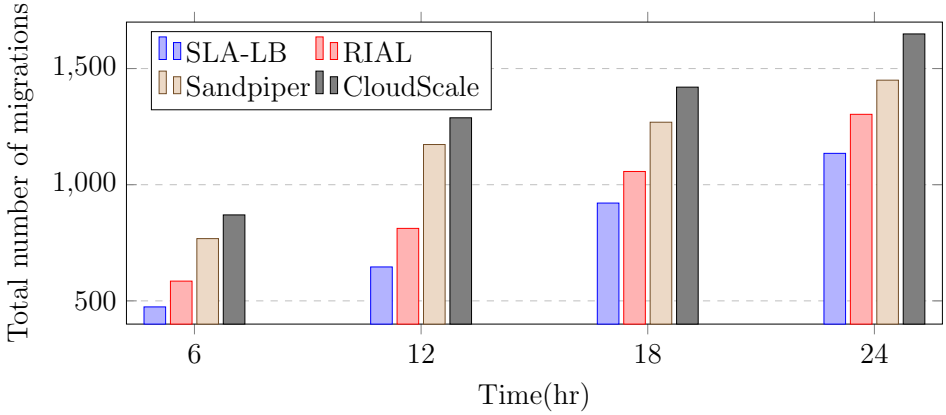

FIGURE 4.3: The number of VM migrations using GCC trace.

below their allowable SLO violation threshold ($\xi - 1$), resulting in reduced migration activity. As the simulation progresses, VMs gradually approach and exceed this threshold, leading to increased migration activity. This pattern demonstrates the framework's ability to balance immediate performance requirements with long-term SLA compliance.

Comparative analysis across methods shows distinct behavioral patterns. CloudScale generates the highest number of migrations due to its proactive nature, migrating VMs for both correctly and incorrectly predicted overload scenarios. The

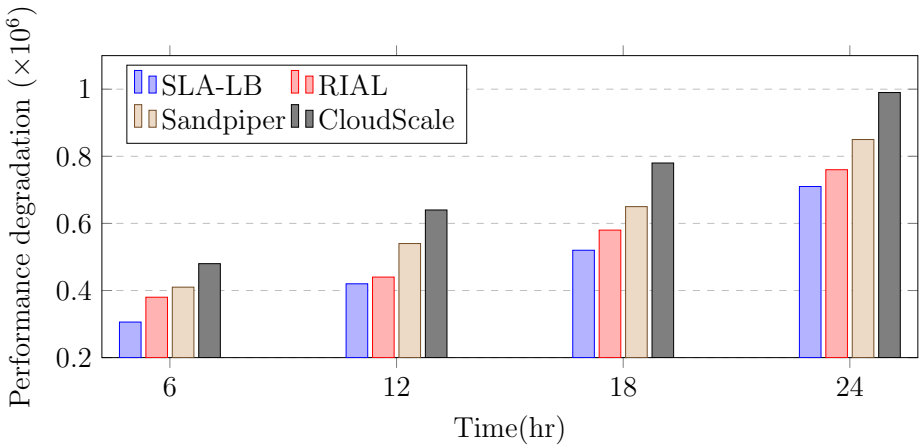

FIGURE 4.4: VMs performance degradation using PlanetLab trace.

reactive approaches (Sandpiper and RIAL) limit migrations to actual overload occurrences, with RIAL producing fewer migrations than Sandpiper due to its differentiated resource weighting mechanism. Quantitative results over a 24-hour period using PlanetLab trace demonstrate that SLA-LB generates 10%, 20%, and 32% fewer VM migrations compared to RIAL, Sandpiper, and CloudScale, respectively. The GCC trace yields even more favorable results with reductions of 15%, 31%, and 39% respectively. The higher migration frequency observed with PlanetLab trace reflects its relatively higher workload intensity compared to GCC trace, resulting in more frequent host overloading scenarios.

4.8.2 Assessment of Performance Degradation

VM migration inherently impacts application performance through temporary resource unavailability and network overhead. The evaluation quantifies performance degradation using established models that account for CPU utilization impact during migration periods.

The time required for migrating a VM depends on the memory consumed by the VM and the available network bandwidth. The performance degradation D of a VM

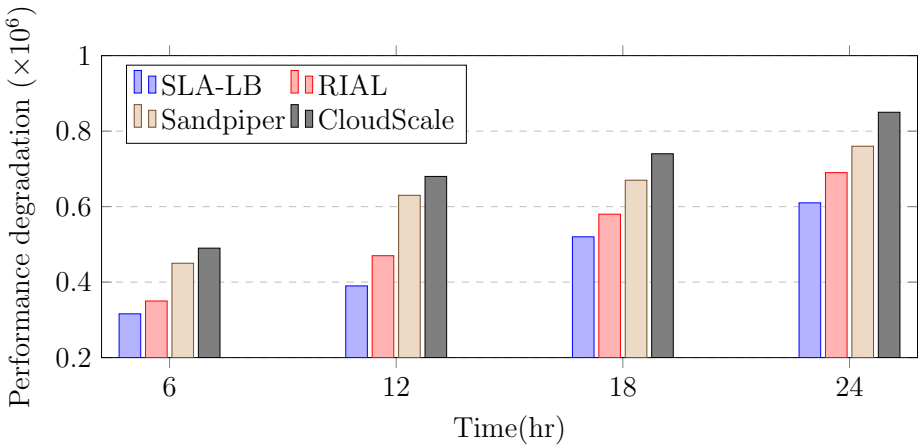

FIGURE 4.5: VMs performance degradation using GCC trace.

j resulting from migration is defined as:

$$D = \frac{1}{10} \int_{t}^{t+\frac{M}{B}} u_j(t)dt \tag{4.29}$$

where, t is the time when migration starts, M is the amount of memory used by VM j, B is the available network bandwidth, $\frac{M}{B}$ indicates the time to complete the migration and $u_j(t)$ is the CPU utilization of VM j.

Figures 4.4 and 4.5 illustrate the migration-induced performance degradation observed across the two traces, respectively, demonstrating the consistent superiority of SLA-LB across both workloads. The reduced degradation directly correlates with lower migration frequency, as fewer migrations inherently result in reduced overall performance impact. Additionally, the framework's VM selection heuristic prioritizes VMs with lower consumption of non-overutilized resources, further minimizing performance impact during migration operations. The performance degradation hierarchy follows the pattern: SLA-LB < RIAL < Sandpiper < CloudScale for both evaluated traces. Over 24-hour periods, SLA-LB achieves performance degradation reductions of 9.5%, 20%, and 32% compared to RIAL, Sandpiper, and CloudScale

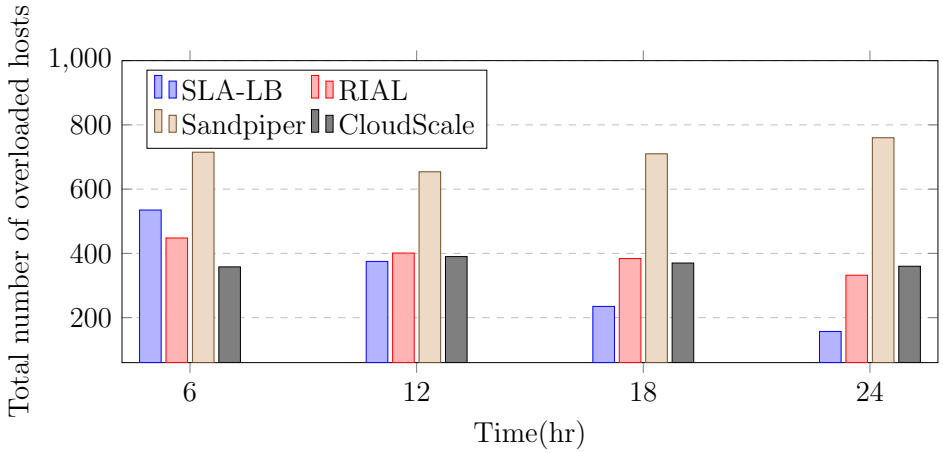

FIGURE 4.6: The number of overloaded hosts using PlanetLab trace.

respectively using PlanetLab trace. GCC trace demonstrates even greater improvements with reductions of 12%, 27%, and 33% respectively.

4.8.3 Number of Overloaded Hosts

The analysis of overloaded host frequency provides insights into load distribution effectiveness and system stability maintenance. Figures 4.6 and 4.7 present the frequency of overloaded hosts across two traces. Results measured at 6-hour intervals demonstrate SLA-LB's superior long-term performance in maintaining balanced load distribution across the DC infrastructure. The temporal patterns reveal characteristic differences between proactive and reactive approaches. SLA-LB initially exhibits higher overloaded host counts due to SLO violation constraints limiting immediate migration actions. However, the framework demonstrates consistent improvement over time as the destination host selection strategy maintains long-term load balance through probabilistic placement decisions.

The proactive nature of both SLA-LB and CloudScale enables overload prediction and preemptive VM migration, resulting in lower overloaded host counts compared to reactive approaches (RIAL and Sandpiper). The reactive methods can only respond

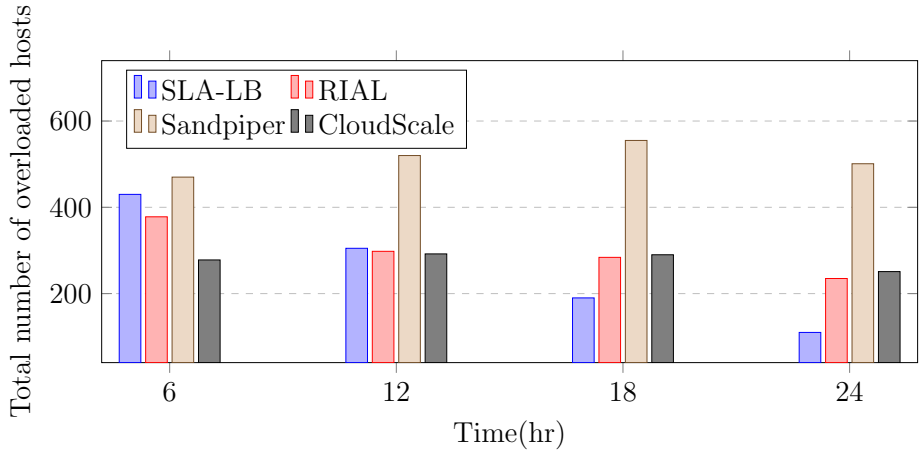

FIGURE 4.7: The number of overloaded hosts using GCC trace.

TABLE 4.1: Summary of Key QoS Metrics Across Load Balancing Methods

Method	**Avg. Migration Count/hr**		**Avg. Response Time Overhead (ms/hr)**		**Avg. SLA Violations/hr**	
	PlanetLab	GCC	PlanetLab	GCC	PlanetLab	GCC
SLA-LB	162.08	132.33	1246	1024	54.25	43.12
RIAL	172.17	156.54	1288	1152	65.20	49.79
Sandpiper	198.83	194.17	1634	1498	118.29	85.25
CloudScale	234.37	217.79	1520	1368	61.58	46.29

to overload occurrences after they manifest, limiting their effectiveness in preventing overload situations. Quantitative analysis over 24-hour periods reveals significant improvements in overload management. Using PlanetLab trace, SLA-LB achieves a total of 1302 overloaded hosts, representing reductions of 17%, 54%, and 12% compared to RIAL, Sandpiper, and CloudScale respectively. GCC trace results show 1035 overloaded hosts using SLA-LB, with corresponding reductions of 13%, 49%, and 7% compared to the alternative approaches.

Table 4.1 summarizes the average QoS metrics across all evaluated methods for both workload traces. The average migration count per hour reflects the rate of VM migrations over the 24-hour simulation period, directly indicating system stability and migration overhead. The average response time overhead per hour is derived

from the migration time component in Equation 4.29, where migration duration depends on the ratio of VM memory to available network bandwidth. The average SLA violations per hour are captured through the hourly rate of overloaded hosts, as host overloading directly leads to resource deficits and service level breaches. SLA-LB consistently achieves the lowest average migration count and response time overhead across both traces, reflecting the effectiveness of its SLO-aware VM selection. It also maintains the lowest average SLA violations per hour, while Sandpiper records the highest due to its purely reactive detection mechanism. Overall, SLA-LB delivers the best balanced performance across all three QoS dimensions simultaneously.

4.8.4 Resource Utilization Efficiency

Resource utilization analysis focuses on maximizing infrastructure efficiency while maintaining overload probability below the established threshold θ. The evaluation examines both CPU and memory utilization patterns across 6-hour intervals for all evaluated approaches using both workload traces. The corresponding CPU and memory utilization trends are presented in Figures 4.8–4.11. The utilization trends presented in these figures capture both steady-state behavior and transient effects that occur during VM migrations. These patterns reflect realistic dynamic cloud conditions observed throughout the full operational period, beyond the initial 6-hour training phase.

The resource utilization hierarchy consistently follows the pattern: SLA-LB > RIAL > Sandpiper > CloudScale across both CPU and memory dimensions. The stochastic methods employed in overload detection and destination host selection enable SLA-LB to maintain the average CPU and memory utilization of the DC at approximately 68% and 57% respectively over 24-hour periods across both workload traces.

The superior performance of SLA-LB compared to RIAL, despite both employing

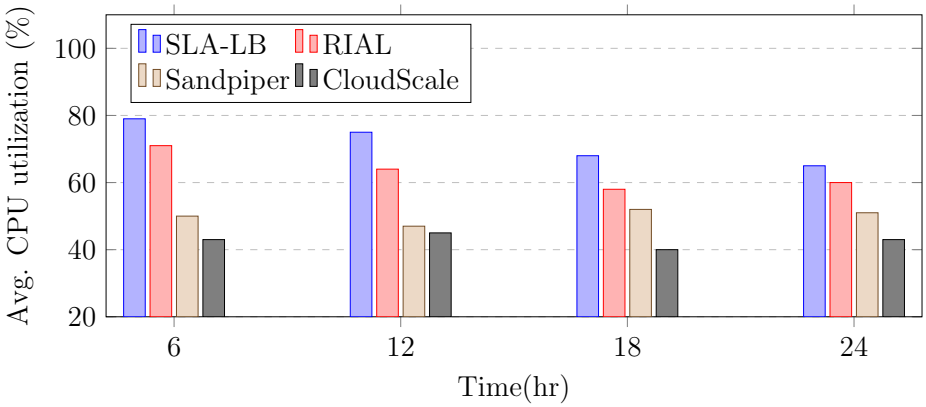

FIGURE 4.8: Average CPU utilization using PlanetLab trace.

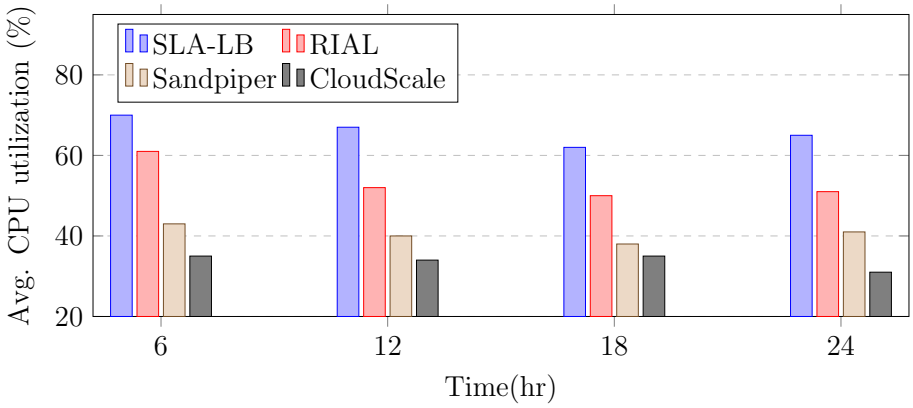

FIGURE 4.9: Average CPU utilization using GCC trace.

stochastic methods, stems from fundamental differences in destination host selection strategies. SLA-LB considers virtually all hosts for destination selection regardless of their current overload status, while RIAL restricts consideration to non-overloaded hosts only. This broader selection scope enables more effective resource utilization while maintaining probabilistic overload guarantees. The comprehensive resource utilization results demonstrate SLA-LB's effectiveness in achieving the dual objectives of maximizing infrastructure efficiency and maintaining service quality guarantees.

The framework's ability to maintain higher utilization levels while simultaneously

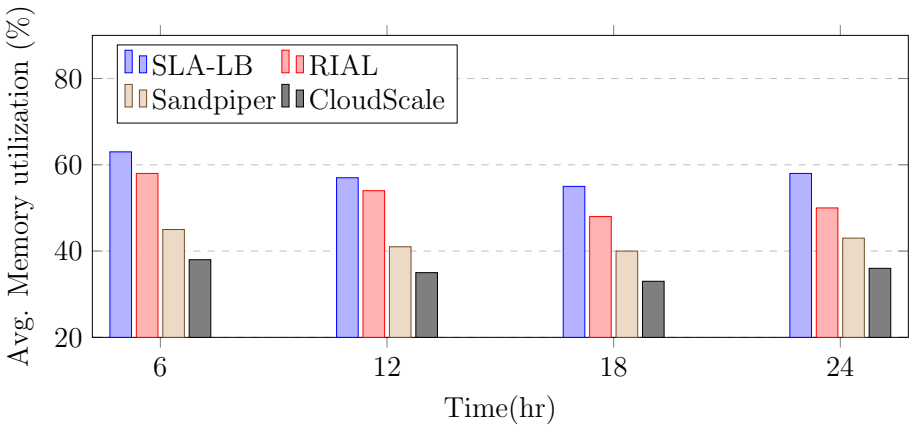

FIGURE 4.10: Average Memory utilization using Planetlab trace.

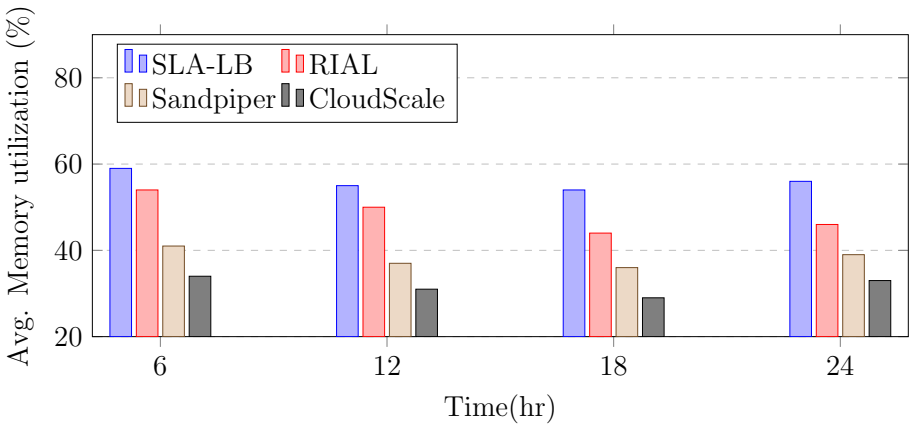

FIGURE 4.11: Average Memory utilization using GCC trace.

reducing migration frequency and overload occurrences validates the effectiveness of the stochastic approach in cloud resource management scenarios.

4.9 Summary

The SLA-aware load balancing framework successfully addressed the dynamic resource management challenges that emerged from the static placement strategies of Chapter 3. Through probabilistic overload detection, intelligent VM selection, and strategic destination placement algorithms, the SLA-LB framework achieved

substantial improvements in service quality maintenance while optimizing resource utilization in dynamic cloud environments.

Comprehensive experimental evaluation using real-world workload traces validated the framework's effectiveness across multiple performance dimensions. The SLA-LB approach achieved 10-39% fewer VM migrations compared to existing methods, maintained superior resource utilization efficiency, and prevented SLA violations through cumulative violation tracking and risk-aware migration strategies. The framework demonstrated 17-54% fewer overloaded hosts and 9-33% reduction in performance degradation compared to existing approaches. The probabilistic foundation from Chapter 3 proved instrumental in enabling sophisticated overload detection that anticipates resource deficit situations before they manifest as service quality issues. The integration of workload prediction capabilities with dynamic load balancing created a comprehensive solution that maintained energy efficiency benefits from optimal initial placement while continuously adapting to evolving operational conditions.

Despite these achievements, the SLA-LB framework reveals fundamental energy efficiency constraints. The reactive load balancing nature, while proactive in overload detection, maintains multiple active hosts to accommodate dynamic workload redistribution, preventing aggressive consolidation opportunities. Resource fragmentation patterns emerge where successful load balancing distributes VMs across multiple hosts in configurations that maintain excellent service quality but result in suboptimal energy consumption. The energy-service quality trade-off becomes apparent as load balancing requires maintaining sufficient spare capacity across hosts for migration targets, directly conflicting with energy-efficient consolidation strategies that minimize active servers. This temporal resource fragmentation accumulates over time, creating scenarios where numerous hosts operate at moderate utilization rather than achieving high-density consolidation on fewer servers.

