Dynamic Management of Virtual Machine and Container Scheduling in Multi-Cloud Data Centers

Mohammad A. Altahat

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

April 2024

Abstract

Dynamic Management of Virtual Machine and Container Scheduling in Multi-Cloud Data Centers

Mohammad A. Altahat, Ph.D.

Concordia University, 2024

Efficiently managing virtual resources is a critical component of server virtualization technology. The scheduler is crucial in strategically distributing Virtual Machines (VMs) and containers across diverse computing nodes, responsible for the allocation and the placement of VMs and containers on different computing nodes, and the migration of deployed ones between different nodes. In this thesis, we propose novel solutions in scheduling virtual resources, particularly in the management of VMs and containers deployed across multi-data center cloud environments. The proposed solutions leverage mathematical models, machine learning techniques, and blockchain technology to optimize scheduling decisions, enhance server consolidation, minimize energy consumption, and secure container scheduling. We introduce mathematical models for live VM migration techniques used in simulating and studying live VM migration in cloud systems environments. We present a novel distributed scheduling model that leverages blockchain technology to facilitate efficient sharing of VM status across multiple data centers. This enables prompt Local Area Network (LAN) or Wide Area Network (WAN) scheduling decisions for VMs. Additionally, we employ machine and deep learning techniques in a VM migration prediction service to identify the most suitable live migration method for each VM based on its unique characteristics. Our blockchain-based model reduces the total messages exchanged for the VM migration with percentages ranging from 0.5% to 22% and the total communication delay by 8% to 72% compared to a REST-based distributed model. The proposed blockchain-based distributed model also reduces the number of communication messages by 41.79% to 49.85% and total delay by 2% to 12% compared to a VPN-based centralized model. The Service Level Agreement (SLA) compliance rate of the proposed VM migration prediction service ranges from 18% to 94.9% for different machine learning algorithms and SLA policies. The proposed solution reduces the total migration time by 14% to 79% and the downtime by 64% to 99%. Furthermore, we present a novel two-stage container scheduling solution that addresses node imbalances and efficiently deploys containers as an optimization problem, integrating various objective functions and constraints to enhance server consolidation and minimize energy consumption. The confidentiality of migrated containers is ensured through encryption, and the associated costs of the proposed attributes-based encryption model are incorporated into the optimization constraints. The proposed solution's efficacy is demonstrated in its ability to efficiently deploy containers in multi-data center cloud environments and seamlessly migrate them between hosts within the same data center or across different data centers. The results show optimal consolidation with a reduction in the number of running hosts, ranging from 4% to over 18%. Additionally, the solution promotes minimal total power consumption with savings ranging from 3.5 to 16.25 megawatts, while also ensuring balanced server loads, highlighting the effectiveness of the proposed container scheduling approach.

Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful.

All praise is due to Almighty Allah, who has bestowed His guidance, strength, and mercy upon me throughout this challenging yet rewarding journey of completing my PhD. Without His divine will and blessings, this thesis would not have reached its completion.

I extend my deepest gratitude to my esteemed thesis supervisor, Dr. Anjali Agarwal, whose guidance, wealth of knowledge, and unwavering support have been instrumental in my academic success. Her frequent insights and enduring patience have been a source of inspiration, and I am truly proud of the accomplishments we have achieved together.

I would like to express my sincere appreciation to the members of my advisory thesis committee for their valuable contributions and insights that have enriched the depth and quality of this work.

My heartfelt thanks go to my family for their unwavering support and encouragement throughout my academic endeavors. To my mother and father, thank you for your endless love, prayers, and sacrifices that have laid the foundation for my success. To my beloved wife, your patience, understanding, and constant support have been my anchor in times of difficulty. To my children, your smiles and affection have been a source of joy and motivation, reminding me of the purpose of my hard work. To my siblings, your encouragement and belief in me have been invaluable. And to my friends for their encouragement and camaraderie throughout this journey.

This journey has been a collective effort, and I am profoundly thankful to everyone who has been part of this significant chapter in my life.

Contents

List of Figures

List of Tables

List of Acronyms

ABAC	Attribute-Based Access Control
ABE	Attribute-Based Encryption
ACO	Ant Colony Optimization
AES	Advanced Encryption Standard
AI	artificial intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CBR	content-based redundancy
CLT	Central Limit Theorem
CPU	Central Processing Unit
CP-ABE	Ciphertext-Policy ABE
CNN	Convolutional Neural Network
CR	Checkpoint and Restart
DES	Data Encryption Standard
DFF	Discrete Firefly Algorithm
DNS	Domain Name System
DRDB	Distributed Replicated Block Device
DRL	Deep reinforcement learning
DRM	Dynamic Resource Management
EAs	Evolutionary algorithms
ECC	Elliptic-curve cryptography
EDR	Edge Demand Response
FF	first fit
FFD	First Fit Decreasing
GA	genetic algorithm
GPU	graphical processing units
HTTP	HyperText Transfer Protocol
ILP	Integer linear programming
IO	Input Output
IoT	Internet of Things
IPS	instructions per second
KDE	Kernel Density Estimation
KNN	K-nearest neighbor
KNNR	K-nearest neighbor regression
KP-ABE	Key Policy Attribute Based Encryption
LAN	Local Area Network
LRA	Long-Running Application
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MILP	mixed-integer linear programming
ML	Machine Learning
MLP	multi-layer perceptron
MSE	Mean Square Error
NFS	Network File System
NIC	Network Interface Card
NIST	National Institute of Standards and Technology

NLP	natural language processing
NN	Neural Network
OS	Operating System
PDF	Probability Density Function
PSO	Particle Swarm optimization
QLDB	Amazon Quantum Ledger Database
QoE	Quality of Experience
QoS	Quality of Service
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RPS	requests per second
RSA	Rivest–Shamir–Adleman
SEV	Secure Encrypted Virtualization
SFTP	Secure File Transfer Protocol
SLA	Service Level Agreement
SSDs	Solid State Disks
SSH	Secure Shell
SVM	support vector machine
SVR	support vector regression
TCP	Transmission Control Protocol
VCG	Vickrey-Clarke-Groves
VM	Virtual Machine
VMM	Virtual Machines Manager
VPN	Virtual Private Network
WAN	Wide Area Network

Chapter 1

Introduction

With the continuous development of servers' virtualization technology, the importance of live Virtual Machine (VM) migration has increased to lead to better hardware utilization, increase the power efficiency of data centers, and decrease the effect of data center failures and service downtime in maintenance. Virtualization platforms include VM based platforms and container-based platforms. Containers can run on bare metal servers or in virtual machines. The migration of VMs and Containers is one of the key strategies in achieving server consolidation and reducing energy consumption which minimizes the overall operating costs of data centers.

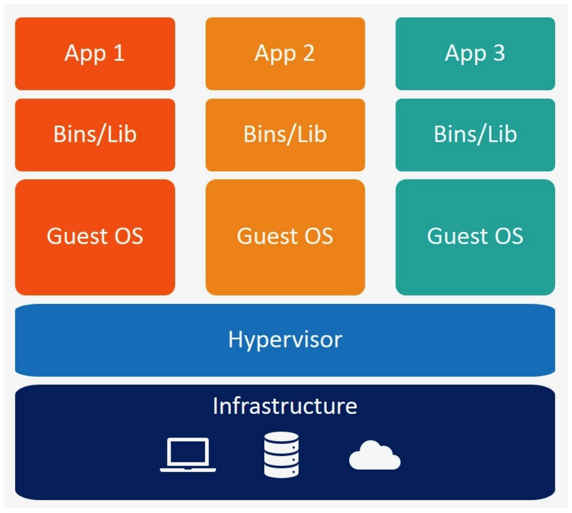
VM is a software that runs its Operating System (OS) and applications as if it is a physical computer. It behaves like a physical computer and has a virtual Central Processing Unit (CPU), Memory, Disk, and Network Interface Card (NIC) [1]. Live VM migration is the process of copying the CPU state and the memory and disk of a VM from a hosting server to another destination server at the same data center, or to another data center connected over LAN or WAN, with the least service downtime, in order not to affect the Quality of Experience (QoE) of virtualization service users.

A container is a full runtime environment of an application and its dependencies. Dependencies include codes, binaries, system libraries, and any configuration files the application needs to run [2]. A container engine must be hosted on the machine for the container to work. Docker, LNX, OpenVZ, and Containerd are examples of known container engines. Containers can run on bare metal servers or in virtual machines. A container is split into the base image layer, which consists of the binaries and libraries needed by the container to run, and the application layer [3]. Examples of known base images are Ubuntu, Alpine, and Red Hat.

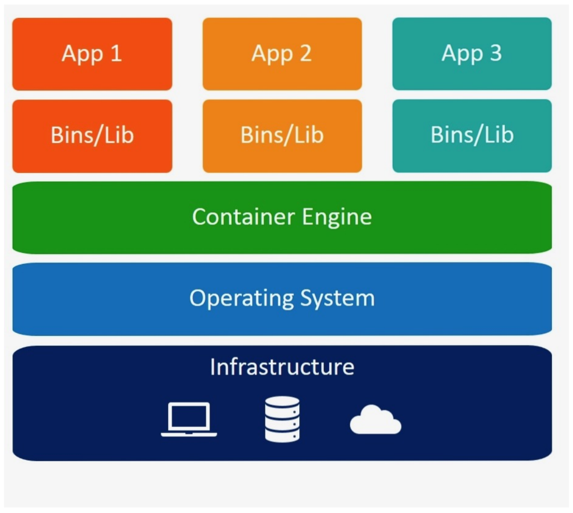
Container is a running instance of both the application or microservice and the base image. The base image must be present on the host machine so the container can run. If different container applications have similar dependencies, one base image can be used by different containers. This architecture for container-based platforms allows containers to be deployed, migrated, terminated, and replicated in milliseconds, which helps improving the flexibility and scalability of computing systems [4]. Figure 1.1 shows the architecture for VM and container virtualization platforms and how they are stacked in a host machine [5].

Because VMs serve live clients, live VM migration must be seamless for the clients and should not affect their experience. The VM CPU, memory, and disk states are always changing, even during live migration. The best way to transfer the VM states, while continuing to run the VM and its applications, is done by iteratively sending the memory changes to the destination host while the VM continues running either on the source or the destination hosts. Pre-copy [6] and post-copy [7] are two known classical live VM migration approaches.

Container orchestration is the process of managing containers running on different computing nodes. Container orchestration is still being developed, especially for heterogeneous cloud environments like edge computing [8]. The scheduler is a key component of container orchestration that



(a) Virtual Machines.



(b) Containers.

Figure 1.1: Virtual Machine vs. Container virtualization architecture.

deploys container applications on computing nodes, it decides how computing requests are mapped to containers and how containers are mapped to computing nodes complying with a set of objectives and constraints.

In this thesis, we propose solutions to enhance the performance of cloud computing systems in both VM-based and container-based environments. We present solutions that speed up the VM migration process by selecting the best VM migration method to migrate a specific VM. Selecting a migration method is a function of many parameters like the characteristics of the VM, the workload type of the applications running on the VM, the status of source and destination hosts, and the SLA requirements. Then, we propose a new solution for the allocation and scheduling of containers in cloud platforms by using optimization methods to efficiently allocate containers to computing nodes and to migrate the already allocated containers to new nodes to achieve better server consolidation and less energy consumption. The container data security is incorporated into the solution by considering the containers encryption overhead in the optimization objective functions.

1.1 Problem Statement and Motivation

The general problem of this thesis is to improve the process of live VM migration in VM-based cloud and edge systems, and the allocation and scheduling of containers in container-based cloud systems. These improvements will enhance the migration and scheduling performance with minimal service downtime and total migration time and will efficiently deploy containers to computing nodes and migrate them between different nodes with minimum costs.

One of the main challenges when migrating VMs across different data centers is the need to transfer VM's data over WAN links that have limited bandwidth compared to LAN links inside the data center itself. VM migration via WAN will increase the total migration time and the downtime, in addition to the more bandwidth needed for transferring more amounts of data [9].

Different migration techniques have different performances when migrating a VM. Selecting the best VM migration method to migrate a certain VM based on a set of parameters when migrating the VM to a different data center over WAN networks can help in migrating that VM with minimal downtime and total migration time and reduce the amount of transferred data, based on the SLA requirements for that migration process.

CloudSim [10, 11] is one of the popular open-source cloud simulators. It provides a framework to model and simulate cloud computing infrastructures and services. CloudSim simulates the VM migration process by using the trivial Stop-and-Copy migration method, which stops the VM at the source host, copies the whole data once to the destination host, and starts it there. Existing CloudSim extensions do not provide support for different migration methods used in real cloud systems such as Pre-copy, Post-copy, and their enhancements.