Effective cloud resource management requires a third critical component: intelligent VM consolidation that actively pursues energy efficiency while maintaining the stability and service quality achieved through load balancing. The foundation established through SLA-aware load balancing provides essential building blocks for energy-efficient consolidation, but requires addressing new challenges including consolidation opportunity identification, energy-aware migration strategies, and dynamic algorithms that balance energy savings with system responsiveness. This establishes the critical need for dynamic VM consolidation mechanisms that complement both proactive placement and SLA-aware load balancing, completing the transition toward comprehensive cloud resource management that simultaneously optimizes energy efficiency, maintains service quality, and adapts to dynamic operational conditions.

Chapter 5

Energy Efficient Dynamic VM Consolidation

5.1 Introduction

Building upon the SLA-aware load balancing framework established in Chapter 4, this chapter addresses the critical challenge of achieving comprehensive energy efficiency in cloud data centers through intelligent VM consolidation while maintaining service quality guarantees and dynamic responsiveness. While the SLA-LB framework successfully maintained service quality and optimized resource utilization during dynamic operations, experimental analysis revealed fundamental energy efficiency limitations necessitating sophisticated consolidation strategies. The transition from load balancing to energy optimization presents multifaceted challenges extending beyond service quality maintenance. Resource fragmentation patterns emerge as a primary concern. Load balancing operations distribute VMs across multiple hosts maintaining excellent load distribution and SLA compliance, yet resulting in suboptimal energy consumption [83]. These patterns accumulate over time, creating scenarios where numerous hosts operate at moderate utilization levels

rather than achieving high-density consolidation on minimal active servers, directly conflicting with energy efficiency objectives.

The energy-service quality trade-off represents the fundamental challenge in VM consolidation. Aggressive consolidation toward fewer active hosts can compromise the system's ability to handle dynamic workload variations and maintain responsive load balancing capabilities. Traditional consolidation approaches often create tightly packed configurations that become vulnerable to cascading overload conditions when workload spikes occur [131, 132]. This necessitates sophisticated consolidation strategies that preserve system resilience while maximizing energy savings.

Contemporary cloud workloads exhibit temporal patterns that create natural consolidation opportunities during periods of reduced resource demand [133]. However, identifying and exploiting these opportunities requires advanced analysis capabilities that consider both immediate consolidation benefits and long-term system stability implications. The challenge is further complicated by the multi-dimensional nature of resources. Additionally, the system must maintain sufficient capacity for handling dynamic load variations that the SLA-LB framework was designed to manage.

This chapter introduces a comprehensive Resource-Optimized VM Consolidation (RO-VMC) framework addressing these limitations through innovative integration of energy optimization with service quality preservation. The framework extends stochastic modeling from previous chapters to intelligent consolidation decision-making. It incorporates stochastic load imbalance detection that identifies consolidation opportunities while maintaining load balancing compatibility. The framework also employs resource intensity-aware VM distribution through game-theoretic optimization. Additionally, it implements multi-objective optimization that balances energy minimization with migration reduction. Through this advancement, the chapter makes three primary contributions:

- **Stochastic Load Imbalance Detection Framework**: Development of probabilistic models that identify consolidation opportunities while maintaining compatibility with dynamic load balancing requirements, enabling coordinated optimization across multiple resource management time horizons.

- **Resource Intensity-Aware VM Consolidation**: Design of game-theoretic algorithms that optimize VM distribution for energy efficiency while preserving system resilience and service quality guarantees through strategic resource allocation modeling.

- **Multi-Objective Energy-QoS Optimization**: Integration of energy minimization with service quality preservation through constrained optimization that balances aggressive consolidation with migration overhead control and system stability maintenance.

The significance lies in completing the comprehensive cloud resource management framework, creating an integrated solution that seamlessly coordinates proactive placement, dynamic load balancing, and energy-efficient consolidation to achieve operational excellence through simultaneous optimization of energy efficiency, resource utilization, and service delivery quality.

5.2 System Models

The system model provides the mathematical foundation for the Resource-Optimized VMC (RO-VMC) approach, formally defining the cloud data center environment and relationships between energy consumption, resource utilization, and quality of service requirements. It characterizes the multi-resource infrastructure, workload stochasticity, and the fundamental trade-offs in VM consolidation decisions.

The system operates within a virtualized cloud environment comprising a data center with a set $\mathcal{H} = \{H_i \mid i = 1 \text{ to } M\}$ of M heterogeneous hosts, where each

host H_i is characterized by its resource capacity tuple $H_i = \langle C_{ir_k} \mid \forall r_k \in \mathcal{R} \rangle$ across d resource dimensions $\mathcal{R} = \{r_k \mid k = 1 \text{ to } \mathsf{d}\}$. At time instant t, the system hosts a set $\mathcal{V}^t$ of N^t VMs, where each VM $V_j \in \mathcal{V}^t$ exhibits dynamic resource demands $V_j = \langle D^t_{jr_k} \mid \forall r_k \in \mathcal{R} \rangle$, with $D^t_{jr_k} = f^t_{jr_k} \cdot C_{jr_k}$ representing the actual consumption of resource r_k by VM V_j, where $f^t_{jr_k} \in [0, 1]$ denotes the utilization fraction of the VM's allocated capacity C_{jr_k}. The VM-to-host allocation is represented by the binary allocation matrix $\mathcal{X}^t = [X^t_{ij}]_{M \times N^t}$, where:

$$X^t_{ij} = \begin{cases} 1, & \text{if } V_j \text{ is allocated to } H_i \text{ at } t^{th} \text{ time instant} \\ 0, & \text{otherwise} \end{cases}$$

The utilization of host H_i for resource r_k at time t is defined as:

$$\mathcal{U}^t_{ir_k} = \frac{\sum_j X^t_{ij} \times D^t_{jr_k}}{C_{ir_k}} \times 100 \tag{5.1}$$

Correspondingly, the data center's overall utilization for resource r_k across all active hosts $\mathcal{H}^t_{active} \subseteq \mathcal{H}$ is the following:

$$DC^t_{r_k} = \frac{\sum_{H_i \in \mathcal{H}^t_{active}} \sum_j X^t_{ij} \times D^t_{jr_k}}{\sum_{H_i \in \mathcal{H}^t_{active}} C_{ir_k}} \times 100 \tag{5.2}$$

This foundational framework captures temporal variations in resource consumption patterns and establishes the mathematical basis for subsequent energy optimization and QoS satisfaction models.

The model is structured into three interconnected components: *the Resource Imbalance Model*, which quantifies distribution imbalances across multi-dimensional resources to guide consolidation strategies; *the Power and Energy Consumption Model*, which establishes the relationship between host utilization and energy consumption patterns; and *the QoS Satisfaction Model*, which defines service quality metrics and

violation thresholds to ensure performance guarantees. Each component enables RO-VMC to optimize migration decisions while balancing energy minimization and QoS preservation in multi-resource cloud environments.

5.2.1 Resource Imbalance Model

Dynamic resource allocation requires balancing utilization and performance through threshold-based monitoring. CSPs must prevent both overutilization bottlenecks and underutilization waste. Since VM resource demands fluctuate stochastically, the framework from Section 3.5.2 quantifies multi-dimensional load using probabilistic workload estimation to guide consolidation decisions.

Load imbalance detection uses resource overloading probabilities at host level. A host H_i is *overloaded* if any resource's overloading probability exceeds threshold θ_{over}, and *under-loaded* if all resource's under-loading probabilities exceed threshold θ_{under}. For resource $r_k \in \mathcal{R}$, the overloading and under-loading probabilities of host H_i at time t are:

$$\Pr^t_{\text{over}}(H_i)^{r_k} = 1 - Pr\left(L^t_{r_k}(H_i) \leq C_{ir_k}\right) \tag{5.3}$$

$$\Pr^t_{\text{under}}(H_i)^{r_k} = Pr\left(L^t_{r_k}(H_i) \leq \tau \cdot C_{ir_k}\right) \tag{5.4}$$

where $L^t_{r_k}(H_i) = \sum_j X^t_{ij} \cdot D^t_{jr_k}$ is the aggregate load, and $\tau \in (0, 1)$ is the under-utilization threshold.

Given the normal distribution assumption for VM resource demands $D^t_{jr_k} \sim \mathcal{N}(\mu^t_{r_k}(V_j), \sigma^t_{r_k}(V_j)^2)$, the probability calculations utilize:

$$Pr\left(L^t_{r_k}(H_i) \leq C_{ir_k}\right) = \Phi\left(\frac{C_{ir_k} - \mu^t_{r_k}(H_i)}{\sigma^t_{r_k}(H_i)}\right) \tag{5.5}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function, $\mu^t_{r_k}(H_i) = \sum_j X^t_{ij} \cdot \mu^t_{r_k}(V_j)$ and $\sigma^t_{r_k}(H_i) = \sqrt{\sum_j \left[X^t_{ij} \cdot \sigma^t_{r_k}(V_j)\right]^2}$. Here, $\mu^t_{r_k}(V_j)$ and $\sigma^t_{r_k}(V_j)$ are the mean and standard deviation of V_j's resource demand for resource r_k at time

t, respectively.

A host H_i is classified as overloaded if $\exists r_k \in \mathcal{R} : \Pr^t_{\text{over}}(H_i)^{r_k} > \theta_{over}$ and under-loaded if $\forall r_k \in \mathcal{R} : \Pr^t_{\text{under}}(H_i)^{r_k} > \theta_{under}$. Sets $\mathcal{H}^t_{ovr}$ and $\mathcal{H}^t_{undr}$ denote overloaded and under-loaded hosts respectively. The data center experiences load imbalance when $\mathcal{H}^t_{ovr} \cup \mathcal{H}^t_{undr} \neq \emptyset$, with total imbalanced hosts quantified as $\big(\mid \mathcal{H}^t_{ovr} \mid + \mid \mathcal{H}^t_{undr} \mid \big)$. This probabilistic framework enables RO-VMC to proactively identify consolidation opportunities while maintaining performance guarantees through quantifiable risk assessment.

5.2.2 Power and Energy Consumption Model

To consistently maintain service quality, CSPs must ensure physical resource availability to meet aggregated VM demands. However, data centers may have excess resources relative to available hosts. The number of active hosts depends on the allocation strategy, and power consumption is directly linked to active host count. Therefore, distributing VMs efficiently among hosts is crucial for improving power efficiency, with consolidation within minimal hosts resulting in overall decrease in DC power consumption.

A host's power consumption depends on resource utilization, with CPU having the highest influence. Studies show that host power consumption can be accurately described by a linear relationship with CPU utilization [134]. The CPU utilization-based power model has been widely adopted in the literature [16]. Thus, the power model proposed in [97] estimates host power consumption in a cloud DC. For given p intervals, $\{[0, 1/\mathsf{p}), [1/\mathsf{p}, 2/\mathsf{p}), \ldots, [(\mathsf{p}-1)/\mathsf{p}, 1]\}$ of average CPU utilization, the power model for H_i is:

$$\mathcal{P}ower(H_i) = \alpha_q \,.\, \mathcal{F}\left(\mathcal{U}^t_{ir_k}\right) + \rho^{lower}_q, \frac{q-1}{\mathsf{p}} \leq \mathcal{U}^t_{ir_k} < \frac{q}{\mathsf{p}} \tag{5.6}$$

Here, $r_k = \text{CPU} \in \mathcal{R}$; $\alpha_q (q = 1, \ldots, \mathsf{p})$ is the power difference of H_i for q^{th} interval,

i.e., $\alpha_q = \left(\rho_q^{upper} - \rho_q^{lower}\right)$, where ρ_q^{upper} and ρ_q^{lower} are the upper and lower bound on the power of H_i; and $\mathcal{F}\left(\mathcal{U}_{ir_k}^t\right)$ is the ratio of H_i's CPU utilization. As $\mathcal{U}_{ir_k}^t$ varies with time t, $\mathcal{P}ower(H_i)$ can be regarded as a function of t. Thus, the total energy consumption of H_i during a time interval t_f to t_l (denoted by $EC(H_i)$) is:

$$EC(H_i)^{[t_f \ldots t_l]} = \int_{t=t_f}^{t_l} \mathcal{P}ower(H_i(t))dt \tag{5.7}$$

5.2.3 QoS Satisfaction Model

Quality of Service satisfaction in virtualized cloud environments is fundamentally influenced by the frequency and impact of VM migrations required to maintain system balance. In highly dynamic environments, system load imbalance is unavoidable due to unpredictable resource demand patterns, leading to degraded performance for VMs on overloaded hosts and inefficient resource utilization on under-loaded hosts. While VM consolidation is necessary to address these imbalances, each migration operation introduces significant service disruptions that directly compromise QoS satisfaction.

Migration operations impose dual penalties on service quality: temporary unavailability during VM transfer results in service downtime, while additional resource overhead causes performance degradation in terms of reduced responsiveness and throughput for migrated VMs. These disruptions collectively affect user experience and service delivery, making migration frequency a critical indicator of QoS satisfaction. The QoS Satisfaction Model quantifies these impacts through migration frequency analysis, establishing migration minimization as a primary objective for maintaining service level agreements while achieving load balance.

The QoS impact can be quantified through the total number of VM migrations required during load balancing operations. Given a load imbalance detection at time instant t, let $\mathcal{X}^{t^*} = [X_{ij}^{t^*}]_{M \times N^t}$ represent the post-migration allocation. The

migration indicator variable Y_{ij}^t captures the migration of V_j from H_i at time t:

$$Y_{ij}^t = \begin{cases} 1, & \text{if } X_{ij}^t = 1 \text{ and } X_{ij}^{t^*} = 0 \\ 0, & \text{otherwise} \end{cases}$$

Therefore, the total number of migrations at time t (denoted by $\mathcal{S}_{Mig}^t$) is:

$$\mathcal{S}_{Mig}^t = \sum_{H_i \in \mathcal{H}_{ovr}^t \cup \mathcal{H}_{undr}^t} \sum_{V_j \in \mathcal{V}_*^t} Y_{ij}^t \tag{5.8}$$

where $\mathcal{V}_*^t = \bigcup_{H_i \in \mathcal{H}_{ovr}^t \cup \mathcal{H}_{undr}^t} \mathcal{V}_i^t$ and $\mathcal{V}_i^t \subset \mathcal{V}^t$ is the VM set of host H_i. Migration minimization becomes the primary objective for maintaining SLAs while achieving consolidation goals.

5.3 Problem Description

This section formulates the multi-objective Resource-Optimized VM Consolidation problem in dynamic cloud environments. The fundamental challenge lies in simultaneously minimizing energy consumption and migration-induced QoS violations while maintaining load balance across multi-dimensional resources under stochastic workload conditions. The problem addresses conflicting optimization objectives: energy minimization drives consolidation toward fewer active hosts, while QoS preservation requires limiting migration frequency. This trade-off is complicated by the probabilistic nature of resource demands, creating dynamic imbalances necessitating continuous consolidation adjustments. The formulation integrates probabilistic resource imbalance detection, energy-aware host selection, and migration frequency constraints to achieve system-wide optimization while preserving service level agreements.

5.3.1 Problem Statement

Given a time period T and different set of VMs $\mathcal{V}^t$ running at intervals ($t \mid t = 1 \dots z$), the objective is to operate VMs with minimal energy consumption while ensuring least service interruptions. However, achieving this involves a resource utilization trade-off. Workload uncertainty often poses severe challenges through increasing load imbalance risk, leading to VM migrations that cause QoS deterioration. Therefore, maintaining system load with reduced migrations while reducing operational hosts is critical. To address this, a VM consolidation plan must adjust the mapping between VMs and hosts accordingly.

The notations EC^T and Mig^T represent the data center's total energy consumption and total migrations over period T:

$$EC^T = \sum_{i=1}^{M} EC(H_i)^{[1\dots z]} \quad \text{and} \quad Mig^T = \sum_{t=1}^{z} \mathcal{S}_{Mig}^t$$

Hence, the optimization objectives are:

$$Minimize \begin{cases} EC^T \\ Mig^T \end{cases}$$

To simplify the multi-objective optimization, Mig^T minimization is reformulated as an ϵ-constraint: $Mig^T \leq Mig^T_{\max} \cdot \epsilon$, where $Mig^T_{\max}$ is the maximum migrations during T and $\epsilon \in (0, 1)$. Since maximum migrations cannot exceed VMs running at any time interval, $Mig^T_{\max} = \sum_{t=1}^{z} N^t$. Therefore, the optimization objectives are reformulated as:

$$Minimize\ EC^T$$

Subject to the following constraints

$$Mig^T \leq Mig^T_{\max} \cdot \epsilon \tag{5.9}$$

$$\forall t \in T, \forall H_i \in \mathcal{H}, \forall r_k \in \mathcal{R} : \begin{cases} \Pr^t_{\text{over}}(H_i)^{r_k} < \theta_{over} \\ \Pr^t_{\text{under}}(H_i)^{r_k} < \theta_{under} \end{cases}$$

The value of ϵ can be adjusted based on desired migration tolerance, enabling effective exploration of the energy efficiency and migration reduction trade-off. To systematically address this optimization problem, the following section proposes a Resource Optimized VM Consolidation Framework that leverages advanced optimization techniques, providing an efficient mechanism for dynamic VM placement and migration decisions while maintaining balance between energy consumption and service quality.

5.4 Resource Optimized VM Consolidation Framework

Efficient VM consolidation requires detecting load imbalances and redistributing migrating VMs. The Resource Optimized VM Consolidation Framework (**RO-VMC**) manages VMs by optimizing resource utilization, reducing energy consumption while maintaining QoS. RO-VMC comprises two modules: *Stochastic Load Imbalance Detection* (**StoLID**) for detecting load imbalances by analyzing real-time resource utilization, and *Resource intensity-aware VM Distribution* (**RintVMD**) which distributes VMs based on resource usage intensities using game-theoretic approach to select migrating VMs and destination hosts.

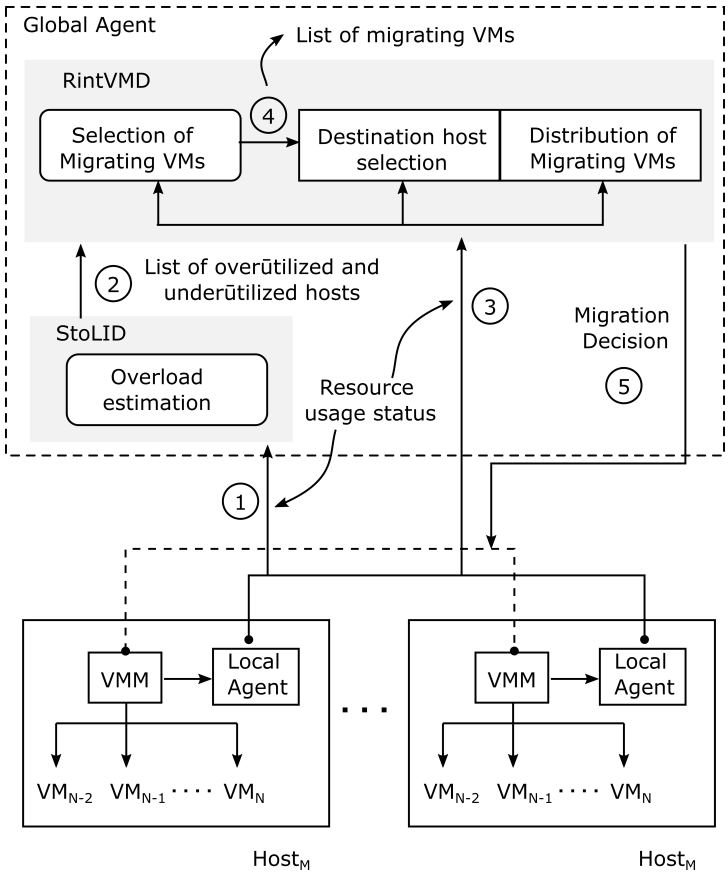

FIGURE 5.1: Architecture of RO-VMC

5.4.1 Architectural Components and Information Flow

The RO-VMC framework adopts a hierarchical approach where the Global Agent operates from a master node for comprehensive optimization decisions while Local Agents provide distributed monitoring. Figure 5.1 illustrates the architecture and information flow among interconnected modules.

The framework comprises specialized modules achieving optimal VM consolidation with minimal energy consumption:

1. *Local Agents (LAs)*: Distributed monitoring modules. Each LA tracks host resource usage and forwards metrics to the Global Agent.

2. *Global Agent (GA)*: Central intelligence hub. Receives data from Local Agents, analyzes system-wide state, and coordinates VM consolidation.

3. *StoLID*: Probabilistic load imbalance detection component. Analyzes resource data to identify hosts requiring load redistribution.

4. *RintVMD*: Optimization engine invoked when StoLID detects imbalances. Comprises:

    - *Selection of Migrating VMs*: Selects VM candidates based on resource patterns and migration costs.
    - *Distribution of Migrating VMs*: Determines optimal destination hosts ensuring energy efficiency.

5. *VM Migration Orchestrator*: Translates placement matrix into migration commands and coordinates with VMMs.

The framework operates through systematic flow: *Local Agents* collect data and forward to *Global Agent*, where *StoLID* detects imbalances. *RintVMD* generates consolidation strategies, executed via *VM Migration Orchestrator*. This cycle adapts to workload changes while maintaining energy efficiency. The hierarchical architecture combines distributed monitoring with centralized optimization, providing data center visibility and scalability. The following sections examine RintVMD's components, as this module addresses VM selection and placement for optimal energy efficiency with controlled migration levels.

5.5 Selection of Migrating VMs with Minimal Resource Impact

In dynamic cloud environments, balancing system load with minimal VM migration is crucial for maintaining QoS. Migration can disrupt QoS, so minimizing the number

of migrating VMs is essential. Under-loaded hosts require migrating all VMs to enter sleep mode and conserve energy. Conversely, overloaded hosts benefit from selectively migrating a subset of VMs to balance the load and alleviate strain.

Algorithm `MNVMM(`t`)` systematically minimizes VM migrations while resolving load imbalances at time t. Consider a detection of system load imbalance at time instant t (i.e., $\mathcal{H}^t_{ovr} \cup \mathcal{H}^t_{undr} \neq \emptyset$) and $\mathcal{V}^t_{mig}$ to be the set of migrating VMs. Then, $\mathcal{V}^t_{mig}$ typically includes all VMs from hosts in $\mathcal{H}^t_{under}$ and a subset of VMs from hosts in $\mathcal{H}^t_{ovr}$. Therefore, minimizing migrations during VM consolidation focuses on reducing the number of migrating VMs from $\mathcal{H}^t_{ovr}$.

5.5.1 Prioritization of VMs for Migration

The algorithm introduces a weight-based mechanism for optimizing VM selection from overloaded hosts. Each resource on overloaded hosts is assigned a weight based on usage intensity to ensure migration decisions prioritize the most critical resources. Assume H_i to be overloaded at time t (i.e., $H_i \in \mathcal{H}^t_{ovr}$) and O^t_i to be the set of over-utilized resources of H_i. VMs selected for migration from H_i should exhibit higher resource consumption for all resources in O^t_i compared to non-overutilized resources. The weights for each resource $r_k \in \mathcal{R}$ in H_i at time t are:

$$\omega^t_{ir_k} = \begin{cases} \frac{1}{1-\mathcal{U}^t_{ir_k}}, & \text{for } r_k \in O^t_i \\ 1 - \mathcal{U}^t_{ir_k}, & \text{otherwise} \end{cases} \tag{5.10}$$

where $\omega^t_{ir_k}$ is the relative weight of resource r_k in H_i at time t. The weight assignment prioritizes over-utilized resources by assigning them relatively higher weights compared to non-overutilized ones. For any resource $r_k \in O^t_i$, the weight $\omega^t_{ir_k} > 1$ is always higher than $\omega^t_{ir_k} < 1$ for any resource $r_k \in \overline{O^t_i}$, ensuring finer distinctions within each utilization class based on relative usage intensity.

MNVMM(t): Selection of **M**inimal **N**umber of **VM**s for **M**igration at time t

Input: $\mathcal{H}^t_{ovr}$: set of overloaded hosts at time t, $\mathcal{H}^t_{undr}$: set of underloaded hosts at time t, $\mathcal{R}$: set of all resource types, θ_{over}: overload threshold
$\{\mathcal{V}^t_i \mid \forall H_i \in \mathcal{H}^t_{ovr} \cup \mathcal{H}^t_{undr}\}$: VM sets for each problematic host
O^t_i: set of over-utilized resources for host H_i at time t

Output: $\mathcal{V}^t_{mig}$: set of migrating VMs at time t

$\mathcal{V}^t_{mig} \leftarrow \emptyset$ // Initialize empty migration set
for *each* $H_i \in \mathcal{H}^t_{ovr}$ **do**
  $\forall r_k \in \mathcal{R} : \omega^t_{ir_k} \leftarrow WgtAssign(r_k)$
    // Weight assignment function
  $\mathbf{W} \leftarrow [\omega^t_{ir_k}]_{1\times \mathsf{d}}$ // Construct weight vector
  $\mathbf{D} \leftarrow [\mathsf{D}_{j,k}]_{\mathsf{n}^t_i \times \mathsf{d}}$ where $\mathsf{D}_{j,k} = D^t_{jr_k}$ // Build demand matrix
  for *each* $r_k \in \mathcal{R}$ **do**
    if $r_k \in O^t_i$ **then**
      $D^t_{\gamma r_k} \leftarrow \max_{V_j \in \mathcal{V}^t_i} D^t_{jr_k}$ // Max demand for over-utilized resources
    else
      $D^t_{\gamma r_k} \leftarrow \min_{V_j \in \mathcal{V}^t_i} D^t_{jr_k}$ // Min demand for non-overutilized resources
    end
  end
  $\mathbf{c} \leftarrow [D^t_{\gamma r_k}]_{1\times \mathsf{d}}$ // Create reference vector for ideal VM
  $\mathbf{S} \leftarrow [\mathsf{D}_{j,k} - \mathsf{c}_k]_{\mathsf{n}^t_i \times \mathsf{d}}$ // Compute similarity matrix
  $\mathbf{E} \leftarrow \sqrt{\mathbf{W} \cdot \widehat{\mathbf{S}}\widehat{\mathbf{S}}^{\mathsf{T}}}$ // Calculate weighted Euclidean distances
  do
    $\beta \leftarrow \text{argmin}_j \, \mathsf{E}_j$ // Select VM closest to ideal candidate
    $\mathcal{V}^t_{mig} \leftarrow \mathcal{V}^t_{mig} \cup \{V_\beta\}$ // Add to migration set
    $X^t_{i\beta} \leftarrow 0$ // Mark VM as migrated from host
    $\mathcal{V}^t_i \leftarrow \mathcal{V}^t_i \setminus \{V_\beta\}$ // Remove VM from host
    $\mathbf{E} \leftarrow [\mathsf{E_j} \mid j \in \{1, 2, \ldots, \mathsf{n}^t_i\} \setminus \{\beta\}]_{1\times|\mathcal{V}^t_i|}$ // Update decision matrix
  while $\exists r_k \in \mathcal{R} : Pr^t_{over}(H_i)^{r_k} > \theta_{over}$;
end
for *each* $H_i \in \mathcal{H}^t_{undr}$ **do**
  $\mathcal{V}^t_{mig} \leftarrow \mathcal{V}^t_{mig} \cup \mathcal{V}^t_i$ // Add all VMs to migration set
  $\forall V_j \in \mathcal{V}^t_i : X^t_{ij} \leftarrow 0$
end
return $\mathcal{V}^t_{mig}$ // Return final migration set

5.5.2 Three-Phase VM Selection Mechanism

The algorithm operates through three sequential phases. First, the algorithm initializes the migration set as empty (`line 1`) and processes each overloaded host $H_i \in \mathcal{H}^t_{ovr}$ (`lines 2-16`). Each overloaded host undergoes resource priority establishment through weight assignment based on Eq. 5.10 (`line 2`). Resources receive weights consolidated into a weight vector $\mathbf{W}$ of dimension $1 \times \mathsf{d}$, defined as $\mathbf{W} = [\mathsf{W}_k]_{1\times\mathsf{d}}$, where each element W_k represents the assigned weight for resource r_k on host H_i at time t (i.e., $\mathsf{W}_k = \omega^t_{ir_k}$) (`line 3`). A comprehensive demand matrix $\mathbf{D}$ of size $\mathsf{n}^t_i \times \mathsf{d}$ captures resource requirements of all VMs on each host, where $\mid \mathcal{V}^t_i \mid = \mathsf{n}^t_i$. The matrix is formulated as $\mathbf{D} = [\mathsf{D}_{j,k}]_{\mathsf{n}^t_i\times\mathsf{d}}$, where each element $\mathsf{D}_{j,k}$ represents the demand of VM $V_j \in \mathcal{V}^t_i$ for resource $r_k \in \mathcal{R}$ at time t (i.e., $\mathsf{D}_{j,k} = D^t_{jr_k}$) (`line 4`).

Second, the algorithm identifies an imaginary VM V_γ, representing optimal migration candidate characteristics from host H_i at time t (`lines 5-10`). This theoretical construct exhibits maximum demand for over-utilized resources and minimum demand for non-over-utilized resources. The resource demand profile of V_γ is defined as $V_\gamma = \left\langle D^t_{\gamma r_k} \mid \forall r_k \in \mathcal{R} \right\rangle$ where:

$$D^t_{\gamma r_k} = \begin{cases} \max\limits_{V_j \in \mathcal{V}^t_i} D^t_{jr_k}, & \text{if } r_k \in O^t_i \\ \min\limits_{V_j \in \mathcal{V}^t_i} D^t_{jr_k}, & \text{otherwise} \end{cases}$$

This ideal VM serves as a reference benchmark for evaluating actual VM migration suitability. The potentiality of each VM $V_j \in \mathcal{V}^t_i$ as a migration candidate is evaluated by comparing their similarity to V_γ using Euclidean distance. For any VM V_j,

the distance from the ideal VM V_γ is:

$$\zeta_{\gamma j} = \sqrt{\sum_{r_k \in \mathcal{R}} \omega^t_{ir_k} \left[D^t_{jr_k} - D^t_{\gamma r_k} \right]^2}$$

Third, the algorithm constructs a decision matrix $\mathbf{E} = [\mathsf{E}_j]_{1 \times \mathsf{n}^t_i}$ (`lines 11-13`), where each element E_j represents the distance metric for VM V_j in the candidate set $\mathcal{V}^t_i$:

$$\mathbf{E} = \sqrt{\mathbf{W} \cdot \widehat{\mathbf{S}} \widehat{\mathbf{S}}^\mathsf{T}} \tag{5.11}$$

where $\mathbf{S} = [\mathsf{S}_{j,k}]_{\mathsf{n}^t_i \times \mathsf{d}}$ is a similarity matrix of the same size as the demand matrix $\mathbf{D}$, and $\widehat{\mathbf{S}}$ is its normalized form. The similarity matrix $\mathbf{S}$ is derived by first forming a demand vector $\mathbf{c} = [\mathsf{c}_k]_{1 \times \mathsf{d}}$, where each element $\mathsf{c}_k = D^t_{\gamma r_k}$ represents the resource demand (`line 11`). Subsequently, each element of $\mathbf{S}$ is calculated as $\mathsf{S}_{j,k} = \mathsf{D}_{j,k} - \mathsf{c}_k$ (`line 12`). The normalization of matrix $\mathbf{S}$ ensures a comparable scale for all resources and prevents bias towards any particular resource type.

Each entry in $\mathbf{E}$ reflects the degree of appropriateness of VM V_j for migration from host H_i (`line 13`). A lower value indicates that the VM is close to the ideal candidate and represents a good choice for migration, while a higher value suggests that migrating that VM may not be necessary. The matrix $\mathbf{E}$ is used iteratively to select VMs for migration until host H_i becomes non-overloaded (`lines 14-19`), with the most suitable candidate V_β identified as $\beta = \text{argmin}_j \, \mathsf{E}_j$ (`line 15`). The selected VM is added to the migration set (`line 16`), marked as migrated (`line 17`), removed from the host (`line 18`), and the decision matrix is updated (`line 19`). This iterative process continues until the overload probability for all resources falls within the threshold.

For underloaded hosts, the algorithm performs straightforward complete VM evacuation (`lines 20-23`). The entire set of VMs from each host $H_i \in \mathcal{H}^t_{undr}$ is added to the migration set $\mathcal{V}^t_{mig}$ (`line 21`), enabling these hosts to be powered

down for energy conservation. This complete evacuation strategy maximizes energy savings by allowing underutilized physical resources to be deactivated. The process concludes by returning the consolidated migration set $\mathcal{V}_{mig}^{t}$ containing all selected VMs (`line 24`). The algorithm's design ensures QoS preservation by minimizing unnecessary migrations while effectively addressing both overload and underload scenarios.

5.6 Distribution of Migrating VMs

Following the selection of migrating VMs, the system addresses their distribution among destination hosts through careful host selection that meets VM requirements while enhancing data center energy efficiency. The distribution process encompasses two integrated components: *identifying hosts with sufficient cumulative resources to accommodate all VM requests* and *optimally distributing VMs among selected hosts*. This ensures resource utilization optimization without compromising quality of service satisfaction.

5.6.1 Determining the Optimal Set of Hosts

Algorithm `MNHPVM(`t`)` systematically determines the optimal destination host set $\mathcal{H}_{des}^{t}$ for VM placement from $\mathcal{V}_{mig}^{t}$ at time t while minimizing resource wastage and host activation. An essential parameter in determining $\mathcal{H}_{des}^{t}$ involves finding appropriate resource amounts that adequately serve migrating VMs while minimizing wastage across all `d` resource dimensions, guaranteeing minimal host count without causing system overloading.

However, host overloading probability depends on resource allocation patterns, which remain difficult to determine before actual placement occurs. Placing VMs based solely on host total capacity may result in substantial overloading risk. To address this challenge, the algorithm operates through a two-stage process where

the first stage estimates minimum required resources for successful VM placement, while the second stage selects appropriate hosts.

MNHPVM(t): Selection of **M**inimal **N**umber of **H**osts for the **P**lacement of **VM**s at time t.

Input: $\mathcal{V}^t_{mig}$: set of migrating VMs at time t, $\mathcal{R}$: set of all resource types
$\mathcal{L}^t_{req}$: set of minimum required resource amounts
$\mathcal{H}^t_{\mathsf{nor}}$: set of normally-loaded hosts at time t
$\mathcal{H}^t_{\mathsf{idl}}$: set of idle hosts at time t, θ_{over}: overload threshold

Output: $\mathcal{H}^t_{des}$: set of destination hosts for VM placement

Initialization: $\mathcal{H}^t_{mat} \leftarrow \emptyset$, $E_{dist} \leftarrow 0$, $\mathcal{H}^t_{des} \leftarrow \emptyset$,
$\forall r_k \in \mathcal{R} : D^t_{\alpha r_k} \leftarrow \left(\sum_{V_j \in \mathcal{V}^t_{mig}} D^t_{jr_k}\right)/mig^t$
// Initialize hosts sets and resource demands of V_α

for each $H_i \in \mathcal{H}^t_{\mathsf{nor}}$ **do**
  $\mathcal{H}^t_{des} \cup \{H_i\}$ // Add to destination candidate set
  $\forall r_k \in \mathcal{R} : L^t_{r_k}(H_i) \leftarrow +D^t_{\alpha r_k}$ // Simulate average VM load addition
  if $\forall r_k \in \mathcal{R} : Pr^t_{over}(H_i)^{r_k} \leq \theta_{over}$ **then**
    $\mathcal{H}^t_{mat} \cup \{H_i\}$ // Mark as feasible for VM placement
end
$\forall r_k \in \mathcal{R} : \mathcal{J}^t_{r_k} \leftarrow \mathcal{L}^t_{r_k} - \left[\sum_{H_i \in \mathcal{H}^t_{mat}} \left(C_{ir_k} - \mathcal{U}^t_{ir_k} \times \frac{C_{ir_k}}{100}\right)\right]$ // Calculate remaining resource requirements after feasible hosts
$\forall r_k \in \mathcal{R} : C_{\beta r_k} \leftarrow \mathcal{J}^t_{r_k}$ // Define ideal host capacity
if $\exists r_k \in \mathcal{R} : \mathcal{J}^t_{r_k} > 0$ **then**
  for each $H_i \in \mathcal{H}^t_{\mathsf{idl}}$ **do**
    $\xi_{(i,\beta)} = \sqrt{\sum_{r_k \in \mathcal{R}} [C_{ir_k} - C_{\beta r_k}]^2}$ // Euclidean distance calculation
    E_{dist}.append($\xi_{(i,\beta)}$) // Store distance for sorting
  end
  $\mathcal{H}^t_{sorted} \leftarrow$ Descending.Sort $\left(\mathcal{H}^t_{\mathsf{idl}}, E_{dist}\right)$ // Sort by similarity distance
  for each $H_i \in \mathcal{H}^t_{sorted}$ **do**
    $\mathcal{H}^t_{des} \cup \{H_i\}$ // Add host to destination set
    $\mathcal{H}^t_{\mathsf{idl}} \setminus \{H_i\}$ // Remove from idle host pool
    if $\forall r_k \in \mathcal{R} : (\mathcal{J}^t_{r_k} - C_{ir_k}) < 0$ **then**
      break // All resource requirements satisfied
    else
      $\mathcal{J}^t_{r_k} \leftarrow (\mathcal{J}^t_{r_k} - C_{ir_k})$ // Update remaining requirements
    end
  end
$\mathcal{H}^t_{\mathsf{idl}} \cdot$ Sleep() // Power down unused idle hosts for energy conservation
return $\mathcal{H}^t_{des}$

The resource estimation approach determines minimum resource capacity satisfying aggregated demand of all migrating VMs without disrupting system load stability. The process operates on an imaginary host $H_\alpha = \langle C_{ar_k} \mid \forall r_k \in \mathcal{R} \rangle$ along with resource set $\mathcal{R}$ and migrating VM set $\mathcal{V}^t_{mig}$, producing $\mathcal{L}^t_{req}$ as the set of minimum required resource amounts capable of accommodating all VMs in $\mathcal{V}^t_{mig}$ without system overloading.

Initially, H_α capacity for each resource r_k is set as the cumulative demand of migrating VMs at time t: $C_{\alpha r_k} = \sum_{V_j \in \mathcal{V}^t_{mig}} D^t_{jr_k}$. The system then iterates over each resource $r_k \in \mathcal{R}$ to compute its corresponding least required amount, denoted by $\mathcal{L}^t_{r_k}$. Given the overloading definition, H_α initially experiences overloading. The approach exploits this condition by iteratively estimating $\Pr^t_{\text{over}}(H_\alpha)^{r_k}$ through incremental additions of ∇_{r_k} to $C_{\alpha r_k}$ until H_α achieves non-overloaded status. The final capacity value after iteration becomes $\mathcal{L}^t_{r_k}$. The increment ∇_{r_k} determines the amount by which $C_{\alpha r_k}$ increases: $\nabla_{r_k} = \partial \cdot C_{\alpha r_k}$ where $\partial \in (0, 1)$ controls the increment rate. Higher ∂ values lead to faster convergence toward non-overloaded states but may overestimate required resources, while lower values result in slower convergence but more optimal resource selection. Overloading probability $\Pr^t_{\text{over}}(H_\alpha)^{r_k}$ estimation occurs based on updated $C_{\alpha r_k}$ values. The process continues until $\Pr^t_{\text{over}}(H_\alpha)^{r_k}$ becomes smaller than or equal to over-utilization threshold θ_{over}. At iteration end, $C_{\alpha r_k}$ is assigned as the least required amount $\mathcal{L}^t_{r_k}$, which is added to set $\mathcal{L}^t_{req}$. When $\mathcal{L}^t_{r_k}$ has been calculated for all $r_k \in \mathcal{R}$, the complete set $\mathcal{L}^t_{req}$ becomes available for subsequent host selection.

Once required resource volume is established, the second stage identifies host sets with sufficient cumulative capacity to accommodate that volume with minimal resource wastage. The migration process considers all active normally-loaded hosts as potential destinations. The set of normally-loaded hosts at time t is defined as $\mathcal{H}^t_{\mathsf{nor}} = \mathcal{H}^t_{active} \cap \big(\overline{\mathcal{H}^t_{ovr} \cup \mathcal{H}^t_{undr}}\big)$. However, during VM migration selection, VMs from

both overloaded and underloaded hosts are removed, changing their load status. Overloaded hosts become normally-loaded while underloaded hosts become idle, resulting in $\mathcal{H}^t_{\mathsf{nor}} = \mathcal{H}^t_{\mathsf{nor}} \cup \mathcal{H}^t_{ovr}$. The set of idle hosts at time t is denoted as $\mathcal{H}^t_{\mathsf{idl}} = \mathcal{H}^t_{undr}$.

The algorithm begins by computing average resource demands of migrating VMs to create an imaginary representative VM V_α and initializing essential data structures. The first stage evaluates normally-loaded hosts for placement feasibility (`lines 1-6`). Each host $H_i \in \mathcal{H}^t_{\mathsf{nor}}$ is evaluated on its ability to accommodate imaginary VM V_α alongside VMs currently running on it at time t (`lines 1-2`). The imaginary VM V_α has resource demands equal to the mean of all migrating VM requirements for each respective resource. Given the assumption of normal resource demand distribution, the mean serves as a representative value capturing the collective tendency of VM resource requirements. The demand of V_α for resource r_k at time t, denoted as $D^t_{\alpha r_k}$, is determined as $D^t_{\alpha r_k} = \frac{\sum_{V_j \in \mathcal{V}^t_{mig}} D^t_{jr_k}}{mig^t}$, where $\mid \mathcal{V}^t_{mig} \mid = mig^t$.