Cloud platform providers employ virtualization in their data centers and services. Selecting the best VM migration method to migrate a specific VM is an important problem. Selecting a migration method is a function of many parameters like the characteristics of the VM, the workload type of the applications running on the VM, the status of migration source and destination hosts, and the SLAs with customers.

Efficient virtual resource management and scheduling are essential components of optimizing cloud data center utilization while minimizing costs and efforts. Maintaining data center stability is vital to reduce management and operational expenses. Achieving this requires a comprehensive view of the status of all data centers and the use of intelligent and seamless scheduling techniques. By analyzing global and local migration data for virtual machines, A comprehensive understanding of data center status is gained, enabling more accurate VM scheduling and migration decisions. The availability of precise and up-to-date status information is crucial in facilitating accurate, swift, and secure scheduling and migration decisions for VM management across data centers. This, in turn, has significant implications for enhancing server consolidation and load balancing, ultimately optimizing the allocation of computational resources.

In centralized cloud management models, consistency and mutual execution in scheduling techniques can be maintained through a global manager that gathers status information from all data centers and stores it in a central database. This can be accomplished through periodic data pulls or local managers pushing the information. Once the necessary data is collected, scheduling algorithms are utilized to determine the best course of action. However, this approach presents specific challenges that must be addressed to ensure optimal performance when dealing with large distributed data models. These challenges include potential communication and management issues, single points of failure, security concerns, and contention access due to queued transaction updates and refreshes [12]. It is vital to thoroughly evaluate these factors and implement appropriate measures to mitigate any associated risks.

Centralized management may not be the optimal or secure approach to manage tasks and maintain data consistency among different cloud data centers. Distributed management models offer a promising alternative to centralized models, as they can enhance resource allocation and decrease communication overhead [13]. A distributed cloud management model involves each data center having a duplicate of the status of all other centers. The cloud manager ensures that the duplicate copy is consistent with the latest version of VMs and data center status across all centers. Additionally, the cloud manager enables direct communication between data centers to facilitate scheduling operations across data centers.

Numerous mathematical models have been introduced to analyze various live migration methods over time [14, 15, 16, 17]. These models are indeed valuable in investigating cloud system performance before and after migration. However, they do not consider all the migration characteristics and performance metrics that can impact the migration process. Therefore, they do not provide precise performance predictions for these migration methods. Additionally, these models overlook the importance of SLA requirements, which are fundamental in managing real cloud systems.

Machine learning algorithms are commonly used for scheduling and allocating cloud resources. Machine learning models can be utilized to create regression models for VM migration performance by considering additional migration parameters and SLA constraints that cannot be mathematically modeled. The accurate prediction of performance for different migration methods can facilitate the selection of the best technique for a VM with specific characteristics, which is particularly critical in low bandwidth environments like WAN to reduce migration time, downtime, and data transfer.

Neural networks can automatically learn hidden features from input data before computing an output value, and established algorithms exist for finding the optimal internal parameters (weights and biases) based on a training dataset. They can learn and model the relationships between input and output data that are nonlinear and complex. By adding deep layers, the model can give more accurate decisions and predictions [18]. Convolutional Neural Network (CNN) can recognize patterns in data, making it a better option for image processing where feature extractions are most needed, but it can also be used for data regression. CNN adds convolution layers to the neural network that consider the correlation between multivariate variables, reduce the data size, and pass only the main features to the neural network layers, which help significantly reduce the training time [19]. Random forests consist of an ensemble of decision trees, each trained with a random subset of the training dataset. This method corrects the tendency to overfit the input data and archives better prediction accuracy [20].

The migration of VMs and containers in Dynamic Resource Management (DRM) of cloud computing systems is a critical factor in reducing data center energy consumption, which reduces total data center operating costs and limits their impact on climate change. Scheduling seeks a solution that meets a specific objective function, which can be translated into an optimization problem that can be mathematically solved. Optimization modeling is a well-known method for simulating complex problems by employing analytical functions. Optimization problems are made up of several decision variables, constraints, objective functions, and model assumptions. The objective function in the container placement problem is related to goals such as minimizing latency, minimizing the overall cost of providing services, minimizing total energy used by the server, minimizing load imbalance, and maximizing SLA satisfaction.

1.2 Objectives and Contributions

This thesis is divided into three main parts: 1) Modeling and Analysis of VM Migration Methods in Cloud Environments, 2) Improving VM Migration Across Cloud and Edge Data Centers, and 3) Optimizing Secure Scheduling of Cloud Containers. The objectives and contributions of each part are summarized as follows:

1. Modeling and Analysis of VM Migration Methods in Cloud Environments

In this part of the thesis, we present the mathematical models for the classical live VM migration methods Pre-copy, Post-copy, and Hybrid-copy and the proposed Dynamic Hybrid-copy that combines both Pre-copy and Post-copy. We present comparisons between different migration methods for different connection speeds and different VM metrics. These mathematical models are valuable to study the performance of cloud systems before and after migration. The contributions of this part of the thesis are as follows:

(1) Enhancing the cloud computing frameworks by providing the support of different VM migration approaches through mathematical modeling of these approaches.

(2) Proposing a new dynamic hybrid migration method solution that selects the best number of iterative copy iterations in the Pre-copy phase of the migration method to achieve the minimum migration time and downtime and copy the least amount of data over WAN connections.

(3) Because different migration methods have different performances for different environments and VM characteristics, once the VM is selected to be migrated, the mathematical models are used to select the best migration method to use to migrate a VM with the least migration time and downtime and minimum amounts of transferred data.

(4) Presenting a new extension of the CloudSim simulator that supports different VM migration methods instead of the stop-and-copy methods the simulator uses. After that, a recommendation system will be built to recommend the best migration method to use for each VM chosen as a candidate to be migrated.

2. Improving VM Migration Across Cloud and Edge Data Centers

The main focus of this part of the thesis is to improve the live VM migration process by utilizing machine learning algorithms to accurately predict the performance of VM migration. This prediction then helps selecting the best migration method that minimizes downtime and total migration time, especially over limited WAN links. The objective of this solution is to make the overall migration process as seamless and efficient as possible.

A novel distributed cloud management model is built using blockchain technology. The blockchain nodes are distributed across cloud data centers and serve as a registry for securely and efficiently sharing data center status information. The recent and accurate status of data centers helps make accurate, fast, and secure scheduling and allocation decisions for VMs management across data centers and achieve better server consolidation and load balancing.

The contributions of this part of the thesis are as follows:

(1) Design and model a novel blockchain-based distributed data center management framework, which considers all VM scheduling and migration aspects. Blockchain is integrated with the cloud manager of data centers to securely and authentically share the status of VMs and hosts across data centers. The shared information is used for scheduling and allocating VMs across data centers.

(2) A REST-based distributed cloud management model and a VPN-based centralized cloud management model, that is based on the definitions of centralized systems, are implemented to compare with our novel blockchain-based distributed model.

(3) Machine and deep learning algorithms are applied to accurately predict the migration performance metrics of VMs. Artificial neural networks (ANN), Convolutional Neural Networks (CNN), and Random Forests regression models are used for more accurate prediction.

(4) A VM migration method selector is built by using the predicted performance metrics and considering predefined SLA policies that mathematical models do not consider.

(5) The prediction service is integrated over the novel blockchain-based distributed cloud management model for secure and authentic information sharing across multiple data centers and edge nodes. This, in turn, has significant implications for enhancing server consolidation and load balancing, ultimately optimizing the allocation of computational resources in cloud and edge environments.

3. Optimizing Secure Scheduling of Cloud Containers

In this part of the thesis, we tackle the pressing issues surrounding scheduling virtualized cloud resources. Our proposed solution aims to optimize the placement of containers on computing nodes. We can achieve superior server consolidation and balance by migrating containers to different nodes. To ensure the confidentiality of migrated containers, we apply encryption and factor in its overhead as an objective function of the scheduling process. In summary, the contributions of this part of the thesis are outlined as follows:

(1) Studying the performance of the common and recent data hybrid encryption schemes: AES-RSA and AES-ECC.

(2) Defining a set of security levels and security SLA policies based on the complexity of encryption requirements demanded for cloud containers.

(3) Proposing a novel solution for the allocation and scheduling of containers in cloud platforms. The use of a multi-objective optimization model is proposed to allocate containers efficiently to computing resources and to migrate the already allocated containers to new hosts for better load balancing, server consolidation, and less energy consumption.

(4) Considering the affinity of containers as an optimization constraint to place all dependent containers on the same host to improve performance and efficiency.

(5) Incorporating container data security into the optimization objective functions, diligently focusing on encryption measures. To ensure this, we carefully evaluate the encryption computational overhead relative to the data center machine's capabilities and capacity. We thoughtfully choose encryption algorithms and key sizes, informed by a combination of theoretical and statistical methodologies. By utilizing that, we ensure a comprehensive evaluation of the security landscape within the optimization framework.

Chapter 2

Background and Literature Review

This chapter explains the various tools, protocols, simulators, and algorithms that have been employed in the development of the thesis research. In addition, a comprehensive literature review is presented in this chapter, which sheds light on the key research findings and insights that have been previously established in the research domain.

2.1 Background

2.1.1 Classical VM Migration Methods

A. Pre-copy

The pre-copy approach firstly copies all memory pages to the destination host while the VM continues to run on the source host in a phase called warm-up phase [6]. After that, while the VM runs on the source host, the pre-copy method iteratively copies the modified memory pages of the VM to the destination host and stops after reaching a limit that is defined by iterations number threshold or having a consistent memory state of the VM at the destination host. The highly modified memory pages are copied to the destination host along with the VM CPU state after stopping the VM at the source host in a phase called stop and copy phase and then resuming the VM to continue running at the destination host.

Figure 2.1 shows the timeline of the pre-copy migration processes. At stages 0 and 1, while the VM is still running on the source host, a destination host is selected and the VM is prepared for

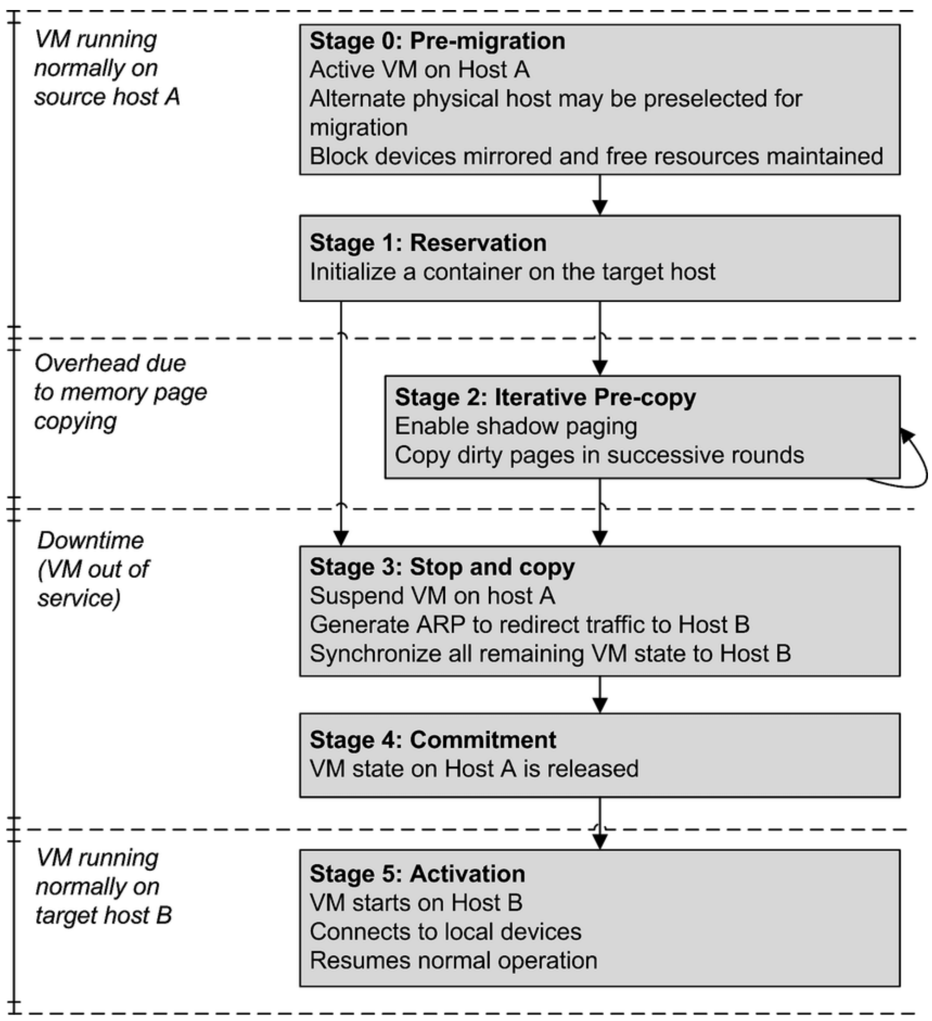

VM running normally on source host A
Stage 0: Pre-migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to memory page copying
Stage 2: Iterative Pre-copy
Enable shadow paging
Copy dirty pages in successive rounds
Downtime (VM out of service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on target host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation

Figure 2.1: Pre-copy migration process timeline.

the migration process by releasing its allocated resources and devices. At stage 2, the iterative copy process of the dirty memory pages will start, which causes more overhead to the running VM due to the copying process. At stages 3 and 4, the VM at the source host will be paused, the CPU status and the rest of the dirty memory pages will be copied to the destination host, and then the network traffic will be directed to the new VM location at the destination host, in order for the VM to resume its operation at the new host at stage 5.