Each host undergoes load simulation by adding the average VM resource demands (`line 3`), and overloading probability assessment determines whether the host can accommodate additional workload without exceeding threshold θ_{over} across all resource dimensions (`lines 4-5`). If the overloading probability of H_i at time t falls within θ_{over} for all resources, then H_i is considered feasible and added to the set $\mathcal{H}^t_{mat}$, representing the collection of feasible hosts at time t (`line 5`).

These feasible hosts determine the remaining amount of $\mathcal{L}^t_{r_k}$ for all $r_k \in \mathcal{R}$ that require satisfaction. The algorithm then computes remaining resource requirements after accounting for available capacity from feasible hosts (`line 7`). The remaining amount for resource r_k, denoted by $\mathcal{J}^t_{r_k}$, is calculated using:

$$\mathcal{J}^t_{r_k} = \mathcal{L}^t_{r_k} - \left[\sum_{H_i \in \mathcal{H}^t_{mat}} \left(C_{ir_k} - \mathcal{U}^t_{ir_k} \times \frac{C_{ir_k}}{100} \right) \right] \tag{5.12}$$

If any resource $r_k \in \mathcal{R}$ has remaining requirements, the algorithm characterizes an ideal host H_β with capacity matching these remaining needs (`line 8`). To optimize selection in terms of resource wastage, the ideal host H_β is defined with capacity for each resource equal to the remaining required capacity for corresponding resources, setting $\mathcal{J}^t_{r_k}$ as $C_{\beta r_k}$ for all $r_k \in \mathcal{R}$.

If additional resources are required, the second stage processes idle hosts through distance-based optimization (`lines 9-23`). The approach aligns idle hosts based on their resource capacities relative to corresponding remaining required resources. Euclidean distance computation quantifies similarity between each idle host's resource distribution and the ideal configuration (`lines 10-13`), providing mathematically rigorous assessment of similarity between multi-dimensional vectors. The distance from each host $H_i \in \mathcal{H}^t_{\mathsf{idl}}$ to H_β, denoted as $\xi_{(i,\beta)}$, is calculated as:

$$\xi_{(i,\beta)} = \sqrt{\sum_{r_k \in \mathcal{R}} [C_{ir_k} - C_{\beta r_k}]^2} \tag{5.13}$$

This measurement assesses how closely proportional capacities of H_i align with those of H_β across all resources. Smaller $\xi_{(i,\beta)}$ values indicate greater similarity between capacity distributions. A distance list E_{dist} stores distances of all idle hosts (`line 12`), followed by descending-order sorting to prioritize hosts with optimal resource alignment (`line 14`). The algorithm iteratively selects sorted idle hosts and updates remaining requirements until all resource needs are satisfied (`lines 15-23`). The host set $\mathcal{H}^t_{\mathsf{idl}}$ is arranged in ascending order based on distances from H_β and stored in $\mathcal{H}^t_{sorted}$ (`line 14`). The process iterates over each host $H_i \in \mathcal{H}^t_{sorted}$ and adds them to set $\mathcal{H}^t_{des}$ until all required resources are satisfied (`lines 15-23`).

The process concludes by powering down unused idle hosts for energy conservation (`line 24`) and returns the optimized destination host set $\mathcal{H}^t_{des}$ (`line 25`). This design ensures minimal host activation while maintaining load balance and

preventing system overutilization.

5.6.2 Distribution of VMs among the hosts

Having identified the optimal set of destination hosts, the next phase involves determining precise allocation of migrating VMs across these hosts, strategically balancing resource utilization efficiency with service quality requirements.

Cloud service providers and cloud users have competing goals: energy efficiency versus quality of service satisfaction. Both rely on the same resource pool but utilize resources differently. Uneven resource utilization fails to meet either party's objectives. The favorable condition requires maximal and fairly balanced resource utilization through interaction between providers and users to jointly determine optimal distribution strategies. This aligns with the strategic game concept from game theory, leading to a VM placement scheme that distributes VMs in $\mathcal{V}^t_{mig}$ onto hosts $\mathcal{H}^t_{des}$.

Game theory analyzes competitive scenarios where multiple players make decisions that collectively determine outcomes. Each player possesses actions and a payoff function determining rewards based on all players' actions, with the goal of maximizing payoff. In VM placement context, the preferable outcome involves deployed hosts having minimal resource wastage while satisfying VM resource demands. This requires maximizing utilization of all resources in a balanced manner, establishing balance between cloud service provider and cloud user requirements.

5.6.2.1 Game Formulation for VM Distribution

Algorithm `GBVMP(`t`)` employs game-theoretic principles to optimize VM distribution across destination hosts. The game design considers migrating VMs as players and assignments on each selected host as their probable behaviors or strategies. This consideration stems from complexity reduction in determining player strategies and

payoffs. Moreover, interaction between cloud service providers and cloud users occurs solely through VMs. Therefore, establishing VMs as players not only simplifies strategy selection by choosing identical strategy sets across all players, but also aids in establishing common payoffs reflecting both cloud service provider and cloud user purposes.

VM placement at the t^{th} time instant can be described as an n-player finite strategic game $\mathcal{G} = \langle \mathcal{P}, \mathcal{S}, \mathcal{F} \rangle$ (assuming $\mid \mathcal{V}_{mig}^{t} \mid = n$), where:

- $\mathcal{P} = \{V_j \mid \forall V_j \in \mathcal{V}_{mig}^{t}\}$ is a set of n players.
- $\mathcal{S} = \prod_{V_j \in \mathcal{V}_{mig}^{t}} S_j$ is the set of strategy profiles, which is the Cartesian product of all the player's strategy set.
- $\mathcal{F} = \{F_j(\mathcal{Q}) \mid \forall V_j \in \mathcal{V}_{mig}^{t}\}$ is the set of payoff functions, where $\mathcal{Q} \in \mathcal{S}$.

Here, S_j represents the strategy set of player V_j, such that $S_j \subseteq \mathcal{H}_{sel}^{t}$. Let $\mathcal{Q} = (Q_j \mid \forall V_j \in \mathcal{V}_{mig}^{t})$ be a strategy profile, where Q_j is a specific strategy of player V_j, namely $Q_j = H_k \in \mathcal{H}_{des}^{t}$. For each player V_j, strategy preference, known as the payoff, is determined based on the specific strategy profile $\mathcal{Q}$, including the strategy chosen by the player as well as strategies chosen by all other players. Hence, the payoff is obtained for player V_j as $F_j(\mathcal{Q})$.

Each strategy profile $\mathcal{Q}$ represents a possible placement of VMs in $\mathcal{V}_{mig}^{t}$ onto any hosts in $\mathcal{H}_{des}^{t}$, where VMs make host selections simultaneously and independently. The payoff of each player for $\mathcal{Q}$ represents the utility of the player's strategies toward VM placement, with the purpose being to find a strategy profile that maximizes overall payoff. The strategy profile for which no VMs can improve their corresponding payoffs by changing hosts or strategies constitutes the optimal placement scheme. This implies that the solution must be a Nash equilibrium—a strategy combination with the property that no player can gain by unilaterally deviating from it. The

payoff function for each player V_j is defined as:

$$F_j(\mathcal{Q}) = \Psi_j^t.\varphi_{\mathcal{Q}}^t \tag{5.14}$$

where $\varphi_{\mathcal{Q}}^t$ represents the payoff generated for strategy profile Q at time t and Ψ_j^t is the average resource demand of V_j at time t. Specifically, $\Psi_j^t = \left[\left(\sum_{r_k \in \mathcal{R}} \widehat{D_{jr_k}^t}\right) \Big/ \mathsf{d}\right]$, where $\widehat{D_{jr_k}^t}$ is the normalized demand of V_j for resource r_k at time t.

In Equation 5.14, Ψ_j^t denotes the proportion of $\varphi_{\mathcal{Q}}^t$ that player V_j receives as individual payoff. Since the value of Ψ_j^t for all $V_j \in \mathcal{V}^t$ remains constant irrespective of the chosen strategy profile, the highest $F_j(\mathcal{Q})$ is obtained only for the strategy profile having maximum $\varphi_{\mathcal{Q}}^t$. Using $\mathcal{Q}^{max}$ to specify that profile:

$$\mathcal{Q}^{max} = \underset{\mathcal{Q} \in \mathcal{S}}{\operatorname{argmax}}\, \varphi_{\mathcal{Q}}^t \tag{5.15}$$

This clearly implies that strategy profile $\mathcal{Q}^{max} = (Q_1^{max}, Q_2^{max}, \ldots, Q_n^{max})$ yields the equilibrium state as it satisfies the condition:

$$\forall V_j \in \mathcal{P} : F(Q_j^{max}, \mathcal{Q}_{-j}^{max}) \geq F(Q_j, \mathcal{Q}_{-j}^{max})$$

where, for strategy profile $\mathcal{Q}^{max}$, $\mathcal{Q}_{-j}^{max}$ represents the strategies of all players in $\mathcal{P}$ except player V_j.

The algorithm systematically evaluates all strategy profiles in $\mathcal{S}$ through iterative preference factor analysis (`lines 1-23`). For each strategy profile $\mathcal{Q}$, the placement matrix is established by mapping VMs to hosts according to current strategy assignments (`lines 2-3`). The approach evaluates each strategy profile $\mathcal{Q} \in \mathcal{S}$ and

GBVMP(t): Game-**B**ased **V**irtual **M**achine **P**lacement at time t.

Input: $\mathcal{P} = \{V_j \mid \forall V_j \in \mathcal{V}^t_{mig}\}$: set of migrating VMs as players, $\mathcal{H}^t_{des}$: set of destination hosts
$\mathcal{S} = \prod_{V_j \in \mathcal{V}^t_{mig}} S_j$: set of strategy profiles, $\mathcal{R}$: set of all resource types
θ_{over}: overload threshold, $\mathtt{d}$: number of resource dimensions

Output: $\mathcal{X}^t$: placement matrix defining VM-to-host assignments at time t

Initialization: $\varphi^t_{best} \leftarrow -\infty$, $\mathcal{Q}^{max} \leftarrow \emptyset$, $m \leftarrow |\mathcal{H}^t_{des}|$
`// Initialize best payoff tracker and destination host count`

for each $\mathcal{Q} \in \mathcal{S}$ **do**
  $\forall Q_j \in \mathcal{Q}, \forall H_i \in \mathcal{H}^t_{des}, \forall V_j \in \mathcal{V}^t_{mig} : X^t_{ij} \leftarrow 0$ `// Reset placement matrix`
  $\forall Q_j \in \mathcal{Q} : X^t_{kj} \leftarrow 1$, where $Q_j = H_k \in \mathcal{H}^t_{des}$ `// Set VM-host assignments per strategy`
  $\forall H_i \in \mathcal{H}^t_{des} : PF^t_i \leftarrow 0$ `// Initialize preference factors`
  $\Omega^t_{\mathcal{Q}} \leftarrow 1$ `// Initialize feasibility indicator`
  for each host $H_i \in \mathcal{H}^t_{des}$ **do**
    if $\forall r_k \in \mathcal{R} : Pr^t_{over}(H_i)^{r_k}(H_i) \leq \theta_{over}$ **then**
      $r_m \leftarrow \text{argmax}_{r_k \in \mathcal{R}} \mathcal{U}^t_{ir_k}$ `// Identify maximally utilized resource`
      $\Gamma^t_i \leftarrow \sum_{r_k \in \mathcal{R}} \mathcal{U}^t_{ir_k}$ `// Compute total resource utilization`
      $\Delta^t_i \leftarrow \sum_{r_k \in \mathcal{R} \setminus \{r_m\}} \left[\mathcal{U}^t_{ir_m} - \mathcal{U}^t_{ir_k}\right]$ `// Calculate utilization gap`
      $PF^t_i \leftarrow \Gamma^t_i - \frac{\Delta^t_i}{\mathtt{d}}$ `// Compute preference factor for balanced utilization`
    else
      $\Omega^t_{\mathcal{Q}} \leftarrow 0$ `// Mark strategy as infeasible due to overloading`
      break
    end
  end
  $\mu^t_{PF} \leftarrow \frac{1}{m} \sum_{H_i \in \mathcal{H}^t_{des}} PF^t_i$ `// Calculate mean preference factor`
  $\sigma^t_{PF} \leftarrow \sqrt{\frac{1}{m} \sum_{H_i \in \mathcal{H}^t_{des}} \left(PF^t_i - \mu^t_{PF}\right)^2}$ `// Compute standard deviation for tie-breaking`
  $\varphi^t_{\mathcal{Q}} \leftarrow \left[\sum_{H_i \in \mathcal{H}^t_{des}} PF^t_i - \sigma^t_{PF}\right] \times \Omega^t_{\mathcal{Q}}$ `// Calculate strategy profile payoff`
  if $\varphi^t_{\mathcal{Q}} > \varphi^t_{best}$ **then**
    $\varphi^t_{best} \leftarrow \varphi^t_{\mathcal{Q}}$ `// Update best payoff`
    $\mathcal{Q}^{max} \leftarrow \mathcal{Q}$ `// Store optimal strategy profile`
end
$\forall H_i \in \mathcal{H}^t_{des}, \forall V_j \in \mathcal{V}^t_{mig} : X^t_{ij} \leftarrow 0$ `// Reset final placement matrix`
$\forall Q_j \in \mathcal{Q}^{max} : X^t_{kj} \leftarrow 1$, where $Q_j = H_k \in \mathcal{H}^t_{des}$ `// Apply optimal VM placement`
return $\mathcal{X}^t$

calculates its corresponding payoff $\varphi^t_{\mathcal{Q}}$ using:

$$\varphi^t_{\mathcal{Q}} = \left[\sum_{H_i \in \mathcal{H}^t_{des}} PF^t_i - \sigma^t_{PF}\right] \times \Omega^t_{\mathcal{Q}} \tag{5.16}$$

where PF_i^t is the preference factor of H_i at time t, illustrating how effectively resources are utilized in H_i. Host-level evaluation proceeds through feasibility verification and preference factor computation (`lines 6-16`). The preference factor is obtained as:

$$PF_i^t = \left[\Gamma_i^t - \frac{\Delta_i^t}{\mathsf{d}}\right] \tag{5.17}$$

In this equation, Γ_i^t represents the total utilization of all resources of H_i at time t, namely $\Gamma_i^t = \sum_{r_k \in \mathcal{R}} \mathcal{U}_{ir_k}^t$ (`line 9`). Higher values of Γ_i^t indicate greater resource utilization in H_i, though this doesn't specify individual resource utilization patterns. Therefore, Δ_i^t, termed the utilization gap of H_i at time t, has been introduced to quantify individual resource utilization (`line 10`). For all resources $r_k \in \mathcal{R}$, Δ_i^t is calculated as:

$$\Delta_i^t = \sum_{r_k \in \mathcal{R} - \{r_m\}} \left[\mathcal{U}_{r_m}^t - \mathcal{U}_{ir_k}^t\right] \tag{5.18}$$

Here, $r_m \in \mathcal{R}$ represents the maximally utilized resource in H_i at time t, namely $r_m = \underset{r_k}{\operatorname{argmax}} \mathcal{U}_{ir_k}^t$ (`line 8`). This equation sums the differences in utilization of all resources from that of their maximally utilized counterpart. Smaller values of Δ_i^t indicate better balance between utilization across different resources in H_i. The subtraction of $\frac{\Delta_i^t}{\mathsf{d}}$ from Γ_i^t in Equation 5.17 demonstrates how efficiently the resources of H_i have been utilized.

The division of Δ_i^t by the number of resources serves two specific purposes. First, without this division, strategy profiles may exist where hosts $H_i \in \mathcal{H}_{des}^t$ with smaller utilization gaps Δ_i^t could generate higher preference factors PF_i^t even if total resource utilization Γ_i^t is quite low. In such cases, Γ_i^t should be prioritized over Δ_i^t. Second, for some strategy profiles, if resource utilization is much less compared to the maximally utilized resource, simply computing $[\Gamma_i^t - \Delta_i^t]$ may produce negative values for PF_i^t, adversely impacting the decision-making process. Thus, dividing Δ_i^t by d not only reduces the importance of Δ_i^t over Γ_i^t in certain cases, but ensures Δ_i^t

remains smaller than Γ_i^t. After simplifying Equation 5.17, PF_i^t becomes (`line 11`):

$$PF_i^t = \sum_{r_k \in \mathcal{R}} \mathcal{U}_{ir_k}^t - \frac{1}{\mathsf{d}} \cdot \sum_{r_k \in \mathcal{R} - \{r_m\}} \left[\mathcal{U}_{ir_m}^t - \mathcal{U}_{ir_k}^t\right] \tag{5.19}$$

Successfully validated hosts undergo resource utilization assessment, identifying the maximally utilized resource r_m and calculating total utilization Γ_i^t alongside utilization gap Δ_i^t (`lines 8-10`). The preference factor PF_i^t integrates both metrics to reward high utilization while penalizing resource imbalance (`line 11`). Statistical analysis computes mean and standard deviation of preference factors across destination hosts (`lines 17-18`). The approach computes the preference factor PF_i^t for each host $H_i \in \mathcal{H}_{des}^t$, then calculates the mean μ_{PF}^t of all preference factors, which is subsequently used to determine the preference factor's standard deviation σ_{PF}^t (`line 18`). If m denotes the number of hosts in $\mathcal{H}_{sel}^t$, namely $m = \mid \mathcal{H}_{sel}^t \mid$, then σ_{PF}^t is calculated as:

$$\sigma_{PF}^t = \sqrt{\frac{1}{m} \cdot \sum_{H_i \in \mathcal{H}_{sel}^t} \left(PF_i^t - \mu_{PF}^t\right)^2}$$

The payoff function combines preference factor summation with standard deviation subtraction (`line 19`), where σ_{PF}^t serves as a tie-breaking mechanism for consistent utilization patterns. According to Equation 5.16, σ_{PF}^t is subtracted from the summation of preference factors to indicate the preferred $\mathcal{Q}$ among all profiles in $\mathcal{S}$ where the summation of preference factors generates identical values. In such cases, σ_{PF}^t functions as a tie-breaker by measuring variation across host preference factors, with the strategy profile having the lowest standard deviation selected as the preferred choice. The feasibility indicator $\Omega_{\mathcal{Q}}^t$ ensures only valid configurations contribute to optimization (`lines 12-14`). Continuous comparison identifies maximum payoff

TABLE 5.1: Configuration of selected physical machines or hosts.

Server	**Cores**	**MIPS/core**	**RAM (GB)**	**Storage (GB)**
Hitachi HA8000/TS10 (FK1)	4	3067	8	250
HP ProLiant DL580 G3	8	2660	16	294
Fujitsu PRIMERGY RX2540 M1	36	2300	64	64
IBM System x3630 M3	12	3067	12	160
Dell PowerEdge R7425	64	2200	128	120

strategy profiles representing Nash equilibrium solutions (`lines 20-22`). The algorithm concludes by implementing the optimal strategy profile through final placement matrix configuration (`lines 24-25`), ensuring balanced resource utilization while maintaining system stability and preventing host overloading.

5.7 Performance Evaluation

This section presents a comprehensive series of experiments conducted using real-world workloads to evaluate the performance of the proposed RO-VMC framework against state-of-the-art approaches. The evaluation begins with an overview of the experimental setup and metrics employed for assessment, followed by presentation of experimental results and detailed analysis comparing the proposed approach with existing methods.

5.7.1 Experimental Setup

Performance evaluation of the RO-VMC framework requires comprehensive experimentation across diverse scenarios. Given the challenges of conducting large-scale experiments in real cloud environments, a simulated environment has been developed that accurately replicates data center characteristics and operational dynamics. The experimental framework encompasses three key components: infrastructure configuration, workload characterization, and parameter optimization.

TABLE 5.2: Configuration of selected VMs.

Type	Cores	MIPS/core	Mem (MB)	Storage (MB)
g2.m	1	2300	1024	3000
g2.xl	4	2300	4096	3000
m.l	2	2300	7808	3000
c1.2xl	8	2800	7680	3000
c2.xl	4	2900	3840	3000

TABLE 5.3: Statistics of selected workload's resource usage parameters for Bitbrains and GCC.

Trace	No. of workloads	CPU		Memory		Disk	
		Mean(%)	St.dev.(%)	Mean(%)	St.dev.(%)	Mean(%)	St.dev.(%)
Bitbrains	10000	8.95	4.81	9.57	3.93	3.61	1.74
GCC	200000	2.81	0.86	2.34	0.19	1.62	0.52

TABLE 5.4: Power consumption (in Watts) of the hosts for different CPU utilizations.

Server	0%	20%	40%	60%	80%	100%
Hitachi HA8000/TS10(FK1)	43.5	58.4	70.0	84.5	107	124
HP ProLiant DL580 G3	520	587	705	766	803	833
Fujitsu PRIMERGY RX2540 M1	39	98.2	128	161	216	271
IBM system x3630 M3	80.2	133	155	182	217	251
Dell PowerEdge R7425	84.9	154	182	214	253	287

5.7.1.1 Infrastructure Configuration

The simulation framework operates in a 3-resource environment encompassing CPU, memory, and disk resources, implemented using Python 3.0 on a system with Intel(R) Core(TM) i5-9400f CPU @ 2.90 GHz, 16 GB RAM, and Windows 10 64-bit OS. To replicate data center heterogeneity, the setup incorporates 1000 physical machines across 5 different models (200 hosts per model) and 5 VM types corresponding to Amazon EC2 instance types. Detailed configurations are provided in Tables 5.1 and 5.2, respectively. It directly states that VM configurations remain fixed throughout the experiments and are not altered during the consolidation process.

5.7.1.2 Workload Description

Two distinct real-world traces validate RO-VMC effectiveness across diverse scenarios: Bitbrains [120] and Google Cloud Cluster (GCC) [19]. The Bitbrains trace represents business-critical workloads with applications running on the same VM for extended periods, capturing CPU, memory, and disk utilization every 5 minutes for approximately 1200 VMs daily over a month. The GCC trace tracks resource usage for 25 million tasks over 29 days in May 2011, with day 18 selected as representative following [135], from which 200,000 tasks were randomly selected.

For manageable experimentation, 1000 workloads were randomly selected daily for 10 consecutive days from Bitbrains, while the GCC selection was distributed to generate equal VM type quantities (40,000 VMs per type for GCC and 2,000 per type for Bitbrains). Resource utilization statistics for selected workloads are presented in Table 5.3.

5.7.1.3 Migration Modeling and Experimental Parameters

VM migration simulation employs the base model from [136], incorporating dynamic dirty page changes to estimate network traffic and migration time. Two key parameters are defined: dirty page memory rate (DR) following normal distribution DR $\sim \mathcal{N}(0.4, 0.2)$ MB/s, and memory transmission rate (TR) (TR = DR + 100 MB/s), where 100 MB/s represents dedicated migration bandwidth.

Experimental procedures include FIFO queuing for incoming VM requests with MBFD for initial placement. Parameter values are set as $\theta_{over} = \theta_{under} = 0.80$, $\tau = 0.3$, and $\partial = 0.20$. Data center resource states are updated every 5 minutes, recording host utilization status, overloaded/underloaded host counts, and migration statistics during each interval.

5.8 Experimental Results

This section presents experimental findings evaluating RO-VMC performance against three existing approaches: PEAS [95], AFED-EF [79], and EQ-VMC [86]. PEAS optimizes only CPU usage for energy efficiency, potentially leading to suboptimal consolidation decisions, while RO-VMC employs multi-criteria optimization prioritizing host resource usage intensities. AFED-EF equally weights energy and SLA factors without adapting to dynamic workloads, whereas RO-VMC achieves balanced resource utilization through game theory-based distribution. EQ-VMC prioritizes resource reservation over maximizing usage, reducing energy savings opportunities, unlike RO-VMC's intensity-aware optimization approach.

The experiments evaluate RO-VMC against these approaches using diverse real-world workload traces, presenting comparative analysis of energy optimization, load imbalance mitigation, QoS enhancement, and resource utilization improvement in dynamic cloud environments.

5.8.1 Energy Efficiency Analysis

Energy efficiency evaluation focuses on active host counts and data center energy consumption, as these metrics are directly correlated. Host energy consumption is estimated using the power model in Eq. 5.6, calculating power based on CPU utilization with coefficient values from SPECpower benchmark data (Table 5.4).

Figures 5.2, 5.3 and 5.4, 5.5 illustrate performance comparisons across both traces for active hosts and energy consumption, with data collected hourly over 24-hour simulation periods. RO-VMC consistently outperforms all alternatives. For the Bitbrains trace (Figure 5.2), RO-VMC maintains 18.6 active hosts per hour compared to 23.8, 28.7, and 35.3 for PEAS, AFED-EF, and EQ-VMC respectively, representing reductions of 21%, 35%, and 47%. Similarly, for GCC trace (Figure 5.3), RO-VMC achieves 6.1 active hosts per hour, reducing active hosts by 40%, 52%,

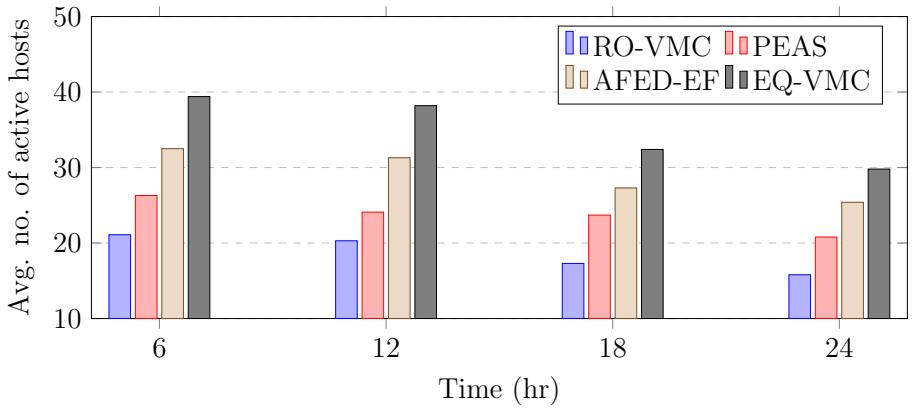

FIGURE 5.2: Number of active hosts over time using Bitbrains trace

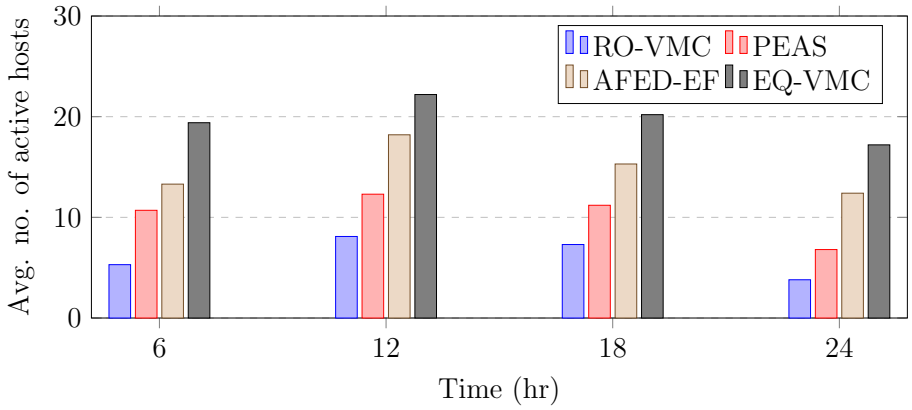

FIGURE 5.3: Number of active hosts over time using GCC trace

and 65% compared to the other approaches. Energy consumption patterns align with these trends. For Bitbrains trace (Figure 5.4), RO-VMC achieves 27.2 kW per hour, which is 17%, 26%, and 36% lower than PEAS, AFED-EF, and EQ-VMC respectively. For GCC trace (Figure 5.5), RO-VMC consumes 14.6 kW per hour, outperforming alternatives by 25%, 34%, and 46%.

PEAS emerges as the second most energy-efficient approach due to its CPU optimization focus, which significantly influences host power consumption. However,

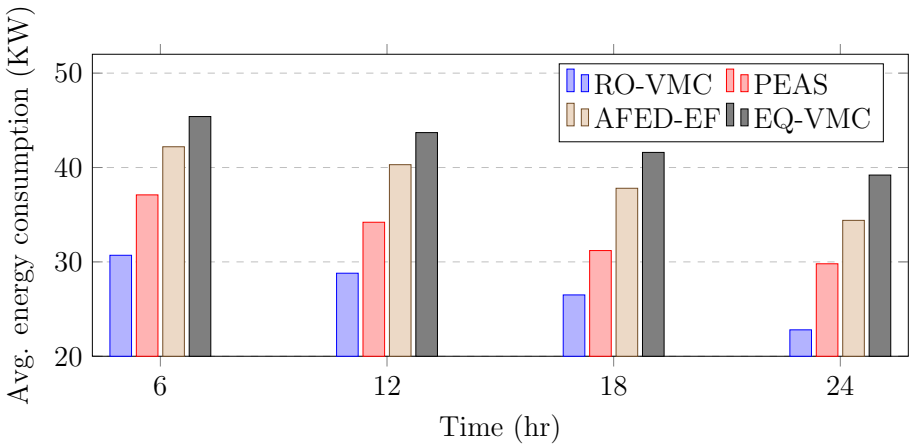

FIGURE 5.4: Average energy consumption over the simulation period using Bitbrains trace

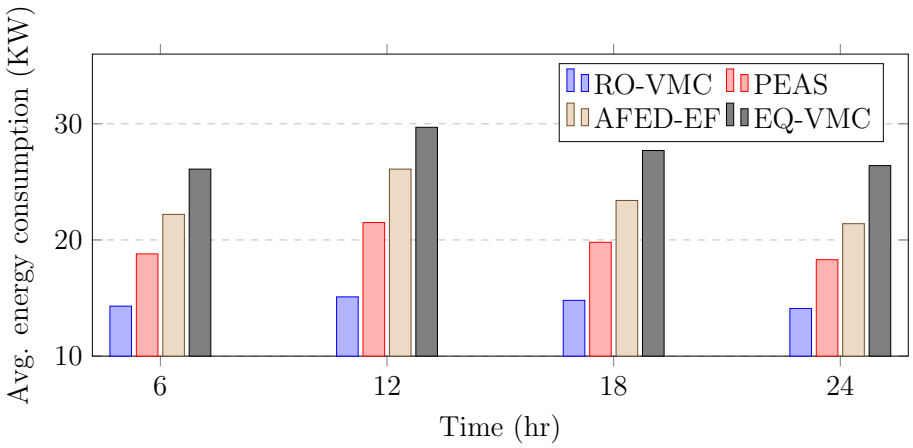

FIGURE 5.5: Average energy consumption over the simulation period using GCC trace

its inability to manage multiple resources simultaneously leads to suboptimal placements. AFED-EF struggles with rapidly fluctuating workloads, hindering its adaptability. EQ-VMC performs worst due to prioritizing resource reservation over utilization maximization. RO-VMC's superiority stems from effective multi-dimensional resource utilization, proving to be the most energy-efficient consolidation approach. This observation is further supported by the results summarized in Table 5.5, which reports the average number of active hosts per hour and the average hourly energy

TABLE 5.5: Summary of average active hosts per hour and average energy consumption per hour across Bitbrains and GCC traces

Approach	**Avg. Active Hosts/hr**		**Avg. Energy (kW/hr)**	
	Bitbrains	GCC	Bitbrains	GCC
RO-VMC	18.62	6.12	27.2	14.57
PEAS	23.72	10.25	33.07	19.60
AFED-EF	29.12	14.8	38.67	23.27
EQ-VMC	34.95	19.75	42.47	27.47

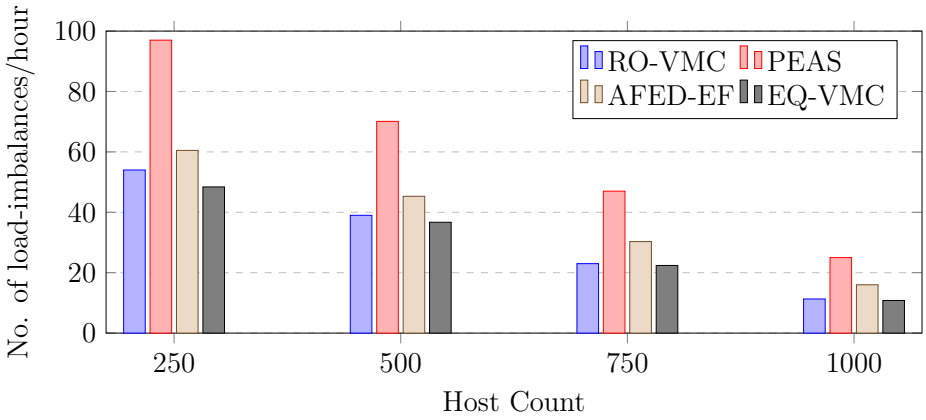

FIGURE 5.6: Number of load imbalances per hour on varying host counts using Bitbrains trace

consumption for all approaches across both workload traces.

5.8.2 Host Count Impact on Load Imbalances

Load imbalance evaluation under varying resource availability uses 105,000 VMs (5,000 from Bitbrains and 100,000 from GCC) across 24-hour simulations with host counts increasing from 250 to 1000 in increments of 250, maintaining equal distribution of each host type.

Figures 5.6 and 5.7 demonstrate imbalance occurrence patterns as host count increases. RO-VMC consistently generates lowest imbalance frequency on GCC trace across all host levels, exhibiting 37-70% fewer imbalances than alternatives. On Bitbrains trace, RO-VMC slightly lags behind EQ-VMC by 5% but outperforms other

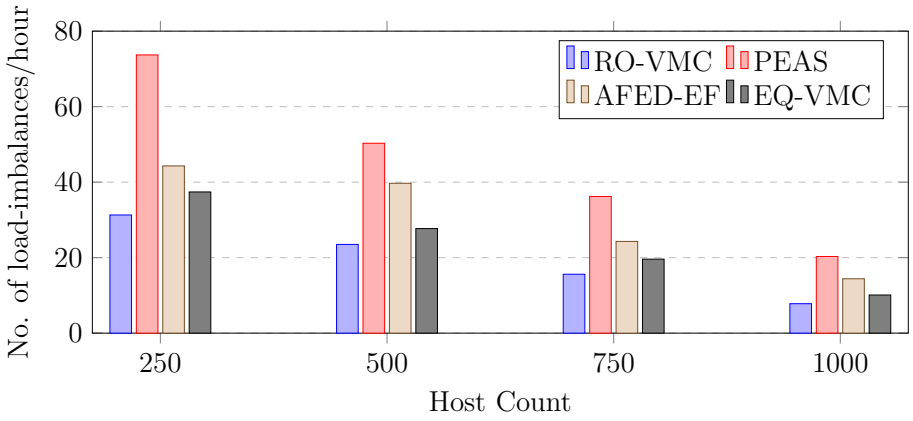

FIGURE 5.7: Number of load imbalances per hour on varying host counts using GCC trace

methods by 20-65%. PEAS exhibits volatile load imbalance patterns. For CPU-skewed workloads like GCC, PEAS's CPU consolidation reduces CPU imbalances but overloads memory and disk resources on densely packed hosts. This ineffectiveness is exacerbated for balanced multi-resource demands like Bitbrains, where CPU-centric optimization causes underutilization and multi-resource imbalances.

AFED-EF and EQ-VMC perform better but cannot match RO-VMC's consistent performance on GCC trace. On Bitbrains, EQ-VMC slightly outperforms RO-VMC due to resource reservation focus, while AFED-EF falls short. RO-VMC's superiority stems from multidimensional resource optimization, maximizing consolidation opportunities regardless of deployment scale. Notably, imbalance frequency decreases as host count increases due to larger destination host pools enabling more informed migration decisions and reduced per-host resource demands.

5.8.3 Assessment of QoS

To quantify the effect of migration on QoS, three performance metrics are defined. SLATAH measures the SLA violation time per active host, indicating prolonged

duration of SLA violations caused by host overloading, defined as:

$$SLATAH = \frac{1}{M}\sum_{i=1}^{M}\frac{T_{over}^{i}}{T_{act}^{i}} \tag{5.20}$$

where T_{over}^{i} and T_{act}^{i} represent the overloading duration and total active time of host H_i, respectively. PDM denotes performance degradation due to migrations, capturing the percentage reduction in expected VM performance:

$$PDM = \frac{1}{Z}\sum_{j=1}^{Z}\frac{R_{deg}^{j}}{R_{req}^{j}} \tag{5.21}$$

where Z is the total number of VMs operational throughout the execution period, R_{deg}^{j} is the estimated performance degradation of VM V_j and R_{req}^{j} is the total CPU capacity requested by V_j. In this work, R_{derg}^{j} is estimated as 10% of V_j's CPU utilization, which has been calculated as:

$$R_{deg}^{j} = \frac{1}{10}\int_{t}^{t+Mig_{j}^{t}} D_{jr_k}^{t}dt \tag{5.22}$$

where $r_k = CPU$ and Mig_j^t is the time taken to migrate V_j at time t. SLAV calculates the combined impact of SLATAH and PDM, reflecting overall QoS satisfaction:

$$SLAV = SLATAH \cdot PDM \tag{5.23}$$

QoS evaluation employs SLATAH, PDM, and SLAV metrics to assess approach effectiveness. Tables 5.6 and 5.7 summarize performance across both traces, including frequency of overloading (FO) and VM migrations (VMMs) that directly relate to SLATAH and PDM respectively.

For SLATAH, RO-VMC slightly lags behind EQ-VMC on Bitbrains trace by 6% due to higher resource demands and workload variations resulting in increased

TABLE 5.6: Simulation results on QoS metrics using Bitbrains trace.

	FO/hr	**SLATAH(%)**	**VMM/hr**	**PDM(%)**	**SLAV**
RO-VMC	6.8	2.97	64.4	0.026	0.078
PEAS	11.4	3.78	160.1	0.052	0.197
AFED-EF	7.3	3.13	167.5	0.041	0.128
EQ-VMC	5.7	2.78	114.3	0.033	0.092

TABLE 5.7: Simulation results on QoS metrics using GCC trace.

	FO/hr	**SLATAH(%)**	**VMM/hr**	**PDM(%)**	**SLAV**
RO-VMC	3.9	1.83	97.4	0.021	0.039
PEAS	9.3	3.28	208.1	0.034	0.111
AFED-EF	4.7	2.23	194.3	0.028	0.062
EQ-VMC	4.4	2.18	161.2	0.019	0.041

FO despite fewer active hosts. However, on GCC trace, RO-VMC outperforms alternatives by 44%, 18%, and 16% compared to PEAS, AFED-EF, and EQ-VMC respectively. The consistent workload demands of GCC allow RO-VMC to manage VMs with significantly reduced FO, while PEAS consistently underperforms due to its CPU-centric approach's inability to handle multi-resource environments effectively.

Regarding PDM evaluation, RO-VMC achieves the lowest VMMs for both traces, minimizing migration-induced disruptions. For Bitbrains trace, RO-VMC outperforms PEAS, AFED-EF, and EQ-VMC by 50%, 36%, and 21% respectively. On GCC trace, while EQ-VMC outperforms RO-VMC by 9% due to its VM selection strategy considering both migration time and overloading probability, RO-VMC still demonstrates superiority over PEAS and AFED-EF by 38% and 25% respectively. SLAV assessment demonstrates RO-VMC's overall superiority by combining both metrics. For Bitbrains trace, RO-VMC outperforms PEAS, AFED-EF, and EQ-VMC by 60%, 39%, and 15% respectively, while for GCC trace the margins are 65%, 37%, and 4% respectively.

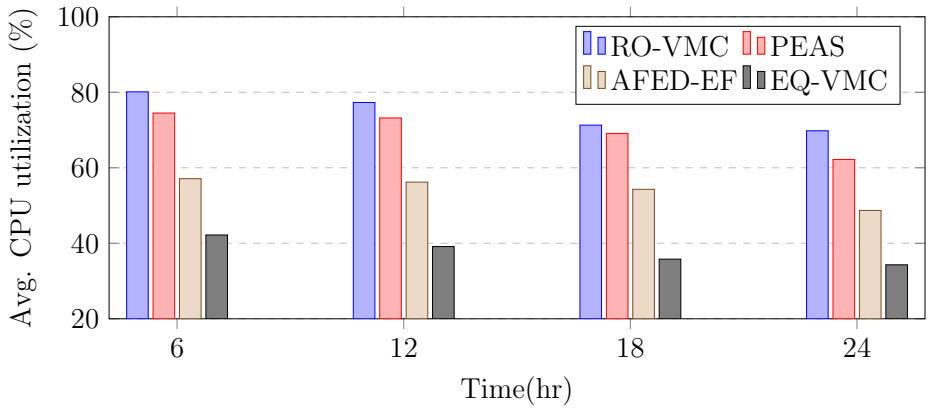

FIGURE 5.8: CPU utilization over the simulation period using Bitbrains trace.

5.8.4 Resource Utilization Efficiency

Resource utilization evaluation examines CPU, memory, and disk efficiency alongside QoS and energy consumption metrics. Figures 5.8 - 5.13 present the hourly average CPU, memory and disk utilization of the DC for all the approaches during the simulation period on both traces. The results demonstrate RO-VMC's superiority with performance order: RO-VMC > PEAS > AFED-EF > EQ-VMC.