Some applications that run on the migrated VM have more memory page dirtying rate than others, in the case of not reaching a consistent memory state at the destination; the iterations will stop, the VM will be paused, and all the remaining dirty pages will be copied to the destination host. If the VM memory has a high dirty rate then the remaining pages at the stop and copy phase will be higher, resulting in a higher downtime. The pre-copy method can be used to migrate VMs through fast network connections, especially in LAN connected servers, where the bandwidth could be high enough to transfer large amounts of data in a short time, but adopting this method alone without further improvements when migrating through lower bandwidth networks, like WAN, will be inefficient and results in high service downtime and network burden. When migrating the memory using the pre-copy method, the memory migration time will depend on many factors like the virtual memory size, the allocated bandwidth, and the memory dirtying rate.

B. Post-copy

The migration process in post-copy starts by stopping the VM, copying its CPU state and any other execution states needed for the VM to start working on the destination host, and then starts fetching memory pages while the VM is running on the destination host.

Figure 2.2 shows the difference between the Pre-copy and Post-copy methods [7]. Figure 2.2 (a) describes the Pre-copy iterations and the Downtime (stop and copy) phase. Downtime in Pre-copy goes at the end of the migration process after the last iteration, and it is the delay of copying the CPU state and the remaining dirty memory pages after finishing the copy iterations.

The Post-copy phases are shown in Figure 2.2 (b). The downtime (stop and copy) phase goes at the beginning of the migration process. It is the delay of copying only the CPU state and other

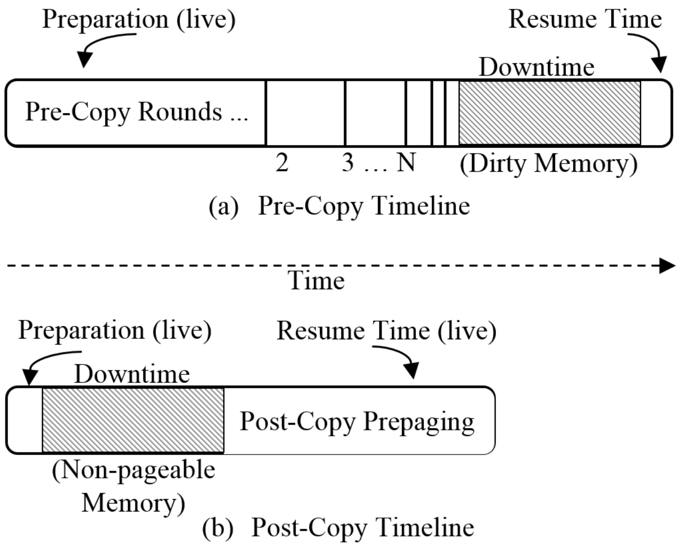


Figure 2.2: Pre-copy and Post-copy timeline comparison.

states required for the VM to run at the destination host. Then it is followed by fetching the memory pages from the source host to the destination host while the VM runs at the destination host.

The technique used to fetch memory pages gives different variants of post-copy. For example, memory pages can be transferred from the beginning to the end of memory with every page being sent only once. Another technique is to request the memory pages from the source host when they are needed. The three main memory fetching techniques used with the classical Post-copy are: The Demand Paging, the Active Pushing, and the Pre-Paging [21].

1) Demand-Paging: This technique works by processing every memory access for a memory page that has not been fetched yet at the destination host as a page fault, and then this page fault is serviced by requesting it over the network from the source host. The network latency slows down the VM and decreases the performance of the VM being migrated. Also, this technique will increase the residual dependencies on the source host for an unknown duration. Although this technique copies each memory page only once to have less data being migrated, it is unacceptable to be used because of its lower performance and high dependability on the source host.

2) Active Pushing: This technique works by pushing the memory pages to the destination host where the VM runs there and serving any page fault in parallel via demand paging. This will decrease the duration of source host dependency by pushing memory pages in advance to the memory access, and at the end, each page will be copied only one time, either by active pushing or demand paging.

3) Pre-Paging: This technique estimates the spatial locality of the memory access fault to predict a faulting pattern and push memory pages following that pattern. A window of the pages to be transmitted is used, with the page that caused the fault is located within this window. With the spatial locality concept, this technique will reduce next page faults by copying in advance the pages that may cause faults in the near future.

The performance of the pre-paging algorithm is measured by the percentage of the page faults over the network, the fewer page requests from the source, the better performance. The key is in predicting the pages that might be accessed in the near future and pushing them to the destination host in advance to decrease the demand paging over the network.

2.1.2 CloudSim Simulator

CloudSim [10] is a popular open-source cloud simulator that provides a framework for modeling and simulating cloud computing infrastructures and services. It has classes for data centers, computing resources, virtual machines, apps, and a variety of other cloud resources. It enables cloud developers to test and manage many aspects of the cloud system, such as scheduling and provisioning, in a free simulation environment before applying it to real-world systems. [22]. It allows researchers and industry-based developers to test the performance of newly developed applications and focus on specific system design issues they want to investigate without having to worry about the low-level details of Cloud-based infrastructures and services. [11].

CloudSim is a tool that allows simulating data centers, service brokers, scheduling, and load balancing policies. It supports the modeling and simulation of large-scale Cloud computing data centers, as well as network connections, with the ability to switch between space-shared and time-shared allocation. CloudSim also supports user-defined policies for provisioning and allocating host resources to virtual machines, as well as modeling and simulation of virtualized server hosts. It is simple to analyze new cloud-related strategies using these components, taking into account policies, scheduling algorithms, and load-balancing policies. It may also be used to evaluate the effectiveness of approaches from different perspectives, such as cost and application execution time.

2.1.3 Neural Networks for Regression and Classification

Artificial Neural Network (ANN) or Neural Network (NN) is a machine learning system that conducts neurons (nodes) and weights on the edges linking the neurons. The neural network learns to perform tasks by analyzing training examples known as training data set. The difference between computed and real outputs, or simply the error, is used to iteratively change the network's weights to produce more accurate outputs. Regression analysis and prediction, classification and pattern recognition, filtering, and clustering are some of the applications of neural networks.

A node will assign a number known as a ”weight” to each of its incoming connections. When the network is up and running, each node receives a new data item (a distinct number) and multiplies it by the associated weight. The resultant items are then added together to produce a single number.

If that number falls below a certain threshold, the node does not send any data to the next layer. If the number exceeds the threshold value, the node fires or sends the number (the sum of the weighted inputs) along with all its outgoing connections. All the weights and thresholds of a neural network are initially set to random values when it is being trained. Training data is supplied into the bottom layer (the input layer), where it is multiplied and combined in various ways until it ultimately reaches the output layer, where it is significantly changed. During training, the weights and thresholds are modified until the outputs from training data with the same labels are consistent.

Neural networks can be classified into different types, which are used for different applications [23]. The applications of neural networks include regression analysis and prediction, classification and pattern recognition, filtering, and clustering. The simplest form of a neural network is the perceptron which has a single neuron and was created by Frank Rosenblatt in 1958.

Artificial Neural Network, which can also be known as the multi-layer perceptron (MLP) or the feedforward networks are comprised of multiple layers of artificial neurons. Data is fed to an input layer. They have one or more hidden layers providing levels of abstraction and an output layer. MLPs are the classical type of neural networks, and they are capable of learning nonlinear functions by using activation functions that introduce the nonlinear functionality to the network. The difference between the computed and the actual outputs, or simply the error, is iteratively used to adjust the weights of the network to achieve more accurate outputs.

The Convolutional Neural Network (CNN) is mainly used for image classification, object and pattern recognition, and computer vision. This neural network employs linear algebra principles, specifically matrix multiplication, to identify patterns within images. These tasks can be computationally demanding, requiring graphical processing units (GPU) to train models [24].

A Recurrent Neural Network (RNN) is an artificial neural network that works with time series or sequential data. These deep learning algorithms are widely employed for problems like language translation, natural language processing (NLP), speech recognition, and image captioning. They're used in applications like Siri, voice search, and Google Translate [25]. They are distinguished by their memory that allows them to impact current input and output by using knowledge from previous inputs, which makes their output dependent on the sequence's prior elements.

2.1.4 BigchainDB

BigchainDB is a revolutionary blockchain-based database that enhances security and data integrity in cloud computing [26]. It offers a decentralized and tamper-proof environment for storing and managing data in the cloud [27]. By using cryptographic hashes to link records, BigchainDB ensures the immutability and authenticity of data [28], while also providing real-time auditing and protection against unauthorized access. The integration of BigchainDB with cloud computing addresses challenges such as data security, management, compliance, and reliability. This integration gives users more control over their data and increases trust in software applications hosted in the cloud. BigchainDB is a valuable tool for ensuring data authenticity and integrity in cloud environments, making it a solution for secure and reliable cloud computing.

BigchainDB combines the benefits of distributed databases and traditional blockchains by combining an enterprise-grade distributed database (MongoDB) [29] with a production-ready consensus engine (Tendermint) [30] to provide the benefits of both [31]. BigchainDB has both blockchain properties such as decentralization, immutability, and owner-controlled assets, and database properties such as high transaction rate, low latency, and indexing and querying of structured data.

2.1.5 Cloud Containers Orchestration

The field of container orchestration has been gaining momentum in recent years due to its ability to manage containers running on different computing nodes. However, as the demand for heterogeneous cloud environments like edge computing grows, there is a need for further development in container orchestration [8]. One of the key components of container orchestration is the scheduler, which plays a crucial role in deploying container applications on computing nodes. It determines how computing requests are mapped to containers and how containers are mapped to computing nodes while adhering to a set of objectives and constraints.

The placement of containers is categorized into two distinct approaches: the queueing approach and the concurrent approach. The queueing approach employs a first-in-first-out or priority-based method, with placement decisions made container-by-container. As placement decisions are made independently of other containers, this approach may lead to server imbalance, which may require the relocation of running containers to achieve server balance. The concurrent approach is based on a batch and process concept, where computing requests are collected and container placement decisions are handled as one batch. While this can result in a globally optimal solution, the batching process can add processing delay due to waiting for multiple requests to be collected. This makes it more appropriate for a continuous stream of incoming requests. On the other hand, the container-by-container approach is more suitable for intermittent requests. Kubernetes [32] and Docker Swarm [33] are two widely used orchestration tools in this domain.

Container migration is a process that involves relocating already running containers to other computing nodes for various reasons. One of the most important reasons for migration is to balance the workload within the cluster of computing nodes. This ensures that no node is overburdened with workload while others are underutilized. Apart from load balancing, container migration is also necessary when running services are moved to another computing node due to maintenance or unexpected faults. Overall, container migration plays a critical role in maintaining the optimal performance of the computing infrastructure and the availability of applications by enabling seamless transitions between nodes.

Containers virtualize the operating system and provide execution processes that share the underlying OS kernel [34]. This approach brings container migration more in line with process migration. Checkpoint and Restart (CR) is a popular container migration technology [35], which enables the process memory state to be saved to files (checkpoints) and resumed from the checkpoint at the destination host. In case the base image is not present at the destination node when migrating containers between different nodes, both the application and the base image are migrated.

There are various methods available for container migration, which can be classified into two main types: cold and warm migration. In cold migration, the container service is paused before transferring the container and the base image to another node in a single process, and then it is restarted on the destination node. On the other hand, in warm migration, base images are first replicated on the target node before migrating the container service. The running services are then frozen, saved on checkpointing files, and offloaded to the destination node on top of the previously deployed base images, where they are restarted.