RO-VMC's superior performance stems from two key components: game-based VM placement facilitating minimal resource wastage across all dimensions, and VM selection designed to minimize non-overutilized resource wastage. PEAS performs closely with utilization gaps of 1.4%-7.6% for CPU, 8.3%-16.2% for memory, and 6.2%-9.5% for disk, attributed to selecting hosts with smaller capacities for migrating VMs, maintaining higher utilization states. AFED-EF's distribution criteria based on energy consumption and SLA violation product fails to determine optimal destinations under high dynamicity, exhibiting larger gaps: 16%-23.1% for CPU, 19.2%-27% for memory, and 13.7%-20.6% for disk. EQ-VMC focuses on resource conservation rather than utilization, showing average reductions of 34.2%-38.17% for CPU, 27%-33.9% for memory, and 18.1%-24.6% for disk compared to RO-VMC.

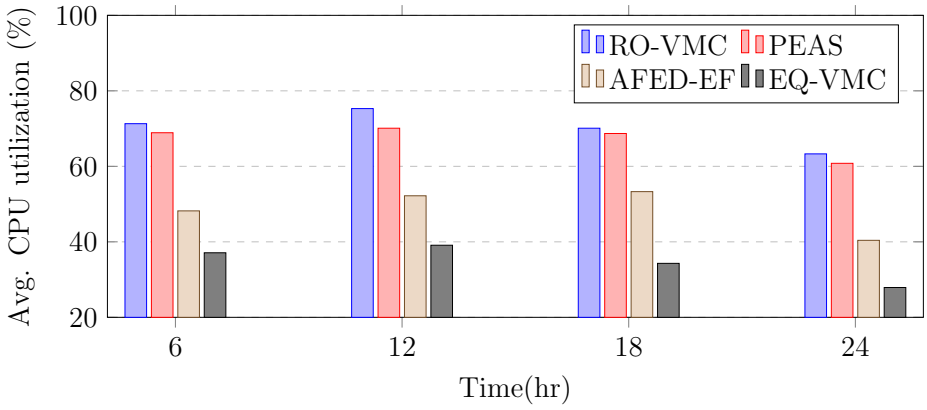

FIGURE 5.9: CPU utilization over the simulation period using GCC trace.

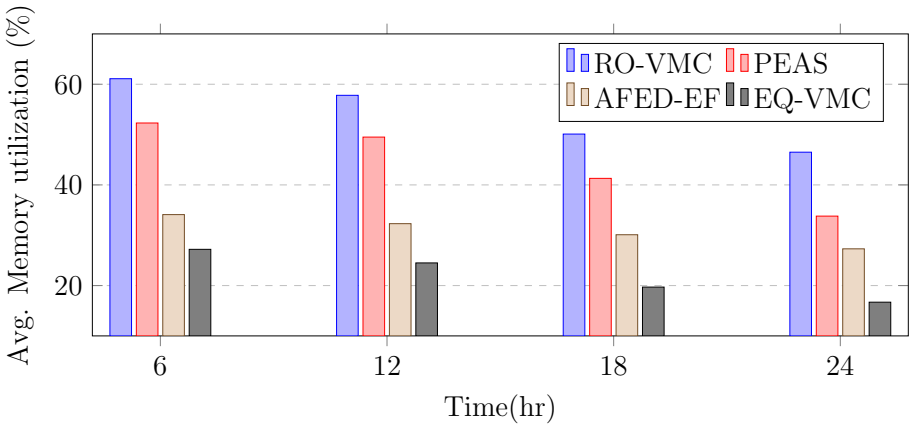

FIGURE 5.10: Memory utilization over the simulation period using Bitbrains trace.

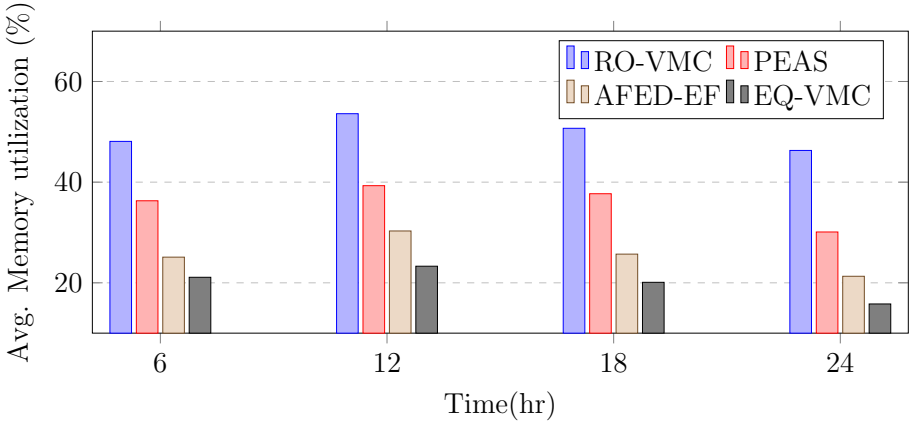

FIGURE 5.11: Memory utilization over the simulation period using GCC trace.

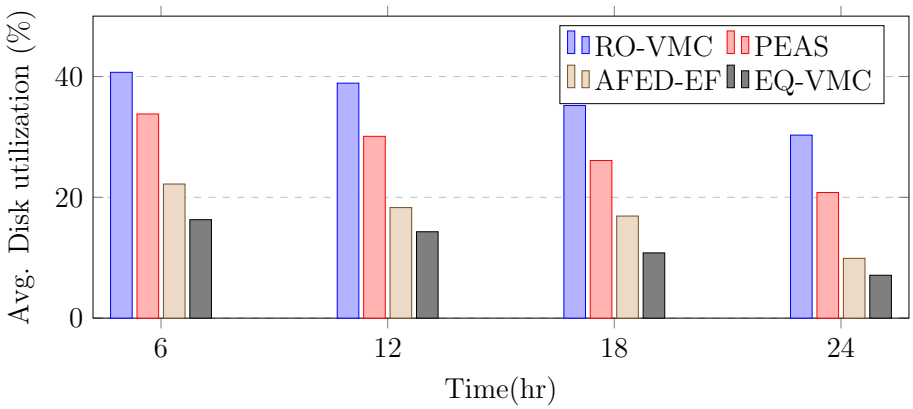

FIGURE 5.12: Disk utilization over the simulation period using Bitbrains trace.

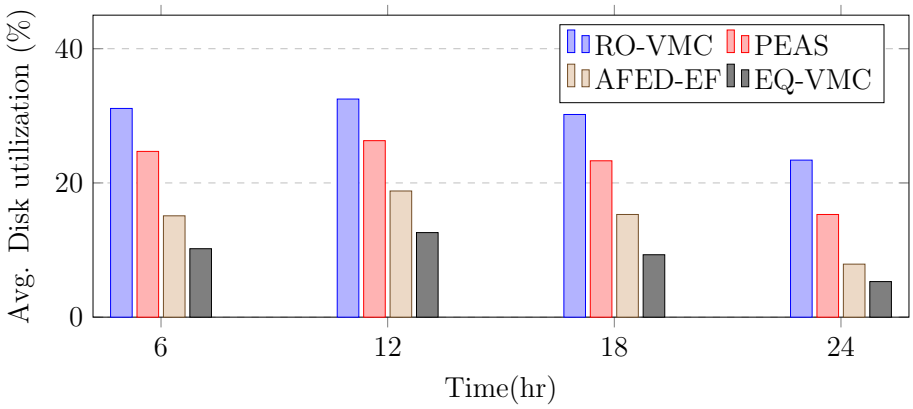

FIGURE 5.13: Disk utilization over the simulation period using GCC trace.

Overall, RO-VMC emerges as the superior alternative, promoting energy efficiency while providing satisfactory QoS through optimal multi-dimensional resource utilization.

5.9 Summary

This chapter addressed the energy efficiency limitations identified in Chapter 4's load balancing framework through the Resource-Optimized VM Consolidation (RO-VMC) approach, completing the comprehensive cloud resource management solution. The framework integrated energy optimization with service quality preservation by employing stochastic load imbalance detection using probabilistic resource modeling, resource intensity-aware VM selection through multi-criteria decision making, and game-theoretic VM distribution achieving Nash equilibrium solutions that balanced energy efficiency with service quality guarantees.

Extensive evaluation using real-world traces (Bitbrains and GCC) demonstrated RO-VMC's superiority: 21-65% reduction in active hosts, 17-46% decrease in energy consumption, 37-70% fewer load imbalances, and 15-65% improvement in overall QoS metrics compared to existing approaches. The multi-objective optimization successfully balanced energy minimization with migration control, demonstrating that aggressive consolidation can be achieved without compromising the service quality and responsiveness established in previous chapters.

The combination of proactive VM placement (Chapter 3), SLA-aware load balancing (Chapter 4), and energy-efficient consolidation (Chapter 5) created a holistic solution addressing the full spectrum of cloud computing challenges. Experimental validation demonstrated simultaneous optimization of energy efficiency, service quality, and resource utilization across diverse workload scenarios. This thesis delivered three interconnected contributions: proactive resource management through advanced workload prediction and intelligent placement, dynamic load balancing via

SLA-aware frameworks maintaining service quality during operations, and energy-efficient consolidation achieving aggressive optimization while preserving system resilience. The stochastic modeling foundation proved instrumental throughout, enabling sophisticated decision-making that handles cloud environment uncertainty.

The research demonstrates that comprehensive cloud resource management requires coordinated optimization across multiple time horizons rather than isolated solutions. The framework's modular design enables flexible deployment while providing measurable improvements across all performance dimensions. This integrated approach establishes a foundation for next-generation cloud systems that adapt intelligently to evolving workloads while maintaining optimal performance, advancing sustainable and efficient cloud computing infrastructure management. The demonstrated performance improvements validate the framework's real-world applicability, enabling cloud providers to achieve operational excellence through simultaneous optimization of energy efficiency, resource utilization, and service delivery quality.

Chapter 6

Conclusion and Future Work

This thesis presented a comprehensive framework for cloud resource management that integrates workload prediction, virtual machine placement, dynamic load balancing, and energy-efficient consolidation into a unified solution. Motivated by the fragmented nature of existing approaches that treat interconnected resource management challenges as isolated problems, this research established a holistic paradigm demonstrating that coordinated optimization across the VM lifecycle yields superior system-wide performance. The stochastic modeling foundation enables sophisticated decision-making under uncertainty, while extensive experimental validation using real-world workload traces confirmed substantial improvements: 12-46% energy reduction, 10-39% fewer migrations, 17-70% fewer load imbalances, and 9-33% improved service quality compared to state-of-the-art approaches.

6.1 Summary of Contributions

This thesis addressed the fundamental challenge of comprehensive resource management in cloud computing through an integrated framework coordinating workload prediction, virtual machine placement, dynamic load balancing, and energy-efficient

consolidation. The research bridged a critical gap in existing literature where interconnected resource management challenges were treated as isolated problems, resulting in suboptimal system-wide performance. The primary contribution lies in establishing a holistic resource management paradigm where multiple optimization phases operate synergistically across the entire VM lifecycle through three interconnected innovations.

First, a proactive resource management approach combining cluster-specific machine learning models with statistical-stochastic forecasting enabled accurate workload prediction under uncertainty. The temporal pattern recognition framework identified distinct workload clusters through Agglomerative Hierarchical Clustering with Dynamic Time Warping, while the stochastic framework provided probabilistic resource consumption modeling. These predictions informed intelligent VM placement strategies—heuristic-based (MBFD) and game-theoretic (GB-VMP)—optimizing initial allocation while minimizing energy consumption. Experimental validation demonstrated 15.2 active hosts while maintaining ¿90% CPU utilization, yielding up to 12% energy reduction.

Second, an SLA-aware load balancing framework extended optimization from static placement to continuous operational management through probabilistic overload detection and intelligent migration optimization incorporating cumulative SLA violation tracking. Evaluation demonstrated 10-39% fewer migrations, 17-54% fewer overloaded hosts, and 9-33% reduction in performance degradation compared to state-of-the-art approaches while maintaining approximately 68% CPU and 57% memory utilization.

Third, energy-efficient VM consolidation through Resource-Optimized VM Consolidation (RO-VMC) integrated stochastic load imbalance detection with resource intensity-aware VM selection and game-theoretic distribution. Comprehensive evaluation demonstrated 21-65% reduction in active hosts, 17-46% decrease in energy

consumption, 37-70% fewer load imbalances, and 15-65% improvement in overall QoS metrics.

The seamless integration demonstrated that coordinated optimization across multiple time horizons yields superior performance compared to isolated solutions. Key principles established include: treating resource consumption as stochastic processes enables robust allocation through explicit uncertainty quantification; coordinating optimization across the VM lifecycle yields synergistic effects; integrating multiple paradigms creates complementary strengths; and balancing competing objectives requires adaptive multi-objective frameworks. The modular design enables flexible incremental deployment, reducing implementation barriers while facilitating practical adoption.

6.2 Limitations and Future Scope

Several promising directions warrant investigation to address emerging challenges and extend framework capabilities.

- Advanced Learning and Heterogeneous Resources: Future research should investigate deep reinforcement learning for end-to-end optimization capturing complex phase interactions. Transfer learning could address cold-start problems, while attention-based architectures may better capture temporal dependencies and cross-resource correlations. Extending the framework to explicitly model GPU utilization, specialized accelerators, comprehensive network topology, and storage performance characteristics would address critical gaps. Developing unified frameworks handling heterogeneous resources while maintaining computational tractability through approximation and hierarchical optimization presents significant challenges.

- Scalability and Emerging Paradigms: Hierarchical resource management architectures with regional controllers and global coordination warrant investigation for hyperscale deployments. Extending to federated cloud environments requires addressing heterogeneous pricing, inter-data center costs, and data sovereignty constraints. Fully autonomous systems automatically adjusting models and parameters through online learning and self-healing mechanisms could enhance autonomy. Explainable AI providing transparency through interpretable predictions and decision justification becomes crucial for trust. Carbon-aware computing considering grid carbon intensity, renewable energy availability, and geographic load shifting represents increasing environmental importance. Adapting for serverless computing and containerized workloads requiring millisecond-scale decisions necessitates fundamentally different strategies. Quantum computing integration for hybrid quantum-classical workloads requires entirely new frameworks.

- Intelligent Autonomy and Explainability: Developing fully autonomous systems automatically adjusting models and parameters based on observed performance through online learning and self-healing mechanisms could enhance autonomy. Integrating explainable AI to provide transparency through interpretable predictions, decision justification, and root cause analysis becomes crucial for building trust in automated systems.

- Security and Validation: Security-aware consolidation integrating co-location

restrictions, isolation requirements, side-channel attack mitigation, and compliance with data sovereignty regulations represents critical needs. Privacy-preserving management using federated learning, differential privacy, and secure multi-party computation could enable cooperation while maintaining confidentiality. Long-term production evaluation assessing performance under diverse real-world conditions, integration with existing platforms, and operational considerations remains essential. Collaboration with providers to establish standardized interfaces, best practices, and benchmarking methodologies would facilitate adoption. Economic analysis including total cost of ownership and return on investment provides necessary business justification.

These directions extend the framework's capabilities while addressing emerging challenges. The fundamental principles established—stochastic modeling under uncertainty, lifecycle-oriented optimization, multi-objective balancing, and integrated decision-making—provide a foundation for future development. As cloud computing evolves with increasing scale, workload diversity, sustainability imperatives, and security requirements, continued research will be essential for next-generation resource management systems maintaining efficiency, reliability, and service quality.

References

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6): 599–616, 2009.

[2] Zhangjun Wu, Xiao Liu, Zhiwei Ni, Dong Yuan, and Yun Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293, 2013.

[3] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

[4] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167, 2017.

[5] Sukhpal Singh and Inderveer Chana. Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems*, 49(3):1005–1069, 2016.

[6] Sunilkumar S Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of network and computer applications*, 41:424–440, 2014.

[7] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data center energy consumption modeling: A survey. *IEEE Communications surveys & tutorials*, 18(1):732–794, 2015.

[8] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4): 1012–1023, 2013.

[9] Nicola Jones et al. How to stop data centres from gobbling up the world's electricity. *nature*, 561(7722):163–166, 2018.

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

[11] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and distributed Computing*, 70(9):962–974, 2010.

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

[13] Minxian Xu, Wenhong Tian, and Rajkumar Buyya. A survey on load balancing algorithms for virtual machines placement in cloud computing. *Concurrency and Computation: Practice and Experience*, 29(12):e4123, 2017.

[14] Jagdeep Singh, Parminder Singh, Mustapha Hedabou, and Neeraj Kumar. An efficient machine learning-based resource allocation scheme for sdn-enabled fog computing environment. *IEEE Transactions on Vehicular Technology*, 72(6):8004–8017, 2023.

[15] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[16] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[17] Mohammed Ala'Anzy and Mohamed Othman. Load balancing and server consolidation in cloud computing environments: a meta-study. *IEEE Access*, 7:141868–141887, 2019.

[18] Patryk Osypanka and Piotr Nawrocki. Qos-aware cloud resource prediction for computing services. *IEEE Transactions on Services Computing*, 16(2):1346–1357, 2022.

[19] Google cluster data. `https://code.google.com/p/googleclusterdata/`, Dec 2020.

[20] PlanetLab workload traces. `https://github.com/beloglazov/planetlab-workload-traces/`, Dec 2020.

[21] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P Williamson. Adversarial queueing theory. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 376–385, 1996.

[22] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 1:1–14, 2011.

[23] Peter A Dinda. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 17(2):160–173, 2006.

[24] Jungmin Son, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers. *IEEE Transactions on Sustainable Computing*, 2(2):76–89, 2017.

[25] Josep Subirats and Jordi Guitart. Assessing and forecasting energy efficiency on cloud computing platforms. *Future Generation Computer Systems*, 45:70–94, 2015.

[26] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the seventh conference on emerging networking experiments and technologies*, pages 1–12, 2011.

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

[28] Zhijia Chen, Yuanchang Zhu, Yanqiang Di, and Shaochong Feng. Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering based fuzzy neural network. *Computational intelligence and neuroscience*, 2015(1):919805, 2015.

[29] Salam Ismaeel and Ali Miri. Using elm techniques to predict data centre vm requests. In *2015 IEEE 2nd international conference on cyber security and cloud computing*, pages 80–86. IEEE, 2015.

[30] Binbin Song, Yao Yu, Yu Zhou, Ziqiang Wang, and Sidan Du. Host load prediction with long short-term memory in cloud computing. *The Journal of Supercomputing*, 74(12):6554–6568, 2018.

[31] Chenglei Peng, Yang Li, Yao Yu, Yu Zhou, and Sidan Du. Multi-step-ahead host load prediction with gru based encoder-decoder in cloud computing. In *2018 10th International Conference on Knowledge and Smart Technology (KST)*, pages 186–191. IEEE, 2018.

[32] Nancy Tran and Daniel A Reed. Automatic arima time series modeling for adaptive i/o prefetching. *IEEE Transactions on parallel and distributed systems*, 15(4):362–377, 2004.

[33] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Energy-efficient resource allocation and provisioning framework for cloud data centers. *IEEE Transactions on Network and Service Management*, 12(3):377–391, 2015.

[34] Peter A Dinda and David R O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, 2000.

[35] Stanly Jayaprakash, Manikanda Devarajan Nagarajan, Rocío Pérez de Prado, Sugumaran Subramanian, and Parameshachari Bidare Divakarachari. A systematic review of energy management strategies for resource allocation in the cloud: Clustering, optimization and machine learning. *Energies*, 14(17):5322, 2021.

[36] Sania Malik, Muhammad Tahir, Muhammad Sardaraz, and Abdullah Alourani. A resource utilization prediction model for cloud data centers using evolutionary algorithms and machine learning techniques. *Applied Sciences*, 12(4):2160, 2022.

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

[38] Kaixuan Kang, Ding Ding, Huamao Xie, Qian Yin, and Jing Zeng. Adaptive drl-based task scheduling for energy-efficient cloud computing. *IEEE Transactions on Network and Service Management*, 19(4):4948–4961, 2021.

[39] Thomas Wang, Simone Ferlin, and Marco Chiesa. Predicting cpu usage for proactive autoscaling. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 31–38, 2021.

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

[41] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE international conference on smart cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

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

[43] KC Anupama, BR Shivakumar, and R Nagaraja. Resource utilization prediction in cloud computing using hybrid model. *International Journal of Advanced Computer Science and Applications*, 12(4), 2021.

[44] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. *ACM SIGOPS operating systems review*, 41(6): 265–278, 2007.

[45] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS operating systems review*, 35(5):103–116, 2001.

[46] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 826–831. IEEE, 2010.

[47] Edward G Coffman, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Combinatorial analysis. In *Handbook of Combinatorial Optimization: Supplement Volume A*, pages 151–207. Springer, 1999.

[48] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE transactions on parallel and distributed systems*, 24(7):1366–1379, 2012.

[49] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE Transactions on Cloud Computing*, 1(2):215–228, 2013.

[50] Arman Iranfar, Marina Zapater, and David Atienza. Machine learning-based quality-aware power and thermal management of multistream hevc encoding on multicore servers. *IEEE Transactions on Parallel and Distributed Systems*, 29(10):2268–2281, 2018.

[51] Hancong Duan, Chao Chen, Geyong Min, and Yu Wu. Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Future Generation*

Computer Systems, 74:142–150, 2017.

[52] Emmanuel Arzuaga and David R Kaeli. Quantifying load imbalance on virtualized enterprise servers. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 235–242, 2010.

[53] Dhinesh Babu LD and P Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied soft computing*, 13(5): 2292–2303, 2013.

[54] Gunjan Khanna, Kirk Beaty, Gautam Kar, and Andrzej Kochut. Application performance management in virtualized server environments. In *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*, pages 373–381. IEEE, 2006.

[55] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.

[56] Haiying Shen and Liuhua Chen. A resource usage intensity aware load balancing method for virtual machine migration in cloud datacenters. *IEEE Transactions on Cloud Computing*, 8(1):17–31, 2017.

[57] Chubo Liu, Kenli Li, and Keqin Li. A game approach to multi-servers load balancing with load-dependent server availability consideration. *IEEE Transactions on Cloud Computing*, 9(1):1–13, 2018.

[58] Said Nabi, Muhammad Ibrahim, and Jose M Jimenez. Dralba: Dynamic and resource aware load balanced scheduling approach for cloud computing. *Ieee Access*, 9:61283–61297, 2021.

[59] Mehran Ashouraei, Seyed Nima Khezr, Rachid Benlamri, and Nima Jafari Navimipour. A new sla-aware load balancing method in the cloud using an improved parallel task scheduling algorithm. In *2018 IEEE 6th international conference on future internet of things and cloud (FiCloud)*, pages 71–76. IEEE, 2018.

[60] Fei Xu, Fangming Liu, Hai Jin, and Athanasios V Vasilakos. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proceedings of the IEEE*, 102(1):11–31, 2013.

[61] Fei Xu, Fangming Liu, and Hai Jin. Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud. *IEEE Transactions on Computers*, 65(8):2470–2483, 2015.

[62] Fei Xu, Fangming Liu, Linghui Liu, Hai Jin, Bo Li, and Baochun Li. iaware: Making live migration of virtual machines interference-aware in the cloud. *IEEE transactions on computers*, 63(12):3012–3025, 2013.

[63] Aroosa Mubeen, Muhammad Ibrahim, Nargis Bibi, Mohammad Baz, Habib Hamam, and Omar Cheikhrouhou. Alts: An adaptive load balanced task scheduling approach for cloud computing. *Processes*, 9(9):1514, 2021.

[64] In Kee Kim, Wei Wang, Yanjun Qi, and Marty Humphrey. Forecasting cloud application workloads with cloudinsight for predictive resource management. *IEEE Transactions on Cloud Computing*, 10(3):1848–1863, 2020.

[65] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–14, 2011.

[66] Qingyi Gao, Peng Tang, Ting Deng, and Tianyu Wo. Virtualrank: A prediction based load balancing technique in virtual computing environment. In *2011 IEEE World Congress on Services*, pages 247–256. IEEE, 2011.

[67] Mayank Sohani and SC Jain. A predictive priority-based dynamic resource provisioning scheme with load balancing in heterogeneous cloud computing. *IEEE access*, 9:62653–62664, 2021.

[68] Deepika Saxena, Ashutosh Kumar Singh, and Rajkumar Buyya. Op-mlb: An online vm prediction-based multi-objective load balancing framework for resource management at cloud data center. *IEEE Transactions on Cloud Computing*, 10(4): 2804–2816, 2021.

[69] Lung-Hsuan Hung, Chih-Hung Wu, Chiung-Hui Tsai, and Hsiang-Cheh Huang. Migration-based load balance of virtual machine servers in cloud computing by load prediction using genetic-based methods. *IEEE Access*, 9:49760–49773, 2021.

[70] Benay Kumar Ray, Avirup Saha, Sunirmal Khatua, and Sarbani Roy. Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment. *IEEE Transactions on Cloud Computing*, 10(2):957–971, 2020.

[71] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128. IEEE, 2007.

[72] Sarita Negi, Man Mohan Singh Rauthan, Kunwar Singh Vaisla, and Neelam Panwar. Cmodlb: an efficient load balancing approach in cloud computing environment. *The Journal of Supercomputing*, 77(8):8787–8839, 2021.

[73] Kethineni Vinod Kumar and A Rajesh. Multi-objective load balancing in cloud computing: a meta-heuristic approach. *Cybernetics and Systems*, 54(8):1466–1493, 2023.

[74] Stavros Souravlas, Sofia D Anastasiadou, Nicoleta Tantalaki, and Stefanos Katsavounis. A fair, dynamic load balanced task distribution strategy for heterogeneous cloud platforms based on markov process modeling. *IEEE Access*, 10:26149–26162, 2022.

[75] Mayank Mishra, Anwesha Das, Purushottam Kulkarni, and Anirudha Sahoo. Dynamic resource management using virtual machine migrations. *IEEE Communications Magazine*, 50(9):34–40, 2012.

[76] Rahmat Zolfaghari and Amir Masoud Rahmani. Virtual machine consolidation in cloud computing systems: Challenges and future trends. *Wireless Personal Communications*, 115(3):2289–2326, 2020.

[77] Kyungmee Chang, Sangun Park, Hyesoo Kong, and Wooju Kim. Optimizing energy consumption for a performance-aware cloud data center in the public sector. *Sustainable Computing: Informatics and Systems*, 20:34–45, 2018.

[78] Weiwei Lin, Siyao Xu, Ligang He, and Jin Li. Multi-resource scheduling and power simulation for cloud computing. *Information Sciences*, 397:168–186, 2017.

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

[80] Hossein Monshizadeh Naeen. Virtual machine consolidation using sla-aware genetic algorithm placement for data centers with non-stationary workloads. In *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, pages 150–156. IEEE, 2021.

[81] Minhaj Ahmad Khan. An efficient energy-aware approach for dynamic vm consolidation on cloud platforms. *Cluster Computing*, 24(4):3293–3310, 2021.

[82] Monireh H Sayadnavard, Abolfazl Toroghi Haghighat, and Amir Masoud Rahmani. A multi-objective approach for energy-efficient and reliable dynamic vm consolidation in cloud data centers. *Engineering science and technology, an International Journal*, 26:100995, 2022.

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

[84] Zhihua Li, Kaiqing Lin, Shunhang Cheng, Lei Yu, and Junhao Qian. Energy-efficient and load-aware vm placement in cloud data centers. *Journal of Grid Computing*, 20 (4):39, 2022.

[85] Bhagyalakshmi Magotra, Deepti Malhotra, and Amit Kr Dogra. Adaptive computational solutions to energy efficiency in cloud computing environment using vm consolidation. *Archives of computational methods in engineering*, 30(3):1789–1818, 2023.

[86] Zhihua Li, Xinrong Yu, Lei Yu, Shujie Guo, and Victor Chang. Energy-efficient and quality-aware vm consolidation method. *Future Generation Computer Systems*, 102: 789–809, 2020.

[87] Abhishek Kumar Pandey and Sarvpal Singh. An energy efficient particle swarm optimization based vm allocation for cloud data centre: Eevmpso. *EAI Endorsed Transactions on Scalable Information Systems*, 10(5), 2023.

[88] Huanlai Xing, Jing Zhu, Rong Qu, Penglin Dai, Shouxi Luo, and Muhammad Azhar Iqbal. An aco for energy-efficient and traffic-aware virtual machine placement in cloud computing. *Swarm and Evolutionary Computation*, 68:101012, 2022.

[89] Anurina Tarafdar, Mukta Debnath, Sunirmal Khatua, and Rajib K Das. Energy and quality of service-aware virtual machine consolidation in a cloud data center. *The Journal of Supercomputing*, 76(11):9095–9126, 2020.

[90] Wei Li, Qi Fan, Wenchao Cui, Fangfang Dang, Xiaoliang Zhang, and Cheng Dai. Dynamic virtual machine consolidation algorithm based on balancing energy consumption and quality of service. *IEEE Access*, 10:80958–80975, 2022.

[91] Jinjiang Wang, Hangyu Gu, Junyang Yu, Yixin Song, Xin He, and Yalin Song. Research on virtual machine consolidation strategy based on combined prediction and energy-aware in cloud computing platform. *Journal of Cloud Computing*, 11(1): 50, 2022.

[92] Jing Zeng, Ding Ding, Kaixuan Kang, HuaMao Xie, and Qian Yin. Adaptive drl-based virtual machine consolidation in energy-efficient cloud data center. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2991–3002, 2022.

[93] Minh-Ngoc Tran, Xuan Tuong Vu, and Younghan Kim. Proactive stateful fault-tolerant system for kubernetes containerized services. *IEEE Access*, 10:102181–102194, 2022.

[94] Monireh H Sayadnavard, Abolfazl Toroghi Haghighat, and Amir Masoud Rahmani. A reliable energy-aware approach for dynamic virtual machine consolidation in cloud data centers: Mh sayadnavard et al. *The Journal of Supercomputing*, 75(4):2126–2147, 2019.

[95] Weiwei Lin, Wentai Wu, and Ligang He. An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers. *IEEE Transactions on Services Computing*, 15(2): 766–777, 2019.

[96] Lei Xie, Shengbo Chen, Wenfeng Shen, and Huaikou Miao. A novel self-adaptive vm consolidation strategy using dynamic multi-thresholds in iaas clouds. *Future Internet*, 10(6):52, 2018.

[97] Zhihua Li, Chengyu Yan, Lei Yu, and Xinrong Yu. Energy-aware and multi-resource overload probability constraint-based virtual machine dynamic consolidation method. *Future Generation Computer Systems*, 80:139–156, 2018.

[98] Seyed Yahya Zahedi Fard, Mohamad Reza Ahmadi, and Sahar Adabi. A dynamic vm consolidation technique for qos and energy consumption in cloud environment. *The Journal of Supercomputing*, 73(10):4347–4368, 2017.

[99] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, Nguyen Trung Hieu, and Hannu Tenhunen. Energy-aware vm consolidation in cloud data centers using utilization prediction model. *IEEE Transactions on Cloud Computing*, 7(2): 524–536, 2016.

[100] Riddhi Thakkar and Madhuri Bhavsar. Achieving multilevel elasticity for distributed stream processing systems in the cloud environment: A review and conceptual framework. In *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing*, pages 81–90, 2022.

[101] Mahfoudh Saeed Al-Asaly, Mohamed A Bencherif, Ahmed Alsanad, and Mohammad Mehedi Hassan. A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment. *Neural Computing and Applications*, 34(13):10211–10228, 2022.

[102] Joshua Peake, Martyn Amos, Nicholas Costen, Giovanni Masala, and Huw Lloyd. Paco-vmp: parallel ant colony optimization for virtual machine placement. *Future Generation Computer Systems*, 129:174–186, 2022.

[103] Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *Future generation computer systems*, 79:849–861, 2018.

[104] Sukhpal Singh Gill, Shreshth Tuli, Minxian Xu, Inderpreet Singh, Karan Vijay Singh, Dominic Lindsay, Shikhar Tuli, Daria Smirnova, Manmeet Singh, Udit Jain, et al. Transformative effects of iot, blockchain and artificial intelligence on cloud computing: Evolution, vision, trends and open challenges. *Internet of Things*, 8: 100118, 2019.

[105] Mazen Farid, Rohaya Latip, Masnida Hussin, and Nor Asilah Wati Abdul Hamid. Scheduling scientific workflow using multi-objective algorithm with fuzzy resource utilization in multi-cloud environment. *IEEE Access*, 8:24309–24322, 2020.

[106] Ilyas Bambrik. A survey on cloud computing simulation and modeling. *SN Computer Science*, 1(5):249, 2020.

[107] Shengyan Wen, Xiaohang Wang, Amit Kumar Singh, Yingtao Jiang, and Mei Yang. Performance optimization of many-core systems by exploiting task migration and dark core allocation. *IEEE Transactions on Computers*, 71(1):92–106, 2020.

[108] Elnaz Parvizi and Mohammad Hossein Rezvani. Utilization-aware energy-efficient virtual machine placement in cloud networks using nsga-iii meta-heuristic approach. *Cluster computing*, 23(4):2945–2967, 2020.

[109] Abdelaziz Said Abohamama and Eslam Hamouda. A hybrid energy–aware virtual machine placement algorithm for cloud environments. *Expert Systems with Applications*, 150:113306, 2020.

[110] Yen-Lin Chen, Ming-Feng Chang, Chao-Wei Yu, Xiu-Zhi Chen, and Wen-Yew Liang. Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems. *Sensors*, 18(9):3068, (2018).

[111] Pedro Pereira Rodrigues, Joao Gama, and Joao Pedroso. Hierarchical clustering of time-series data streams. *IEEE transactions on knowledge and data engineering*, 20 (5):615–627, 2008.

[112] Maciej Łuczak. Hierarchical clustering of time series data with parametric derivative dynamic time warping. *Expert Systems with Applications*, 62:116–130, 2016.

[113] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 181–188. IEEE, 2018.

[114] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. Effective vm sizing in virtualized data centers. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 594–601. IEEE, 2011.

[115] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144, 2017.

[116] Jitendra Kumar and Ashutosh Kumar Singh. Cloud datacenter workload estimation using error preventive time series forecasting models. *Cluster Computing*, 23(2): 1363–1379, 2020.

[117] Maurizio Rossi and Davide Brunelli. Forecasting data centers power consumption with the holt-winters method. In *2015 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS) Proceedings*, pages 210–214. IEEE, 2015.

[118] Mydhili K Nair et al. Bin packing algorithms for virtual machine placement in cloud computing: a review. *International Journal of Electrical & Computer Engineering (2088-8708)*, 9(1), 2019.

[119] The SPECpower Benchmark. `https://www.spec.org/power_ssj2008/results/res2020q1/power_ssj2008-20200310-01018.html/`, 2008.

[120] Trace Description. `http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains/`, 2015.

[121] Siqi Shen, Vincent Van Beek, and Alexandru Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *2015 15th IEEE/ACM international symposium on cluster, cloud and grid computing*, pages 465–474. IEEE, 2015.

[122] Imene El-Taani, Mohand-Cherif Boukala, and Samia Bouzefrane. Energy-aware vm placement based on intra-balanced resource allocation in data centers. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 400–405. IEEE, 2021.

[123] G Madhukar Rao, K Srinivas, Sayyad Samee, K Venkatesh, Pankaj Dadheech, Linesh Raja, and Garvit Yagnik. A secure and efficient data migration over cloud computing. In *IOP Conference Series: Materials Science and Engineering*, volume 1099, page 012082. IOP Publishing, 2021.

[124] Mohammad Aldossary. A review of dynamic resource management in cloud computing environments. *Computer Systems Science & Engineering*, 36(3), 2021.

[125] HM Dipu Kabir, Abbas Khosravi, Subrota K Mondal, Mustaneer Rahman, Saeid Nahavandi, and Rajkumar Buyya. Uncertainty-aware decisions in cloud computing: Foundations and future directions. *ACM Computing Surveys (CSUR)*, 54(4):1–30, 2021.

[126] EI Elsedimy and Fahad Algarni. Toward enhancing the energy efficiency and minimizing the sla violations in cloud data centers. *Applied Computational Intelligence and Soft Computing*, 2021(1):8892734, 2021.

[127] Seyedhamid Mashhadi Moghaddam, Michael O'Sullivan, Charles Peter Unsworth, Sareh Fotuhi Piraghaj, and Cameron Walker. Metrics for improving the management of cloud environments—load balancing using measures of quality of service, service level agreement violations and energy consumption. *Future Generation Computer Systems*, 123:142–155, 2021.

[128] Monika Singh, Pardeep Kumar, and Sanjay Tyagi. Adaptive energy-aware algorithms to minimize power consumption and sla violation in cloud computing. *Recent*

Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science), 14(4):1008–1015, 2021.

[129] Mohamed Esam Elsaid, Hazem M Abbas, and Christoph Meinel. Virtual machines pre-copy live migration cost modeling and prediction: a survey. *Distributed and Parallel Databases*, 40(2):441–474, 2022.

[130] NM Novikova and II Pospelova. Applying the linear scalarization in multicriteria maximin problems. *Moscow University Computational Mathematics and Cybernetics*, 45(2):71–80, 2021.

[131] *Plos one*, 17(1):e0261856, 2022.

[132] Muhammed Tawfiqul Islam, Huaming Wu, Shanika Karunasekera, and Rajkumar Buyya. Sla-based scheduling of spark jobs in hybrid cloud computing environments. *IEEE Transactions on Computers*, 71(5):1117–1132, 2021.

[133] Muhammad Zakarya, Lee Gillam, Khaled Salah, Omer Rana, Santosh Tirunagari, and Rajkumar Buyya. Colocateme: Aggregation-based, energy, performance and cost aware vm placement and consolidation in heterogeneous iaas clouds. *IEEE transactions on services computing*, 16(2):1023–1038, 2022.

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

[135] Xiaomin Zhu, Laurence T Yang, Huangke Chen, Ji Wang, Shu Yin, and Xiaocheng Liu. Real-time tasks oriented energy-aware scheduling in virtualized clouds. *IEEE Transactions on Cloud Computing*, 2(2):168–180, 2014.

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

Sourak Banerjee
30.03.2026

Sarbani Roy 30/03/2026
Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Index No: 247/20/E

Response to the Reviewers' Comments on the Thesis titled "Efficient Resource Management Techniques for Cloud Computing Environment"

By

Sounak Banerjee

I sincerely thank both reviewers for their thorough and constructive evaluation of the thesis. Their comments and queries have been carefully considered and addressed in the revised manuscript. The responses are structured using the headings as specified by the reviewers, and each suggestion has been itemized according to the reviewer's sequence. Note that replies to individual comments are labeled with **"Reply"** and all changes made to the manuscript are labeled with **"Changes"**. The text of all changes incorporated into the revised thesis is presented in red to facilitate easy identification. It is my sincere hope that the revisions adequately address the concerns raised and have contributed to a stronger and more complete manuscript.

Response to Reviewer # 1

I would like to thank Reviewer 1 for the thorough reading of the thesis and for providing detailed, chapter-wise comments and suggestions. Concerning the queries and suggestions from Reviewer 1, the replies and changes to the former version of the thesis are as follows:

Chapter 1

(1) Comment: Chapter 1 makes it apparent why we need a unified way to estimate workloads, balance loads, and manage resources in a way that saves energy. As improvement, a short illustrative example or diagram can be added to make it clearer how these parts function together in the suggested unified framework.

(1) Reply: I agree that a concrete example makes the framework easier to understand. To address this, I have added a short illustrative example in Section 1.4.3. It walks through an e-commerce platform facing three situations — a traffic surge, peak load, and post-midnight idle period. The example shows how workload prediction anticipates the surge, how the load balancer distributes requests while respecting SLAs, and how VM consolidation then powers down idle servers to save energy. This step-by-step scenario makes it clear how the three components work together in a natural, connected flow.

(1) Changes: A short illustrative example has been added in Section 1.4.3 of the revised thesis.

"To better illustrate the interaction among the proposed components, consider a cloud-hosted e-commerce platform that experiences fluctuating traffic patterns throughout the day. During normal business hours the system operates under moderate load, but traffic rises sharply during a flash sale and drops to near-idle conditions after midnight. In the first phase, the Workload Prediction component analyzes historical traffic logs and temporal patterns to forecast the upcoming surge several minutes in advance, triggering proactive VM provisioning rather

Sounak Banerjee
30.03.2026

Sambaran Roy 30/03/2026
Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

than waiting for performance degradation to occur. As the surge begins, the QoS-Aware Load Balancer dynamically redistributes incoming requests across newly provisioned VMs, ensuring that checkout transactions — which carry strict latency SLAs — are prioritized over background analytics jobs. Finally, as traffic subsides post-midnight, the VM Consolidation module identifies underutilized physical servers, migrates residual workloads onto fewer active hosts, and powers down idle servers — directly reducing energy consumption without violating any active SLAs. This sequential interplay illustrates how workload prediction informs load balancing decisions, and how both together create optimal conditions for energy-efficient consolidation, forming a self-reinforcing, closed-loop resource management cycle."

(2) Comment: While industry-standard datasets are mentioned, the chapter could be slightly strengthened by adding a brief justification for the choice of these datasets and how they represent realistic cloud workload scenarios.

(2) Reply: I agree that explaining the dataset choices adds credibility to the research. In response, I have added a brief dataset justification at the end of Section 1.4.3. The Google Cluster Usage Traces and PlanetLab CPU traces are introduced here as the empirical basis for validating the proposed framework. Their selection is justified on the grounds of real-world origin, workload diversity, and recognition as standard benchmarks in cloud computing research.