2.1.6 Containers Data Protection in the Cloud

The protection of cloud containers data requires the consideration of multiple factors, including data at rest, data in use, and data in transit [36]. Each of these aspects demands a different approach to ensure that data is protected and secure. When it comes to encrypting cloud containers data, the data at rest is safeguarded by storing it on the data disk, which is secured using authenticated encryption.

In terms of data in use, the chosen hardware solution plays a significant role in protecting it. The hardware encrypts the RAM and registers of the VM that runs the container, thereby providing confidentiality and integrity. This conceptually secures the data while it is being processed. AMD Secure Encrypted Virtualization (SEV) is a hardware-based technology that guarantees the security and privacy of data and services during their deployment [37].

The security of data in transit is a crucial aspect of safeguarding data, which is achieved through the use of secure communication channels. Such channels establish a secure connection between trusted hardware on the source and destination platforms, ensuring that data is protected while in transit. Examples of secure channels include Virtual Private Network (VPN) [38], Secure Shell (SSH), and Secure File Transfer Protocol (SFTP), which can be established directly or via intermediary entities. To prevent brute force attacks, 2048 RSA key pairs are used instead of login and password for all SSH and SFTP connections. Furthermore, SHA-256 is employed to ensure data integrity during transfer [39].

In modern cryptography, Rivest–Shamir–Adleman (RSA), Advanced Encryption Standard (AES), and Elliptic-curve cryptography (ECC) are considered to be the fundamental building blocks, offering specialized security capabilities to address specific requirements. These methods continue to evolve and find widespread use, with ongoing research aimed at developing new and improved cryptographic techniques to meet the challenges of an ever-changing digital landscape.

RSA is a widely adopted asymmetric encryption algorithm that has been a cornerstone of cryptography since its introduction in 1977. It is named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman [40]. The security of RSA is based on the mathematical difficulty of factoring the product of two large prime numbers, which is known as the integer factorization problem. RSA is generally used for key exchange, digital signatures, and securing sensitive data during transmission. The computational infeasibility of deriving the private key from the public key makes RSA a reliable method for achieving secure communication.

The U.S. National Institute of Standards and Technology (NIST) established AES as a symmetric encryption algorithm standard to replace the Data Encryption Standard (DES) [41]. It is widely used to secure sensitive data, such as financial transactions and government communications. AES operates on fixed-size data blocks and supports key lengths of 128, 192, or 256 bits. The design of the AES algorithm is based on the Rijndael cipher created by Vincent Rijmen and Joan Daemen, which was selected through a competitive process announced by NIST in 1997.

Elliptic Curve Cryptography (ECC) is a method of asymmetric encryption that utilizes the mathematical properties of elliptic curves over finite fields. It provides the same level of security as traditional public-key cryptography, but with much smaller key sizes, thereby making it more efficient for resource-limited environments [42]. ECC is widely used in various applications such as secure communications, digital signatures, and key exchange protocols. The efficiency of ECC is attributed to the elliptic curve discrete logarithm problem, which is considered to be computationally challenging.

An encryption scheme that combines both asymmetric and symmetric encryption is commonly known as a hybrid encryption scheme. Typically, the asymmetric encryption technique such as RSA or ECC is used for key exchange and the symmetric encryption, such as AES, is used for data encryption. The hybrid approach provides a secure key distribution through asymmetric encryption and computational efficiency through symmetric encryption. This is a widely used approach in various secure communication protocols and systems, which helps to strike a balance between security and efficiency in modern encryption practices.

Ensuring the security of sensitive data is paramount, and encryption techniques are highly effective when implemented correctly. The strength of the algorithm and the size of the key play a significant role in maintaining security. The larger the key, the higher the level of security [43]. However, it's essential to select the appropriate key length, taking into account specific security requirements, computational resources, and cryptographic approach [44]. Staying abreast of recommended key sizes is crucial, given the rapid advancements in computing technology, to ensure an unbreakable encryption strategy. Encryption algorithms, along with Key Policy Attribute Based Encryption (KP-ABE), has gained significant attention in the field of security. Therefore, matching system and user requirements to the resources' capabilities by optimizing the security key size with the encryption algorithm is an area of active research.

2.2 Literature Review

2.2.1 VM Memory and Disk States Migration

The CloudNet [45] architecture is implemented on Xen hypervisor and uses VPN to provide connectivity among different datacenters connected by WAN. It migrates VM's disk state by using the Distributed Replicated Block Device (DRDB) distributed storage system with the asynchronous replication mode to copy the disk state, it then switches to the synchronous mode when the migrated disk has been brought to a consistent state and then starts the memory state live migration. The memory is migrated using the original Pre-copy migration. A set of optimizations such as the smart stop and copy, the content-based redundancy (CBR) elimination, and the use of delta pages, are also implemented to enhance the migration process.

XvMotion [46] is an integrated memory and disk migration system in the VMware vShpere virtualization suite. It provides end-to-end VM migration over local and WAN connections. It creates multiple Transmission Control Protocol (TCP) connections between hosts with the support of multipathing across NICs, IP addresses, and routes. The multiple connections of Streams transport help in mitigating head-of- the-line blocking issues of TCP, giving better utilization of low-speed connections in WANs. XvMotion uses the Pre-copy approach to migrate memory and CPU states. Disk migration is done in a process called Clone which works by a single pass bulk copying of the disk from the source host to the destination host. Changes on the disk at the source host are reflected on the destination disk by Input Output (IO) mirroring to ensure having identical versions of the disk after the migration finishes.

The system presented in [47] migrates VMs with local storage for their persistent state over a wide area network. This system overcomes the new IP address problem by combining IP tunneling with Dynamic Domain Name System (DNS) which allows to transparently transfer the existing connections and forwarding the packets on the source host to the the new IP address on the destination, while the VM on the source host remains operating during the migration process. The proposed system uses the original Pre-Copy migration algorithm of Xen to migrate the VM's memory.

Authors in [48] use a combination of the pre-copy and the post-copy migration methods in addition to using the disk working set to achieve virtual machines storage migration. The disk working set is the set of disk blocks that may be accessed in the future. At the beginning of the migration, the disk working set is marked. After that, the migration process goes into three stages: the pre-copy stage, the stop-and-copy stage, and the post-copy stage. At the pre-copy stage, a block copy module sequentially mirrors the disk working set blocks to the destination by using IO mirroring to achieve consistency. While copying the working set blocks, the live migration of memory and CPU states is started. In the stop and copy stage, dirty memory pages and working set blocks are transferred. Finally, in the post-copy stage, all remaining disk blocks are transferred. During that, if any read operation to a block that has not been copied yet is intercepted, an on-demand fetching client requests that block from the source host.

Authors in [15] derive a set of mathematical models for the total migration time, migration downtime, and resource consumption. For the Pre-copy to be convergence, the ratio of the memory-dirtying rate D and the provisioned bandwidth R should be less than 1, which means that the migration bandwidth R should be larger than the memory dirtying rate D. Assuming $\lambda = D/R$, a new strategy is proposed called the λ-range mapping that determines appropriate bandwidth and pre-copy iteration count to optimize migration over a WAN with respect to migration duration, downtime, and network resource consumption. It provides a high λ value to migrate a VM with a higher dirtying rate, and a lower λ for the lower dirtying rate. This ensures that a high value of λ will be provisioned if a VM with a high dirty rate is migrated, and a low value of λ will be provisioned if a VM with a low dirty rate is migrated.

A network traffic-sensitive migration method is proposed in [49] to select between pre-copy or post-copy migration techniques when migrating VMs that are located on the same source host. When using the Pre-copy migration technique; VM stays operated on the source host and outbound VM's applications traffic contends with the outgoing migration traffic. And when using Post-copy, VM operates at the destination side, and VM's inbound applications traffic contends with the incoming migration traffic. The migration technique selection is based on the network traffic direction of the source and destination hosts to avoid contention with the migration traffic which affects the migration time and performance.

The literature provides a comprehensive review of various techniques and architectures used in virtualized environments to enable seamless VM migration over both LAN and WAN connections. The collective studies deepen our understanding and optimization of live VM migration processes, which play a crucial role in minimizing downtime and data transfer while maximizing resource efficiency. A comparison of these studies is presented in Table 2.1. However, there is a notable gap in the literature regarding comprehensive mathematical modeling for different migration methods and the primary live VM migration metrics, such as total migration time, downtime, and data transferred. Additionally, existing cloud simulation environments, like CloudSim, do not fully support the simulation of different migration methods for various migration metrics.

Table 2.1: VM memory and disk states migration literature comparison.

Reference	**Hypervisor**	**Migration Approach**	**Storage System**	**Network Optimization**	**Special Features**
Wood et al. [45]	Xen	Pre-copy	DRDB with asynchronous and synchronous replication	VPN for WAN connectivity	Smart stop and copy, CBR elimination, delta pages
Mashtizadeh et al. [46]	VMware vSphere	Pre-copy	Clone (single pass bulk copying)	Multipathing across NICs, IP addresses, and routes	IO mirroring for disk consistency
Bradford et al. [47]	Xen	Pre-copy	Local storage	IP tunneling with Dynamic DNS	Transparent transfer of existing connections
He et al. [48]	Not specified	Pre-copy and post-copy hybrid	Disk working set with IO mirroring	On-demand fetching for uncopied blocks	Three-stage migration process (pre-copy, stop-and-copy, post-copy)

Reference	Hypervisor	Migration Approach	Storage System	Network Optimization	Special Features
Mandal et al. [15]	Not specified	Pre-copy	Not specified	λ-range mapping for bandwidth optimization	Mathematical modeling for total migration time, downtime, resource consumption
Deshpande et al. [49]	Not specified	Pre-copy and post-copy selection	Not specified	Based on network traffic direction to avoid contention	Selection between pre-copy and post-copy based on traffic direction

In this thesis, we present mathematical modeling to assess live VM migration, which evaluates migration time, downtime, and data transfer across three methods: Pre-copy, Post-copy, and Hybrid. Moreover, we propose Dynamic Hybrid-copy, which dynamically adapts migration strategies based on network conditions and VM characteristics. The efficacy of these models is validated by implementing them in CloudSim, which identified avenues for optimization, enhancing live VM migration performance while minimizing delays, downtime, and resource consumption. The findings of this study have significant implications for the design and management of virtualized environments, particularly in optimizing live VM migration processes.

2.2.2 Machine Learning-based VM Migration in Cloud Data Centers

The use of machine learning is crucial in VM migration across cloud data centers and edge nodes. A solution proposed in [50] involves a cloud-edge collaborative technique for defect detection in PV components using transfer learning. Another solution in [51], MiOvnm, utilizes deep reinforcement learning to optimize virtual network migration and improve network performance. A study on model compression mechanisms using Hopfield neural networks to optimize virtual network embedding for edge-cloud collaborative services is conducted in [52]. In [53], machine learning methods like Matrix Completion were employed to design efficient offloading approaches that consider uncertainties and dynamics in user mobility, ultimately minimizing application turnaround time. A learning-driven method based on Partially Observable Markov Decision Process (POMDP) was proposed in [54] to improve service migration in Multi-access Edge Computing (MEC) scenarios, surpassing existing algorithms.

Migration of virtual machines can cause performance degradation and service disruption in data centers. To minimize the number of migrations, a new scheduling algorithm is proposed in [55] that uses prior knowledge and job information to reduce migrations by over 25%. The algorithm is formulated as an integer linear programming problem and a genetic algorithm is used to obtain a heuristic solution. The article in [56] proposes a machine-learning algorithm, DQNVMC, to address the challenges of cloud computing caused by big data, which puts a heavy burden on cloud service providers and the environment. DQNVMC combines the Q-learning approach with a deep learning neural network to find the optimal solution for VM consolidation. The experiments demonstrate that DQNVMC effectively decreases energy consumption while meeting QoS requirements. The authors in [57] propose a VM placement technique that utilizes a DTMC model to predict future resource utilization and classify physical machines based on their status. To balance energy consumption, resource waste, and system dependability, they use the ϵ-dominance-based multi-objective artificial bee colony algorithm. The method is validated with the CloudSim toolkit.

A new double auction-based algorithm for VM migration, considering communication costs, has been proposed in [58]. It selects VMs for migration based on communication and resource factors, and determines the destination host for the VMs. This algorithm is efficient, with a 35% traffic reduction rate compared to a random algorithm, and a high success rate for VM migration according to simulation results. PAVMM, presented in [59], is a method that predicts VM performance after migration to optimize migration by maximizing user performance and minimizing machine usage and costs. It uses an ant colony optimization-based algorithm and has been proven efficient in experiments.

An adaptive prediction-based model is introduced in [60] to dynamically identify the optimal live migration method to migrate a VM with certain performance metrics. Different application workloads and resource constraints were considered in the presented models. For the considered migration methods, the three regression algorithms: Ridge Regression, support vector regression

(SVR), and K-nearest neighbor regression (KNNR) were used to predict six live VM migration metrics. The experimental results demonstrate that the suggested model may reduce the rate of SLA violation by 31% to 60% while also reducing the overall CPU time for the prediction process.

For several migration algorithm approaches and application workloads, a machine learning model is presented in [61] to predict important performance characteristics of live VM migration. The CSAP dataset [62] was used for the training and testing. The six metrics of live VM migration performance: total migration time, downtime, total data transferred, VM performance degradation, host CPU, and memory utilization were predicted using linear regression. Three regression techniques were employed: linear regression, SVR using non-linear kernels, and SVR with bootstrap aggregation. They determined the best policies for migrating virtual machines between hosts while complying with SLA requirements. The results of their experiments revealed a considerable improvement in migration performance.

A machine learning approach is used in [63] to predict the best time to trigger a live VM migration using linear and non-linear prediction. In VMware, they suggested a migration process to reduce the time needed to migrate one or more VMs. The proposed model works on reducing the network contention resulting from live migration traffic by reducing the delay in starting the VM process. This saves about 27% of migration time.

In [64], ANN-based techniques are applied to predict bandwidth and CPU usage during the live migration process to improve the performance of the data center and reduce bandwidth usage. The proposed schemes can accurately predict bandwidth and CPU usage and reduce network traffic and SLA violations which result in reducing the data center energy consumption. In [65], to accurately predict the CPU utilization, training algorithms such as differential evolution, Particle Swarm optimization (PSO), and the covariance matrix adaptation evolutionary strategy (CMA-ES) are used to train the ANN. This resulted in more accurate prediction results.

A two-step local regression model using regression trees is implemented in [66] to predict SLA violation for applications with strict SLA. For migration decisions, support vector machine (SVM) and K-nearest neighbor (KNN) are proposed to be used. The CPU, memory, and bandwidth usage are used as potential classifiers. The results that regression trees have more accuracy than other regression methods used. Similar work was proposed in [67] to classify the VMs using an SVR-based genetic algorithm (GA) and a k-means learning algorithm.

An algorithm presented in [68] enhances resource utilization and reduces energy consumption in cloud environments. The presented algorithm migrates the VMs from overloaded or underloaded servers to normal servers and shuts down the underloaded servers to reduce energy consumption. The algorithm is evaluated on the PlanetLab workload using CloudSim.

The research surrounding VM migration between cloud data centers and edge nodes emphasizes the crucial role of machine learning techniques in streamlining various aspects of the process. The literature proposes solutions that utilize machine learning to identify defects in photovoltaic components, optimize virtual network migration, and predict post-migration VM performance. These solutions minimize performance degradation and service disruption through innovative scheduling algorithms and provide insight into key performance characteristics of live VM migration, ultimately leading to improved migration decision-making and adherence to SLA requirements. Table 2.2 presents a comparison of the solutions proposed in the literature.

However, the literature reveals a significant research gap in selecting an appropriate migration method from various available options based on predicting migration metrics prior to the migration process itself. There is a lack of comprehensive studies that leverage advanced machine learning and deep learning techniques to predict key live VM migration metrics, such as total migration time, downtime, and data transfer volume. Utilizing these predictive techniques could significantly enhance the scheduling and performance of VM migrations, leading to more efficient resource utilization and minimized disruption during migration. Furthermore, incorporating such predictive models into cloud simulation environments, like CloudSim, would provide a robust framework for analyzing and optimizing different migration strategies.

Table 2.2: Machine learning-based VM migration in cloud data centers literature comparison.

Reference	**Focus**	**Machine Learning Technique**	**Migration Optimization**	**Performance Metrics**	**Special Features**
Wang et al. [50]	Defect detection in PV components	Transfer learning	Cloud-edge collaboration	Accuracy of defect detection	Improved defect detection through transfer learning
Wang et al. [51]	Virtual network migration	Deep reinforcement learning (MiOvnm)	Network performance optimization	Network performance metrics	Enhanced network performance
Yang et al. [52]	Virtual network embedding	Hopfield neural networks	Model compression	Embedding efficiency	Optimized network embedding
Maleki et al. [53]	Efficient offloading	Matrix completion	User mobility consideration	Application turnaround time	Handles user mobility dynamics
Wang et al. [54]	Service migration in MEC	POMDP	Service migration improvement	Service quality	Outperforms existing algorithms
Padhy et al. [55]	Reduce VM migrations	Integer linear programming, genetic algorithm	Scheduling optimization	Migration count reduction	Over 25% reduction in migrations
Tong et al. [56]	VM consolidation	DQNVMC (Q-learning + deep learning)	Energy consumption optimization	QoS, energy consumption	Decreases energy consumption, meets QoS
Sayadnavard et al. [57]	VM placement	DTMC, ϵ-dominance multi-objective ABC	Resource utilization balance	Resource usage, system dependability	Uses CloudSim toolkit
Zhu et al. [58]	VM migration with communication cost	Double auction-based algorithm	Migration efficiency	Traffic reduction	35% traffic reduction, high success rate
Zhao et al. [59]	Optimize migration	Ant colony optimization	User performance maximization	Performance, cost	Efficient optimization method

Reference	**Focus**	**Machine Learning Technique**	**Migration Optimization**	**Performance Metrics**	**Special Features**
Motaki et al. [60]	Live migration method selection	Ridge regression, SVR, KNNR	Dynamic identification of optimal method	SLA violation rate, CPU time	31-60% reduction in SLA violations
Jo et al. [61]	Performance prediction	Linear regression, SVR with bootstrap	Performance characteristics prediction	Migration time, downtime, data transfer, CPU/memory utilization	Uses CSAP dataset
Elsaid et al. [63]	Best time to trigger migration	Linear and non-linear prediction	Network contention reduction	Migration time	27% reduction in migration time
Duggan et al. [64]	Bandwidth and CPU usage prediction	ANN-based techniques	Data center performance improvement	Bandwidth, CPU usage, network traffic	Reduces network traffic, SLA violations
Mason et al. [65]	CPU utilization prediction	Differential evolution, PSO, CMA-ES	Accurate prediction	CPU utilization	More accurate prediction results
Hassan et al. [66]	SLA violation prediction	Regression trees, SVM, KNN	Migration decision support	SLA violation, CPU, memory, bandwidth usage	Regression trees more accurate
Sui et al. [67]	VM classification	SVR-GA, k-means learning	Classification accuracy	VM classification accuracy	Uses SVR-GA and k-means
Garg et al. [68]	Resource utilization	Not specified	Energy consumption reduction	Resource utilization, energy consumption	Evaluated on PlanetLab workload

Our research builds upon this foundation by leveraging machine and deep learning algorithms to accurately predict VM migration performance metrics. Additionally, we introduce a VM migration method selector that takes into account predicted performance metrics and predefined SLA policies, providing a comprehensive approach to optimizing VM migration processes.

2.2.3 Blockchain-based VM Scheduling and Migration in Cloud and Edge Data Centers

Integrating Blockchain technology with cloud and edge clusters has the potential to enhance cloud transaction security and provide access to data and applications. Authors in [69] present a solution that utilizes the Vickrey-Clarke-Groves (VCG) auction theory to optimize the Edge Demand Response (EDR) process, leading to better load balancing of edge nodes and improved utilization of edge resources, resulting in reduced system energy consumption. To enhance the incentive and trust mechanism of edge collaboration, the authors integrate blockchain technology, exploring three scenarios: no collaboration, internal collaboration, and incentive collaboration. The impact of user task's transmission distance on the QoE is carefully considered, and the possible forking attack of blockchain in collaborative edge computing is addressed. By incorporating blockchain, the solution ensures the continuous and tamper-proof records of task execution, establishes a trust mechanism among edge nodes, and incentivizes servers to actively participate in edge collaboration while maintaining task execution quality.

Blockchain has emerged as a promising technology for integrating with cloud clusters, improving the security of cloud transactions, and providing access to data and applications. The approach proposed in [70] utilizes blockchain to place VMs for decentralized cloud and edge computing environments. A dynamic threshold-based system is employed, where each server considers its own and other servers' present and future CPU utilization before deciding on VM migration. Experimental results show that this approach outperforms random peer-to-peer VM placement. Blockchain enables secure communication among servers and ensures the identified server maintains its current and future utilization below the upper threshold usage limit. The article in [71] presents a new cloud scheduler based on blockchain technology, which aims to improve security in task allocation, data storage, and transmission among cloud nodes and clusters. The scheduling optimization problem is solved using an asymmetric Stackelberg game model with schedule execution time as the privileged criterion. The proposed blockchain cloud scheduler is validated by implementing the model as a specially designed simulator called the Blockchain Secure Cloud Scheduler Simulator - BCSchedCloudSim, and its effectiveness is confirmed through experimental analysis.

The study in [72] compares the performance of BigchainDB and Amazon Quantum Ledger Database (QLDB). It is crucial to consider the specific use cases for each technology for the comparison. QLDB, as a proprietary technology designed by Amazon, is best suited for situations where a trusted authority is acknowledged by all participants and centralization is not an issue. On the other hand, BigchainDB provides real decentralization across distributed ledgers, making it optimal for cases where there is no trusted authority and centralization is a concern. Additionally, BigchainDB benefits from the advantages of modern distributed databases and the blockchain, including high transaction rates, low latency, indexing, and structured data querying, as well as decentralized control, tamper-proofing, and traceability [31]. While Amazon QLDB may have better overall performance based on CPU and memory usage metrics, BigchainDB's flexibility and potential for wider application makes it a valuable technology to consider. These findings highlight the importance of carefully evaluating the specific needs and requirements of a given project when choosing between decentralized and centralized ledger databases.

A literature review comparison is presented in Table 2.3. Recent studies have shown that there is a gap in the existing literature when it comes to the development of a secure and efficient distributed system that can handle a large volume of transactions for scheduling cloud resources across multiple data centers using blockchain technology. Our research aims to bridge this gap by utilizing the BigchainDB platform, which combines the best of both worlds by offering a distributed, immutable, and secure blockchain along with the scalability and low latency of distributed databases. We are particularly interested in exploring the migration performance of the cloud scheduler, focusing on reducing the impact of VM migration across data centers on the overall scheduling delays of the system. Our findings have significant implications for the future development of cloud resource scheduling systems and can contribute to the ongoing effort to make cloud environments more efficient and secure.

Table 2.3: Blockchain-based VM scheduling and migration in cloud and edge data centers literature comparison.

Reference	**Focus**	**Blockchain Integration**	**Optimization Techniques**	**Performance Metrics**	**Special Features**
Gao et al. [69]	Optimize EDR process, load balancing	Blockchain for incentive and trust mechanism	VCG auction theory, load balancing	System energy consumption, QoE	Addresses forking attacks, tamper-proof task records
Rathod et al. [70]	Decentralized VM placement	Blockchain for secure communication	Dynamic threshold-based system	CPU utilization	Outperforms random peer-to-peer VM placement
Wilczynski et al. [71]	Improve security in task allocation	Blockchain for cloud scheduler	Asymmetric Stackelberg game model	Schedule execution time	BCSched CloudSim simulator validation
Lupaiescu et al. [72]	Compare BigchainDB and QLDB	Blockchain database comparison	Use case analysis	Transaction rates, latency, indexing	Highlights centralization vs. decentralization trade-offs
BigchainDB [31]	Distributed ledger benefits	Real decentralization	Modern distributed databases	High transaction rates, low latency	Tamper-proofing, traceability

2.2.4 Cloud Containers Scheduling Optimization

Containerization technology has experienced widespread adoption in cloud environments. The deployment of microservices-based applications has also become more efficient and speedy with the aid of Evolutionary algorithms (EAs). However, current EAs have been unable to tackle the problem of placing containers at a large scale due to their poor scalability and high time complexity. A multi-factorial evolutionary algorithm (MFEA) presented in [73] can handle the container placement problem in heterogeneous cluster environments. MFEA is a multi-objective container placement algorithm that integrates a system model of heterogeneous clusters, microservices, containers, and four optimization objectives. With a local search strategy, the proposed algorithm can handle multiple optimization problems simultaneously, significantly reducing optimization time, and providing competitive solutions that consider four conflicting objective functions. DQoES, a differentiated quality of experience-based scheduler for containerized deep learning applications, is proposed in [74]. It allows users to specify objectives for the cloud system and dynamically adjusts resource allocation at runtime to meet individual QoE objectives, resulting in up to 8x more satisfied models compared to existing systems.