(2) Changes: A brief dataset justification has been added at the end of Section 1.4.3 in the revised thesis.

"To validate this framework under realistic conditions, this research employs the Google Cluster Usage Traces [19] and the PlanetLab CPU traces [20] — both sourced from production-scale cloud environments. These datasets capture diverse workload types, bursty traffic patterns, and realistic usage variability, ensuring that the proposed techniques are evaluated against practical cloud scenarios and remain comparable with existing literature."

References:
[19] Google cluster data. https://code.google.com/p/googleclusterdata/, Dec 2020.
[20] PlanetLab workload traces. https://github.com/beloglazov/ planetlab-workload-traces/, Dec 2020."

Chapter 2

(1) Comment: The literature survey discusses several deep learning–based workload prediction approaches such as LSTM- and GRU-based models. Can you clarify why deep learning models were preferred over traditional machine learning techniques in certain surveyed works, particularly in handling long-term temporal dependencies?

(1) Reply: The choice of deep learning over traditional methods is rooted in the fundamental limitations of earlier approaches. ARIMA and time-series methods work well for short, stable patterns but fail when workloads are dynamic and unpredictable. Machine learning models like k-means and Extreme Learning Machines improved non-linear pattern capture; however, they still struggled with longer prediction horizons and required manual feature selection. Deep learning models — particularly LSTM and GRU — were adopted specifically to address these gaps. Their internal memory mechanisms allow them to retain and process historical context over extended time sequences. This is critical in cloud environments where workload demand at any given point is heavily influenced by patterns from hours or even days prior. LSTM achieves this through selective memory cells, while GRU-based Encoder-Decoder architectures extend it further by supporting multi-step forecasting and automatic trend learning.

(1) Changes: A clarifying passage has been added in the third paragraph of Section 2.2.1 of the revised thesis.

"Deep learning models were preferred over traditional approaches primarily because of their ability to handle long-term temporal dependencies. Methods like ARIMA rely on fixed-window assumptions and struggle with dynamic, non-linear workload patterns. Machine learning approaches improved on this but remained limited to shorter prediction windows. LSTM networks address this through memory cells that selectively retain past information over time, making them naturally suited for workloads where historical patterns strongly influence future demand. GRU-based models extend this further by enabling multi-step prediction while automatically learning trends and periodicity — without requiring manual feature engineering. Deep learning was therefore adopted where workload patterns were complex, non-linear, and temporally extended — conditions where shallower models consistently underperformed."

Chapter 3

(1) Comment: The chapter integrates prediction, placement, and energy modeling components in a sequential workflow. You may consider adding a short discussion on the sensitivity of the overall framework to each component, particularly how inaccuracies in workload prediction may affect the performance of the VM placement and energy optimization components.

(1) Reply: The concern about sensitivity propagation across framework components is well-founded. In response, I have added a short discussion at the end of Section 3.6.2, just before the performance evaluation begins. It explains how prediction inaccuracies can affect the heuristic-based placement through suboptimal resource allocation, and how they can distort payoff calculations in the game-theoretic approach, leading to suboptimal placement outcomes. The discussion also highlights the built-in mitigation mechanisms — periodic re-training in the ML approach and continuous parameter updates in the stochastic approach — that actively reduce this sensitivity. It closes by acknowledging residual vulnerability during abrupt workload changes, naturally motivating the dynamic load balancing mechanisms covered in Chapter 4.

(1) Changes: A sensitivity discussion has been added at the end of Section 3.6.2 in the revised thesis.

"While both placement strategies are designed to optimize resource utilization and energy efficiency, it is important to acknowledge that their effectiveness is directly tied to the accuracy of workload predictions. Inaccurate forecasts can cause the heuristic-based approach to allocate insufficient or excessive resources, potentially leading to host overload or unnecessary host activation. Similarly, for the game-theoretic approach, unreliable predictions distort the payoff calculations, pushing the Nash equilibrium toward a suboptimal placement configuration. To mitigate this, the ML-based prediction framework employs periodic re-training at every prediction window boundary, allowing models to adapt to evolving workload patterns. The stochastic framework counters this through continuous parameter updates using recent observations, keeping distribution estimates aligned with actual resource behavior. Together, these mechanisms reduce the downstream impact of prediction errors on placement quality and energy efficiency. However, some residual sensitivity remains — particularly during sudden workload bursts or abrupt behavioral shifts — which motivates the dynamic load balancing mechanisms discussed in the subsequent chapter."

(2) Comment: Could you clarify whether the statistics reported in Table 3.1 are computed over the entire simulation duration or over selected representative time intervals?

(2) Reply: I have added a short clarifying passage to the dataset description paragraph in Section 3.7.1.2. It explicitly states that the statistics in Table 3.1 are computed over the entire one-month observation period covering all 1,250 VMs at 5-minute intervals. This makes clear that the reported values reflect the full distribution of resource behavior across the complete trace and are not limited to any specific or representative time window.

(2) Changes: A clarifying passage has been added to the dataset description paragraph in Section 3.7.1.2 of the revised thesis.

"It represents aggregate descriptive statistics across the full dataset rather than any selected time window. This ensures that the reported values — including mean, minimum, median, maximum, and standard deviation — reflect the complete distribution of resource consumption behavior across diverse workload conditions. The wide gap between requested and actual resource usage is therefore a consistent characteristic observed throughout the entire trace."

(3) Comment: For Table 3.2, can you explain whether the reported power values are derived from empirical measurements, manufacturer specifications, or adopted from prior studies?

(3) Reply: The power values reported in Table 3.2 are adopted from the publicly available SPECpower benchmark database, as referenced in Section 3.7.1.1. These values represent standardized empirical measurements collected under controlled workload conditions across various server configurations. They are neither self-measured nor derived from manufacturer specifications alone, but are independently verified benchmark results widely adopted in cloud computing research for energy modeling purposes.

(3) Changes: No changes were required. The source and nature of the power values are already referenced in Section 3.7.1.1 of the thesis.

Chapter 4

(1) Comment: Chapter 4 presents a well-designed QoS-aware load balancing framework incorporating probabilistic overload detection and intelligent VM migration strategies. As a minor clarification, could you briefly explain how the chosen QoS parameters are prioritized when conflicting QoS requirements arise across different applications?

(1) Reply: The framework does not prioritize one application's QoS requirement over another. Instead, it focuses on resolving conflicts by satisfying all QoS requirements equally and simultaneously. The multi-criteria decision-making approach in Section 4.5.1 evaluates all VMs together through a single unified migration score, ensuring that no single application's service quality is favored during migration decisions. The weight assignment strategy considers all resource types jointly, ensuring the selected migration relieves overload without negatively affecting other applications. Additionally, the cumulative SLO violation tracking in Section 4.5.2 continuously monitors every VM's service compliance. Migration is triggered only when a VM is at risk of breaching its acceptable threshold — not because one application is more important than another.

(1) Changes: No separate addition was required. The mechanisms described above are already detailed in Sections 4.5.1 and 4.5.2.

(2) Comment: Chapter 4 presents the experimental results primarily through figures, which are clear and informative. As a minor enhancement, could you consider including a concise table summarizing the key QoS metrics (e.g., SLA violations, response time, migration count) across the compared load balancing methods to improve readability and quick comparison?

(2) Reply: I have added a summary table at the end of Section 4.8.3, along with a short descriptive passage introducing it. The table consolidates three key QoS metrics — migration count, response time overhead, and SLA violations — across all four compared methods for both PlanetLab and GCC traces. The descriptive passage explains how each metric is defined and derived, making the table self-contained and easy to interpret. All values are fully consistent with the quantitative findings reported throughout Section 4.8.

(2) Changes: A summary table and a descriptive passage explaining the table have been added at the end of Section 4.8.3 in the revised thesis.

Table 4.1: Summary of Key QoS Metrics Across Load Balancing Methods

Method	**Avg. Migration Count/hr**		**Avg. Response Time Overhead (ms/hr)**		**Avg. SLA Violations/hr**	
	PlanetLab	GCC	PlanetLab	GCC	PlanetLab	GCC
SLA-LB	162.08	132.33	1246	1024	54.25	43.12
RIAL	172.17	156.54	1288	1152	65.20	49.79
Sandpiper	198.83	194.17	1634	1498	118.29	85.25
CloudScale	234.37	217.79	1520	1368	61.58	46.29

"Table 4.1 summarizes the average QoS metrics across all evaluated methods for both workload traces. The average migration count per hour reflects the rate of VM migrations over the 24-hour simulation period, directly indicating system stability and migration overhead. The average response time overhead per hour is derived from the migration time component in Equation 4.29, where migration duration depends on the ratio of VM memory to available network bandwidth. The average SLA violations per hour are captured through the hourly rate of overloaded hosts, as host overloading directly leads to resource deficits and service level breaches. SLA-LB consistently achieves the lowest average migration count and response time overhead across both traces, reflecting the effectiveness of its SLO-aware VM selection. It also maintains the lowest average SLA violations per hour, while Sandpiper records the highest due to its purely reactive detection mechanism. Overall, SLA-LB delivers the best balanced performance across all three QoS dimensions simultaneously."

(3) Comment: While the trends are effectively illustrated using plots, you may add a small table capturing the average or cumulative values corresponding to the figures in Chapter 4, to complement the visual analysis and aid quantitative interpretation.

(3) Reply: In response to the previous comment, I have already added a summary table at the end of Section 4.8.3 consolidating migration count, response time overhead, and SLA violations across all four methods for both traces, directly corresponding to the trends in Figures 4.2 through 4.7. Regarding resource utilization efficiency, the average quantitative values are explicitly reported within the text of Section 4.8.4. Specifically, SLA-LB maintains approximately 68% CPU and 57% memory utilization on average, while the relative performance of RIAL, Sandpiper, and CloudScale is discussed with their corresponding utilization ranges. The consistent performance

hierarchy — SLA-LB > RIAL > Sandpiper > CloudScale — is clearly established across both CPU and memory dimensions for both traces.

(3) Changes: The summary table added at the end of Section 4.8.3 (see response to Question 2 above) addresses this concern. No additional tables are required, as quantitative values are already present in Section 4.8.4.

(4) Comment: In Figure 4.10 and Figure 4.11, could you indicate whether the utilization distribution represents steady-state behavior or includes transient effects during VM migrations?

(4) Reply: While the question specifically references Figures 4.10 and 4.11, the same observation applies equally to Figures 4.8 and 4.9, which present CPU utilization trends. All four figures capture both steady-state behavior and transient effects during VM migrations. As described in Section 4.7.1.3, the initial 6 hours serve as a training and calibration phase. Beyond this, transient dips naturally occur as migrations temporarily redistribute workloads across hosts. These effects are inherently embedded within the reported average utilization values.

(4) Changes: A short clarifying statement has been added at the end of the first paragraph of Section 4.8.4, immediately after the sentence introducing Figures 4.8 through 4.11.

"The utilization trends presented in these figures capture both steady-state behavior and transient effects that occur during VM migrations. These patterns reflect realistic dynamic cloud conditions observed throughout the full operational period, beyond the initial 6-hour training phase."

Chapter 5

(1) Comment: For the figures illustrating active host reduction (Figure 5.2 and Figure 5.3) and energy consumption (Figure 5.4 and Figure 5.5), you may consider adding a small summary table reporting the average or cumulative values corresponding to these figures to facilitate quick quantitative comparison.

(1) Reply: I agree that while Figures 5.2–5.5 capture hourly trends well, extracting average values for quick comparison is not straightforward from figures alone. In response, I have added a summary table at the end of Section 5.8.1. It reports the average number of active hosts per hour and the average energy consumption per hour for all four approaches. Both traces — Bitbrains and GCC — are covered in a single compact table. The average values are computed over the full 24-hour simulation and are fully consistent with the figures and the existing text.

(1) Changes: A summary table has been added at the end of Section 5.8.1 in the revised thesis.

"This observation is further supported by the results summarized in Table 5.5, which reports the average number of active hosts per hour and the average hourly energy consumption for all approaches across both workload traces."

Table 5.5: Summary of average active hosts per hour and average energy consumption per hour across Bitbrains and GCC traces

Approach	**Avg. Active Hosts/hr**		**Avg. Energy (kW/hr)**	
	Bitbrains	GCC	Bitbrains	GCC
RO-VMC	18.62	6.12	27.2	14.57
PEAS	23.72	10.25	33.07	19.60
AFED-EF	29.12	14.8	38.67	23.27
EQ-VMC	34.95	19.75	42.47	27.47

(2) Comment: In Table 5.2, could you clarify whether the VM configurations are fixed throughout the experiments or dynamically adjusted during the consolidation process?

(2) Reply: The VM configurations defined in Table 5.2 are fixed throughout all experiments. They are not dynamically adjusted during the consolidation process. Each VM type retains its allocated CPU, memory, and storage capacity for the entire simulation duration. Only the mapping of VMs to hosts changes — not the VM configurations themselves. This is a standard assumption in VM consolidation studies, where resource capacities are pre-assigned at VM creation time, and the consolidation process operates solely on VM placement decisions.

(2) Changes: A clarifying sentence has been added in Section 5.7.1.1 of the revised thesis.

"It directly states that VM configurations remain fixed throughout the experiments and are not altered during the consolidation process."

(3) Comment: In Table 5.3, you may indicate the time granularity or observation window over which the workload statistics are computed.

(3) Reply: In this study, the data center resource state is updated every 5 minutes. The workload statistics reported in Table 5.3 — mean and standard deviation of CPU, memory, and disk utilization — are computed over these 5-minute interval readings. For the Bitbrains trace, the statistics are derived from 1,000 randomly selected workloads per day over 10 consecutive days, covering approximately 1,200 VMs daily. For the GCC trace, they are derived from 200,000 randomly selected tasks from day 18 of the trace. This time granularity is directly tied to how frequently the RO-VMC framework detects load imbalances and triggers consolidation decisions, and is already discussed in detail in Section 5.7.1.2.

(3) Changes: No explicit addition was required, as the relevant time granularity context is already present in Section 5.7.1.2.

Response to Reviewer # 2

I would like to thank Reviewer 2 for the careful and insightful evaluation of the manuscript. The queries and

suggestions raised by Reviewer 2 are primarily conceptual in nature and pertain to the methodological choices made across multiple chapters. Concerning these queries and suggestions, the replies and changes to the former version of the manuscript are as follows:

(1) Comment: Agglomerative Hierarchical Clustering combined with Dynamic Time Warping is known to have high computational and memory complexity. How does your proposed approach address scalability, and how feasible is it for large-scale or near real-time workload characterization?

(1) Reply: I fully acknowledge that AHC combined with DTW carries well-known computational concerns — specifically $O(N^2)$ space complexity for the pairwise distance matrix and $O(N^2 \log N)$ time complexity for the clustering process. These are valid concerns that deserve careful clarification.

First, workload characterization through AHC-DTW is performed only once — during the offline training phase — and not at runtime. The clustering is applied to historical resource usage traces to identify distinct workload patterns. Once clusters are formed and cluster-specific prediction models are trained, the characterization step is not repeated during live system operation. The computational overhead of AHC-DTW therefore has no bearing on the near real-time responsiveness of the framework. Second, the workload traces used in this study represent long-running business-critical tasks observed over a period of months. The nature of such workloads is inherently stable, greatly reducing the need for periodic re-clustering. This is an important practical advantage that further limits the frequency at which the expensive AHC-DTW computation needs to be invoked. Third, for larger-scale deployments, several strategies consistent with the framework's modular design can be employed: (a) batch or mini-batch processing, as DTW distance computations are embarrassingly parallelizable across CPU cores or distributed computing nodes; (b) DTW lower-bound approximations such as `LB_Keogh`, which can prune a large fraction of unnecessary distance computations while preserving clustering quality; and (c) incremental assignment for new VMs, where new VMs arriving at runtime can be assigned to the nearest existing cluster using a lightweight nearest-centroid classification step, bypassing the full AHC-DTW pipeline entirely for online operations.

(1) Changes: A clarification regarding the offline design choice, the time-consuming nature of AHC-DTW, and the incremental assignment strategy for new VMs has been briefly added in Section 3.5.1.1 of the revised thesis.

"It is important to note that AHC is computationally expensive, as it requires computing and storing pairwise distances between all data points. This makes it time-consuming for large-scale datasets. However, in this work, the clustering is performed only once, offline, using historical workload traces. It is not repeated during live system operation. Moreover, the workloads considered in this study are long-running tasks with stable resource consumption patterns. Their behaviour does not change significantly over time. This further reduces the need for re-clustering. For new VMs arriving at runtime, a simple nearest-centroid classification is used to assign them to the closest existing cluster. This avoids running the full AHC pipeline online, keeping the framework efficient and feasible for large-scale deployments."

(2) Comment: What motivated the use of the Silhouette index as a cluster validation metric?

(2) Reply: The choice of the Silhouette index as the cluster validation metric was motivated by several practical and methodological reasons specific to this study.

The primary motivation is that the Silhouette index is an internal validity measure — it does not require any ground truth labels. This is particularly suitable here, as workload characterization is an unsupervised task where no predefined class labels exist for VM resource usage patterns. The Silhouette index evaluates two complementary

aspects simultaneously — intra-cluster cohesion and inter-cluster separation — directly aligning with the goal of workload characterization, where well-separated and compact clusters are essential for building accurate cluster-specific prediction models. Additionally, the Silhouette index is naturally compatible with the DTW-based pairwise distance matrix already computed during the AHC process, making it a computationally convenient and consistent choice within the existing framework. Finally, the Silhouette index provides an interpretable score in the range of -1 to 1, making it straightforward to determine the optimal number of clusters, as clearly demonstrated in Figure 3.2.

(2) Changes: No separate addition was required. The motivation and usage of the Silhouette index are already discussed in Section 3.7.1.3 of the thesis.

(3) Comment: Why does effective load balancing lead to resource fragmentation, and how does this fragmentation impact overall energy consumption in cloud data centers?

(3) Reply: As discussed in Chapter 4, effective load balancing distributes VMs across multiple hosts to maintain balanced resource utilization and prevent overloading. While this achieves its primary goal of service quality preservation, it does so by spreading VMs broadly across the available host pool. As a result, many hosts end up operating at moderate utilization levels rather than being densely packed. This spreading effect leads to resource fragmentation, where available resources are distributed across many partially utilized hosts instead of being consolidated onto a smaller number of well-utilized ones. Since each host continues to consume power while running, this situation leads to considerable cumulative energy waste. This is precisely the motivation behind Chapter 5. The RO-VMC framework is designed as a complementary layer that operates on top of the Chapter 4 solution. It takes the fragmented host state produced by load balancing as its starting point and systematically consolidates it — evacuating underloaded hosts, powering them down, and redistributing their VMs onto a minimal number of active hosts with balanced resource utilization. Experimental results confirm this effectiveness: RO-VMC reduces active hosts by 21–65% and energy consumption by 17–46% compared to existing approaches.

(3) Changes: No changes were required, as this conceptual motivation is already elaborated in Chapter 5 of the thesis.

(4) Comment: Why was stochastic load imbalance detection chosen over deterministic threshold-based methods for identifying consolidation opportunities?

(4) Reply: The choice of stochastic load imbalance detection over deterministic threshold-based methods was motivated by the inherently uncertain and dynamic nature of cloud workloads.

Deterministic threshold-based methods classify a host as overloaded or underloaded based on whether its current resource utilization crosses a fixed threshold at a given instant. While simple, this approach reacts only to the current observed utilization value and does not account for variability or uncertainty in future resource demands. In dynamic cloud environments, a host may appear safe at one instant but become overloaded shortly after due to a sudden demand spike. Conversely, a host may temporarily drop below the underload threshold due to a brief reduction in demand, triggering unnecessary migrations that hurt QoS. Deterministic methods cannot distinguish between these short-term fluctuations and genuine consolidation opportunities. Stochastic detection addresses this directly. As described in Section 5.2.1, each VM's resource demand is modelled as a normal random variable with an estimated mean and variance. Host-level load is characterized as an aggregate distribution, and overloading and under-loading probabilities are computed analytically. Consolidation decisions are thus based on the likelihood of

a host becoming overloaded or underloaded — not just its current utilization value. Furthermore, the thresholds ω_{over} and ω_{under} directly control the acceptable probability of overloading or underloading, providing a principled mechanism for balancing consolidation aggressiveness with system stability — a level of control that deterministic methods simply cannot offer.

(4) Changes: No changes were required. The rationale for stochastic detection is already elaborated in Section 5.2.1 of the thesis.

I am grateful to both reviewers for their time and effort in providing these detailed and constructive comments. All suggestions have been addressed carefully and transparently. The revisions have, in my view, meaningfully strengthened the thesis, and I hope that the responses provided herein adequately satisfy the reviewers' concerns.

Sarbani Roy 30/03/2026

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Sounak Banerjee
30.03.2026