Deep reinforcement learning (DRL) is used in [75, 76, 77]. In [75], authors propose a DRL deployment algorithm for Long-Running Application (LRA) class containers. The algorithm optimizes placement and scheduling objectives, resulting in a 56.2% improvement in performance. This approach improves deployment efficiency and requests per second (RPS) post-deployment. In [76], an innovative scheduling algorithm is introduced to optimize the utilization efficiency of cloud resources and enhance the quality of service. This priority-aware containerized heterogeneous workloads scheduling algorithm is based on DRL and incorporates two priority mechanisms to enhance scheduling accuracy and learning efficiency. This algorithm outperforms other methods in terms of resource utilization and scheduling performance. A Kubernetes-focused scheduler presented in [77] utilizes a multi-agent deep reinforcement learning-based scheduling model that includes a dispatching agent at each edge access point to receive and redirect requests.

An optimized meta-heuristic container placement solution named Discrete Firefly with Local Search Mechanism (DFFLSM) and Discrete Firefly Algorithm (DFF), designed specifically for cloud data centers (DC) is introduced in [78]. The article compares these algorithms with existing options such as first fit (FF), the First Fit Decreasing (FFD), random, and Ant Colony Optimization (ACO) algorithms. The results indicate that DFFLSM and DFF outperformed other algorithms in terms of energy efficiency, active VMs, active PMs, and average overall SLA violations. The article also delves into the impact of various container placement algorithms for different overbooking factors, highlighting the fact that higher overbooking factors lead to increased energy consumption. In [79], an assessment of energy consumption in data centers is conducted using two different container placement strategies. The evaluation of the FFD of Bin Packing and ACO for container placement is done using CloudSim before and after implementing the placement. Notably, the FFD method displays less energy consumption compared to the ACO method.

Load balancing and affinity requirements are the main focus in [80, 81, 82]. Long et al. in [80] introduce a novel solution to the container instance allocation issue in heterogeneous server clusters: the global cost-aware scheduling algorithm (GCCS). By leveraging Integer linear programming (ILP) to represent the number of containers per server and using a Bayesian optimizer to select the cost coefficient, GCCS achieves a reduction in overall power consumption while ensuring that the affinity/anti-affinity requirements of applications are met. This algorithm successfully balances power efficiency and affinity satisfaction, resulting in a significant decrease in the total power consumption of the cluster. Chen et al. in [81] propose a deployment model for containers that prioritizes fault tolerance, resource utilization, and deployment cost optimization in edge computing systems. They develop a strategy based on the genetic-simulated annealing algorithm (IGSAA), which enhances the initialization, crossover, and mutation operations of the genetic algorithm to achieve optimal container deployment even in the event of hardware or deployment failures. The article in [82] introduces a two-stage framework for managing containers. The framework effectively balances loads over a long-term period and minimizes migration costs while maintaining load balancing on a short-term scale. The solution is also strengthened by its ability to predict workload.

The authors in [83] present a secure cloud architecture that prioritizes the safeguarding of data, logic, and computation at every stage, in transit (using RSA), at rest (using AES-XTS), and in use (using AMD SEV technology), through the use of three key encryption technologies: encrypted containers, encrypted VMs, and encrypted storage systems, along with resource isolation. This architecture offers varying levels of security depending on the sensitivity of the data, ensuring that user workflows are always protected. A confidential computing concept is introduced in [36] to securely migrate operating system containers. The approach ensures that data inside the containers is kept confidential and integrity-protected throughout the migration process. The destination platform is authenticated and verified through remote attestation before the data transfer. Hardware-based security protection is proposed for migrated data using traditional migration strategies.

The article in [84] introduces CSSM, a Cloud Secure Storage Mechanism, which addresses the issue of cloud data leakage caused by management negligence and malicious attacks at the storage layer. CSSM combines data dispersion and encryption technologies to improve data security and prevent attackers from stealing user data. CSSM adopts a hierarchical management approach to protect key security further and combines user passwords with secret sharing. CSSM can effectively prevent user data leakage at the cloud storage layer, while the increased time overhead is acceptable to users which effectively protects cryptographic materials from a storage perspective. Authors in [85] presents transparent data encryption solution for data at rest, designed for cloud storage. The architecture is based on a STRIDE threat model, with key management separated from data storage. The architecture is based on a controller plugin and a node plugin, and the encryption keys are stored using kubernetes-secrets. The performance is sufficient for most applications, with potential improvements to be made in the future.

The co-resident threat problem is tackled in [86, 87]. A Secure Container Placement Strategy (SecCPS) that uses deep reinforcement learning to mitigate co-resident threats between containers running on virtual machines in the cloud is proposed in [86]. SecCPS is implemented using a policy gradient algorithm, and its effectiveness and scalability are verified through extensive experiments. SecCPS can effectively reduce co-residence compared to existing strategies and is scalable and flexible for large-scale container deployment scenarios. SecCDS, a Secure Container Deployment Strategy is presented in [87]. The strategy utilizes the Genetic Algorithm (GA) to detect and migrate containers in real time and separates tenants with a high-risk profile. The SecCDS strategy has proven to be highly effective in preventing co-residency and is scalable to meet the security requirements of large data centers.

A comparison of the literature review is presented in Table 2.4. Previous works in [73, 74, 75, 76, 77, 78, 79, 80, 81, 82] have predominantly focused on prioritizing users' Quality of Experience (QoE) and minimizing Service Level Objective (SLO) violations over considerations of container security, cost, and resource utilization. Notably, while some works have addressed container migration, such as [83, 36], they overlook the security costs inherent in the scheduling process and do not advocate for recent strong encryption schemes for data protection. Similarly, solutions like those in [84, 85] primarily tackle data security at rest, neglecting potential risks during container migration when data is in transit, and fail to explore the encryption cost implications during scheduling. Additionally, works such as [86, 87] concentrate on securing data at rest and in use but overlook security considerations for migrated containers in placement strategies. In contrast, our proposed solution offers a comprehensive scheduling approach that encompasses these critical factors.

Table 2.4: Cloud containers scheduling optimization literature comparison.

Reference	**Focus**	**Optimization Techniques**	**Performance Metrics**	**Special Features**
Liu et al. [73]	Multi-objective container placement in heterogeneous clusters	Multi-Factorial Evolutionary Algorithm (MFEA), local search strategy	Optimization time, competitive solutions for four conflicting objectives	Integrates system model of heterogeneous clusters, microservices, and containers
Mao et al. [74]	Dynamic resource allocation for containerized deep learning applications	Differentiated quality of experience-based scheduler (DQoES)	User satisfaction, QoE objectives	Allows users to specify cloud system objectives
Deng et al. [75]	Deployment of LRA class containers	Deep Reinforcement Learning (DRL)	Deployment efficiency, Request Per Second (RPS) post-deployment, performance improvement	56.2% improvement in performance
Zhu et al. [76]	Optimize cloud resource utilization and service quality	Priority-aware containerized heterogeneous workloads scheduling, DRL	Resource utilization, scheduling performance	Two priority mechanisms for enhanced accuracy and efficiency
Han et al. [77]	Kubernetes-focused scheduling model	Multi-agent deep reinforcement learning-based scheduling	Scheduling efficiency	Dispatching agent at each edge access point
Katal et al. [78]	Optimized container placement in cloud data centers	Discrete Firefly with Local Search Mechanism (DFFLSM), Discrete Firefly Algorithm (DFF)	Energy efficiency, active VMs, active PMs, SLA violations	Comparison with FF, FFD, random, and ACO algorithms
Bouaouda et al. [79]	Assessment of energy consumption in data centers	FFD of Bin Packing, ACO for container placement	Energy consumption	Uses CloudSim for evaluation

Reference	Focus	Optimization Techniques	Performance Metrics	Special Features
Long et al. [80]	Container instance allocation in heterogeneous clusters	Global cost-aware scheduling algorithm (GCCS), ILP, Bayesian optimizer	Power consumption, affinity satisfaction	Balances power efficiency and affinity/anti-affinity requirements
Chen et al. [81]	Fault tolerance, resource utilization, and deployment cost optimization	Genetic-Simulated Annealing Algorithm (IGSAA)	Container deployment efficiency	Enhances genetic algorithm operations for optimal deployment
Zhang et al. [82]	Long-term load balancing and migration cost minimization	Two-stage framework	Load balancing, migration costs	Predicts workload for better management
Zhou et al. [83]	Secure cloud architecture for data, logic, and computation protection	RSA, AES-XTS, AMD SEV technology	Data security, resource isolation	Encrypted containers, VMs, and storage systems
Pecholt et al. [36]	Secure migration of OS containers	Confidential computing concept, remote attestation	Data confidentiality, integrity protection	Hardware-based security during migration
Song et al. [84]	Secure storage mechanism for cloud data	CSSM, data dispersion, encryption	Data security, prevention of data leakage	Hierarchical management, secret sharing
Gabriel et al. [85]	Transparent data encryption for cloud storage	STRIDE threat model, Kubernetes-secrets	Data at rest security	Separate key management from data storage
Deng et al. [86]	Mitigate co-resident threats in cloud containers	Secure Container Placement Strategy (SecCPS), deep reinforcement learning	Co-residence reduction, scalability	Policy gradient algorithm for extensive scalability

Reference	Focus	Optimization Techniques	Performance Metrics	Special Features
Kong et al. [87]	Prevent co-residency in container deployment	Secure Container Deployment Strategy (SecCDS), Genetic Algorithm (GA)	Real-time container detection and migration, tenant separation	Effective for large data centers

The presented literature review provides a comprehensive analysis of techniques and architectures for seamless VM migration over both LAN and WAN connections. Emphasis is placed on minimizing total migration time, downtime and data transfer. Additionally, it evaluates recent solutions for secure placement and migration of containers across multi-data center environments. Notably, there is a gap in the comprehensive mathematical modeling for different migration methods and key metrics, as well as in current cloud simulation environments such as CloudSim, which lack support for simulating various migration methods. Furthermore, there is a need for the utilization of advanced and accurate machine learning and deep learning algorithms in VM scheduling and migration management, coupled with secure and efficient communication methods for VM migration information sharing. The current literature also overlooks important aspects and SLA policies in container scheduling solutions, lacking a comprehensive, secure, and efficient approach.

The gaps identified are addressed by presenting mathematical models for assessing live VM migration across various methods and introducing a Dynamic Hybrid-copy strategy, validated within CloudSim. Moreover, machine and deep learning algorithms are employed to predict VM migration performance metrics, and a VM migration method selector based on these predictions is implemented. Furthermore, the exploration of blockchain technology for secure and efficient cloud resource scheduling aims to reduce migration impact on scheduling delays. By prioritizing comprehensive scheduling approaches that consider security, cost, and resource utilization alongside traditional performance metrics, this work builds upon previous research. Finally, this thesis addresses the pressing issues surrounding the scheduling of virtualized cloud resources by presenting a comprehensive and secure optimization solution that enhances the placement of containers on computing nodes, achieving superior server consolidation and balance through container migration.

Chapter 3

Modeling and Analysis of VM Migration Methods in Cloud Environments

The process of transferring VM's data should be done with the least possible delay and service downtime, and the minimum amount of data being transferred. In this chapter, we present the mathematical models to calculate the total migration time, the downtime, and the total data transferred for the three known migration methods: Pre-copy, Post-copy, and a Hybrid of both Pre- copy and Post-copy. After that, we present the mathematical models of an enhanced hybrid method for live VM migration the Dynamic Hybrid-copy. We implemented and simulated the mathematical models on CloudSim, and we propose further optimizations for better performance.

3.1 Classical VM Migration Methods

In this section, we analyze and compare the three migration methods: the Pre-copy, the Post-copy, and the Hybrid-copy method. We show the mathematical model for the Pre-copy method as proposed in [15]. We present our mathematical models for Post-copy and Hybrid-copy methods for the migration downtime, total migration time, and total transferred data during the migration process. We implement the three models using MATLAB and evaluate the performance of these migration methods for different experimental parameters [16].

3.1.1 Classical VM Migration Methods Mathematical Modeling

A. Pre-copy

The pre-copy method can be used to migrate VMs through fast network connections, especially in LAN connected servers, where the bandwidth could be high enough to transfer large amounts of data in a short time, but adopting this method alone without further improvements when migrating through lower bandwidth networks, like WAN, will be inefficient and results in high service downtime and network burden. When migrating the memory using the pre-copy method, the memory migration time will depend on many factors like the virtual memory size, the allocated bandwidth, and the memory dirtying rate. Following is the model describing the Pre- copy approach as presented in [15] and we rederive the equations for the Total Migration Time, Downtime, and the total transferred data over the network.

Figure 3.1 shows the pre-copy iterations for a VM's memory. The size of the memory being migrated in the first iteration equals the full size of the memory (V_{mem}). And then in every other iteration, the size of memory being migrated is the size of memory dirtied while migrating in the previous iteration, which is given by:

$V_i = D_{mem} \times T_{i-1}$,

where V_i is the size of VM dirtied memory pages in iteration i, D_{mem} is the memory dirtying rate, and T_{i-1} is the delay of the previous iteration to iteration i.

For the convergence of the Pre-copy, the Bandwidth (R) must be larger than the memory dirtying rate (D_{mem}), or in other words, the ratio of the memory dirtying rate to the Bandwidth ($\frac{D_{mem}}{R}$) should be less than 1 for the Pre-copy to work properly.

The duration of the first iteration:

$T_1 = \frac{V_{mem}}{R} + \tau$,

where V_{mem} is the VM memory size, R is the bandwidth allocated for migration, and τ is the inter-iteration delay between copy iterations. In later iterations, the delay of iteration i is given by:

$T_i = \frac{V_i}{R} + \tau = \frac{D_{mem} \times T_{i-1}}{R} + \tau$.

Let $\lambda = \frac{D_{mem}}{R}$, then the delay of iteration i is given by:

$T_i = \lambda T_{i-1} + \tau$,

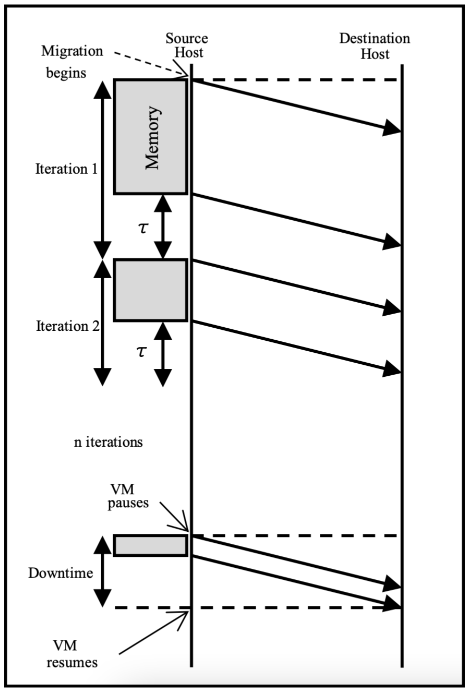


Figure 3.1: Pre-copy iterations for a VM's memory migration.

and the delay of the second iteration is given by:

$T_2 = \lambda T_1 + \tau = \frac{V_{mem}}{R}\lambda + \lambda\tau + \tau = \frac{V_{mem}}{R}\lambda + (1+\lambda)\tau.$

Based on the definition of the finite geometric series: $1 + x + x^2 + x^3 + ... + x^{\mathrm{n}} = \frac{1-x^{\mathrm{n+1}}}{1-x}$,

$T_2 = \frac{V_{mem}}{R}\lambda + \left(\frac{1-\lambda^2}{1-\lambda}\right)\tau,$

$T_3 = \lambda T_2 + \tau = \frac{V_{mem}}{R}\lambda^2 + (1+\lambda+\lambda^2)\tau = \frac{V_{mem}}{R}\lambda^2 + \left(\frac{1-\lambda^3}{1-\lambda}\right)\tau.$

T_1 can now be rewritten as:

$T_1 = \frac{V_{mem}}{R} + \tau = \frac{V_{mem}}{R}\lambda^0 + \left(\frac{1-\lambda^1}{1-\lambda}\right)\tau.$

Therefore, in general:

$T_i = \frac{V_{mem}}{R}\lambda^{\mathrm{i-1}} + \left(\frac{1-\lambda^{\mathrm{i}}}{1-\lambda}\right)\tau.$

Total migration time (T_{mig}), given by Equation 1, is the time needed for all iterations to transfer memory pages, plus the downtime and the network reconfiguration time.

$$\begin{aligned} T_{mig} &= \sum_{i=1}^{n} T_i + T_{Down} + T_{net} \\ &= \frac{V_{mem}}{R}\sum_{i=1}^{n}\lambda^{\text{i-1}} + \frac{\tau}{1-\lambda}\sum_{i=1}^{n}(1-\lambda^{\text{i}}) + T_{Down} + T_{net} \\ &= \frac{V_{mem}}{R}\frac{1-\lambda^{\text{n}}}{1-\lambda} + \tau\frac{n(1-\lambda)-\lambda(1-\lambda^{\text{n+1}})}{(1-\lambda)^2} + T_{Down} + T_{net}, \end{aligned} \tag{1}$$

where T_{Down} is the downtime, and T_{net} is the network reconfiguration time which is the delay needed for reconfiguring and redirecting the network traffic to the new VM's location.

The remaining memory pages after the last copy iteration (after the n$^{\text{th}}$ iteration) phase can be given by:

$V_s = T_n \times D_{mem} = V_{mem} \times \lambda^{\text{n}} + \tau \times D_{mem} \times \frac{1-\lambda^{\text{n}}}{1-\lambda}$,

where V_s is the amount of memory in the stop and copy phase (after the n$^{\text{th}}$ iteration).

The downtime (T_{Down}) is the duration of the stop and copy phase, which the delay to copy the remaining memory pages after the n$^{\text{th}}$ iteration which is given by Equation 2.

$$T_{Down} = \frac{V_{mem}}{R}\lambda^{\text{n}} + \lambda\tau\frac{1-\lambda^{\text{n}}}{1-\lambda}. \tag{2}$$

As mentioned earlier, $V_i = D_{mem} \times T_{i-1}$, then the total transferred memory in each iteration (M_i) will be given by:

$$M_i = \begin{cases} V_{mem} & \text{, i = 1,} \\ D_{mem} \times T_{i-1} & \text{, i = 2, 3, ..., n.} \end{cases}$$

The total amount of data to be migrated (M) will be given by Equation 3

$$\begin{aligned} M &= V_{mem} + \sum_{i=2}^{n}(d_{mem} \times T_{i-1}) \\ &= V_{mem} + d_{mem}\sum_{i=2}^{n} T_{i-1} \\ &= V_{mem} + V_{mem}\lambda\frac{1-\lambda^{\text{n}}}{1-\lambda} + \tau D_{mem}\frac{n(1-\lambda)-\lambda(1-\lambda^{\text{n+1}})}{(1-\lambda)^2}. \end{aligned} \tag{3}$$

B. Post-copy

When migrating the memory using the Post-copy method, the memory migration time will depend on many factors like the virtual memory size, the memory dirtying rate which increases the page fault rate that happens when trying to access a page at the destination host that has not been copied yet, the bandwidth, and the fetching techniques used to fetch the memory pages while the VM runs on the destination host.

The downtime for the post-copy is basically the time of network configuration and the time to copy the CPU status (V_c), which is part of the VM memory size, to the destination host. The downtime is given by Equation 4.

$$T_{down} = \frac{V_c}{R} + T_{net}. \tag{4}$$

The page fault time (T_f) is the time to process and copy a single memory page and is given by:
$T_f = \frac{P_z}{R} + T_{f\_p}$,
where P_z is the memory page size and T_{f_p} is the processing time of a page fault to catch the page fault and request the page from the source.

The technique used to fetch memory pages gives different variants of post-copy. For example, memory pages can be transferred from the beginning to the end of memory with every page being sent only once. Another technique is to request the memory pages from the source host when they are needed. In this research, we will describe three memory fetching techniques used for post-copy: The Demand Paging, the Active Pushing, and the Pre-Paging [21].

1) Demand-Paging:

This technique works by processing every memory access for a memory page that has not been fetched yet at the destination host as a page fault, and then this page fault is serviced by requesting it over the network from the source host. The network latency slows down the VM and decreases the performance of the VM being migrated. Also, this technique will increase the residual dependencies on the source host for an unknown duration. Although this technique copies each memory page only once to have less data being migrated, it is unacceptable to be used because of its lower performance and high dependability on the source host.

The total migration time for the Demand-Paging technique is given by Equation 5.

$$T_{mig} = P_f \times T_f \times \frac{V_{mem} - V_c}{R} + T_{down}, \tag{5}$$

where P_f is the probability of page faults. And the total migrated data is given by Equation 6.

$$M = P_f(V_{mem} - V_c) + V_c. \tag{6}$$

2) Active Pushing:

This technique works by pushing the memory pages to the destination host where the VM runs there and serving any page fault in parallel via demand paging. This will decrease the duration of source host dependency by pushing memory pages in advance to the memory access, and at the end, each page will be copied only one time, either by active pushing or demand paging. The migration time for this technique is given by Equation 7.

$$T_{mig} = T_{mig}(No\_page\_faults) + T_{mig}(Page\_faults) + T_{down}, \tag{7}$$

where:

$T_{mig}(No\_page\_faults) = (1 - P_f)\frac{V_{mem}-V_c}{R}$,

and

$T_{mig}(Page\_faults) = P_f \times Total\_pages \times T_f = P_f \times T_f \times \frac{V_{mem}-V_c}{P_z}$.

The total migrated data is actually the whole VM's memory and is given by Equation 8.

$$M = V_{mem}. \tag{8}$$

3) Pre-Paging:

This technique estimates the spatial locality of the memory access fault to predict a faulting pattern and push memory pages following that pattern. A window of the pages to be transmitted is used, with the page that caused the fault is located within this window. With spatial locality concept, this technique will reduce next page faults by copying in advance the pages that may cause faults in the near future.

The performance of the pre-paging algorithm is measured by the percentage of the page faults over the network; the fewer page requests from the source, the better performance. The key is in predicting the pages that might be accessed in the near future and push them to the destination host in advance to decrease the demand paging over the network. The total migration time is given by:

$$T_{mig} = T_{mig}(No\_page\_faults) + T_{mig}(Page\_faults) + T_{down}, \tag{9}$$

where:

$T_{mig}(No\_page\_faults) = \frac{(1-P_f)(V_{mem}-V_c}{R} - \frac{P_f(w-1)(V_{mem}-V_c)}{R} = (1 - P_f \times w)\frac{V_{mem}-V_c}{R}$,

and

$T_{mig}(Page\_faults) = P_f \times Total\_pages \times T_f + \frac{P_f \times P_z \times (w-1)}{R} = P_f \times (T_f + \frac{P_z(w-1)}{R}) \times \frac{V_{mem}-V_c}{P_z}$,

where w is the window size or the number of memory pages that will be copied when a page fault is processed.

The total migrated data is the whole VM's memory which is given by Equation 10.

$$M = V_{mem}. \tag{10}$$

C. Hybrid-copy

This technique works by running a single pre-copy round to copy the VM's memory while the VM continues running on the source host. After that single round, the VM is stopped and the CPU state is copied to the destination host and resumes running there. After that, the post-copy starts working to copy the remaining dirty pages from the source using the pre-paging memory fetching technique. This technique works better for read-intensive workloads as they would have less dirty memory pages and hence give better performance. The total migration time is given by:

$$T_{mig} = T_{mig}(pre\_copy\_iteration) + T_{mig}(Pre\_paging\_post\_copy) + T_{down}, \tag{11}$$

where:

$T_{mig}(pre\_copy\_iteration) = \frac{V_{mem}}{R}$,

and

$T_{mig}(Pre\_paging\_post\_copy) = (1 - wP_f)\left(\frac{V'_{mem}-V_c}{R}\right) + P_f\left(T_f + \frac{P_z(w-1)}{R}\right)\left(\frac{V'_{mem}-V_c}{P_z}\right)$,
where V'_{mem} is the rest of the memory after doing the pre-copy iteration and is given by:
$V'_{mem} = D_{mem}\frac{V_{mem}}{R}$.

The total migrated memory data is the total of memory pages transferred in the pre-copy iteration and later post-copy, and it is given by Equation 12.

$$M = V_{mem}(1 + \frac{D_{mem}}{R}). \tag{12}$$

3.1.2 Results and Discussion

MATLAB was used to implement the above equations of downtime, total migration time, and total transferred data for the five migration methods: Pre-copy, Post-copy with Demand- Paging, Post-copy with Active-Pushing, Post-copy with Pre-Paging, and the Hybrid-copy. Figures 3.2 and 3.3 show the relation between the number of iterations of the Pre-copy migration method and the total migration time, total transferred data, and downtime. These results were conducted for different dirtying rates (T_{mem}) for the link speed of 1 Gbps and a VM memory size of 4 GB with an inter-iteration delay (τ) of 700 ms and a network configuration delay (T_{net}) of 700 ms.

In Figure 3.2 (a), the total migration time for migrating this VM is higher when the number of iterations is higher. This is a result of repeating the memory copying process many times. The total migration time decreases by lowering the number of iterations. The total transferred data in Figure 3.2 (b) has the same behavior as the total migration time. Having more iterations will result in repeatedly copying the memory pages and increasing the total amount of transferred data between the source and destinations hosts over the connecting link.

The downtime in Figure 3.3 decreases with a higher number of iterations. Downtime in the Pre-copy migration comes from copying the VM's CPU state and the remaining memory pages after the last iteration while the VM is stopped. Having more iterations results in a lower number of memory pages after the last iteration, which decreases the downtime. For a smaller number of iterations, the remaining memory pages will be larger and result in a larger time to copy them.

The total transferred data in Figure 3.3 has the same behavior of the total migration time. Having more iterations will result in repeatedly copying the memory pages, and increasing the total amount of transferred data between the source and destinations hosts over the connecting link.

The number of iterations for the Pre-copy method should be chosen wisely in order to have minimal total migration time, downtime, and total transferred data. Figures 3.2 and 3.3 also show that the Pre-copy migration method performs better for lower Memory Dirtying rates. The total migration time in Figure 3.2 (a) increases when the dirty rate increases for the same VM and link speed. Downtime in Figure 3.3 is longer for larger Memory Dirtying rates. The total transferred data is larger for larger Memory Dirtying rates. This happens because larger Memory Dirtying rates will

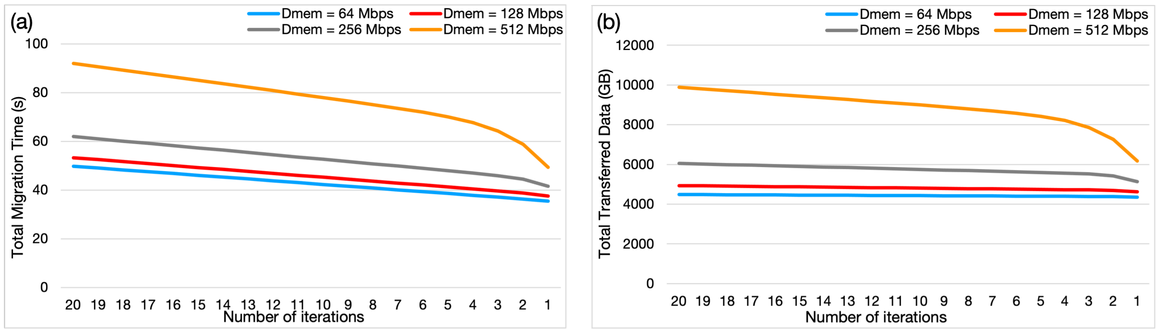


Figure 3.2: Pre-copy total migration time and total data transferred with the number of iterations for different dirtying memory rates for the link speed of 1 Gbps: (a) Total migration time, (b) Total transferred data.

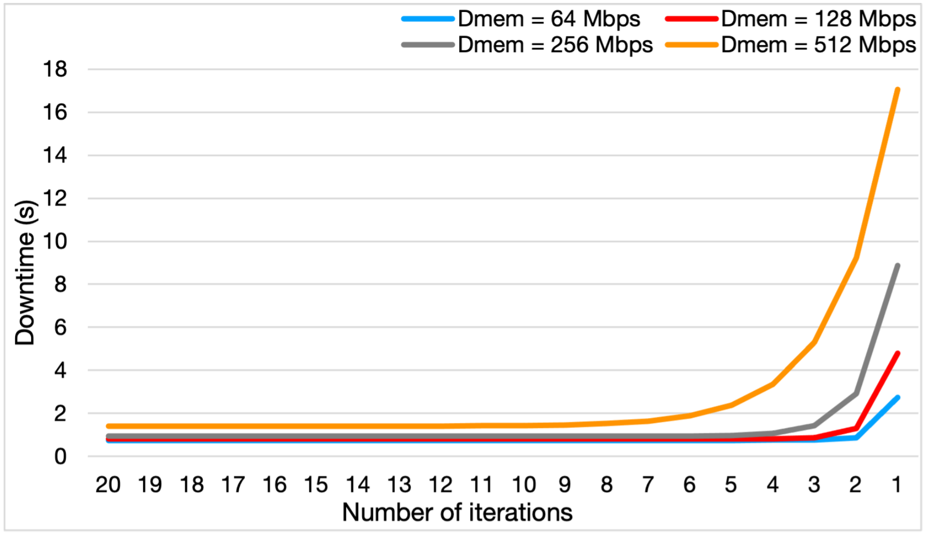


Figure 3.3: Pre-copy downtime with the number of iterations for different dirtying memory rates for the link speed of 1 Gbps.

result in more memory pages getting dirty during the migration process and increase the amount of memory to be copied at each iteration. It also will make the amount of memory after the last iteration larger, causing to have a longer downtime.

Figure 3.4 (a) shows the relation of total migration time and the Page Fault Probability for the Post-copy method with the Pre-Paging fetching technique for the link speed of 1 Gbps and a network configuration delay (T_{net}) of 700 ms. The page fault processing time (T_{f_p}) was assumed to be 5 ms with a memory page size (P_z) of 4 Kilobytes and window size (w) of 4 memory pages. The total migration time increases as the Page Fault Probability increases. The other two Post- copy methods have the same behavior with Page Fault Probability change.

Figure 3.4 (b) shows the total transferred data for the Post-copy methods with relation to the Page Fault Probability. For Post-copy with Active Pushing and Pre-Paging fetching techniques, the total transferred memory is always equal to the whole memory size. For Post-copy with Demand Paging fetching technique, the total amount of memory equals the memory size multiplied by the Page Fault Probability, which means that the total transferred data increases as the Page Fault Probability increases.

The hybrid-copy migration method combines Pre-copy and Post-copy. Its total migration time has the same relation with Memory Dirtying Rate as the Pre-copy and the same relation with the page fault probability as for the Post-copy migration methods. The total amount of transferred data increases when the Memory Dirtying Rate increases.

Figures 3.5 and 3.6 show a comparison of the five migration methods in terms of downtime, total

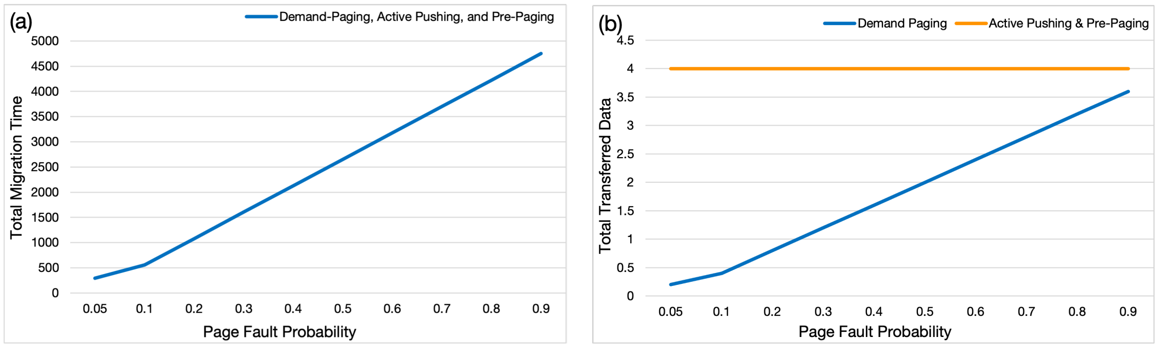


Figure 3.4: Post-copy total migration time and total transferred data with page fault probability: (a) Total migration time, (b) Total transferred data.

migration time, and total transferred data. For these results, the VM memory size (V_{mem}) was set to 4 Gigabytes with a CPU status size (V_c) of 1 Megabyte. The Pre-copy Memory Dirtying Rate was assumed to be half of the bandwidth (50% R) for the four bandwidth values of the experiments, the number of iterations was set to 10 iterations as an optimal value for the Pre-copy number of iterations from Figures 3.2 and 3.3 above, and the inter-iteration (τ) and network configuration (T_{net}) delays were both assumed to be 700 ms, the same values used in [15]. The page fault processing time (T_{f_p}) in the Post-copy method was assumed to be 5 ms with a Memory Page Size (P_z) of 4 Kilobytes.

The probability of page faults (P_f) in the Demand Paging method was assumed to be 50%, and for sake of clarity, we make this probability decrease to 20% in Active Pushing, and to 5% when using Pre-paging with window size (w) of 4 memory pages. These assumptions are shown in Table 3.1.

Figure 3.5 (a) shows the total migration time of the five migration methods for four link speeds. For higher link speeds, Pre-copy outperforms the Post-copy methods and has a much lower total migration time. This comes from the extra delay required to process all the page faults when using Post-copy methods. The Active Pushing has a lower total migration time due to the decrease in page faults probability that comes from copying the memory pages in advance of a page fault. The pre-paging technique also decreases the page faults probability and has a lower total migration time than other Post-copy methods. The Hybrid-copy method has the second-lowest total migration time after the Pre-copy method. For the lowest link speed of 10 Mbps, the Post-copy methods have lower total migration times than the Pre-copy and Hybrid-copy methods. This is because of repeatedly

Table 3.1: Experiment assumptions.

VM configrations	VM Memory Size (V_{mem})		4 GB
	CPU status size (V_c)		1 MB
	Network Configuration delay (T_{net})		0.7 s
Pre-copy	Memory Dirtying Rate (D_{mem})		0.5R
	Inter-iteration delay (τ)		0.7 s
	Number of iterations		10
Post-copy	Page fault processing time (T_{f_p})		0.005 s
	Demand Paging	Probability of page faults (P_z)	50%
	Active Pushing	Probability of page faults (P_z)	20%
	Pre-paging	Window size (w)	4
		Probability of page faults (P_z)	5%

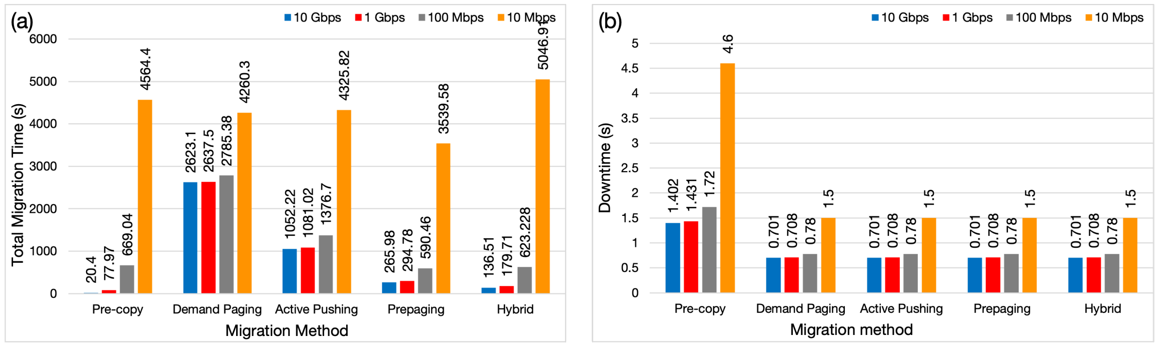


Figure 3.5: Comparison of the five migration methods: (a) Total migration time, (b) Downtime.

copying memory pages over a low-speed link when using the Pre-copy and Hybrid-copy methods, resulting in a higher copying delay and a higher total migration time.

Figure 3.5 (b) shows the downtimes of the five migration methods for four link speeds. The Pre-Copy has more downtime than other methods, which increases with the decrease of the network link speed due to having more dirty memory pages to copy in the last phase of stop-and- copy. Post-Copy and Hybrid-Copy methods have the same downtimes for different network link speeds because they use the same strategy in copying only the VM's CPU and Network states in the Post-Copy phase of the migration.

Figure 3.6 shows the total migrated data of the five migration methods for four link speeds. The total transferred data when using the Active Pushing and Pre-Paging methods equal to the whole memory size of 4 GB. When migrating using the Demand Paging, we will have the least amount of transferred data which is the total memory pages that are requested by the destination host after copying the VM's CPU state and resuming running there.

When using the Hybrid-copy method, the total transferred memory is the summation of the transferred data of one Pre-copy round and the transferred data of applying the Pre-paging Post-copy method on the rest of the memory pages. This is larger than those of Active Pushing and Pre-Paging methods.

The Pre-copy migration method has the most total transferred data. This comes from copying the memory pages multiple times. When the link speed increases, the iteration delay will decrease and the remaining memory pages after each iteration will be larger than that when using lower link

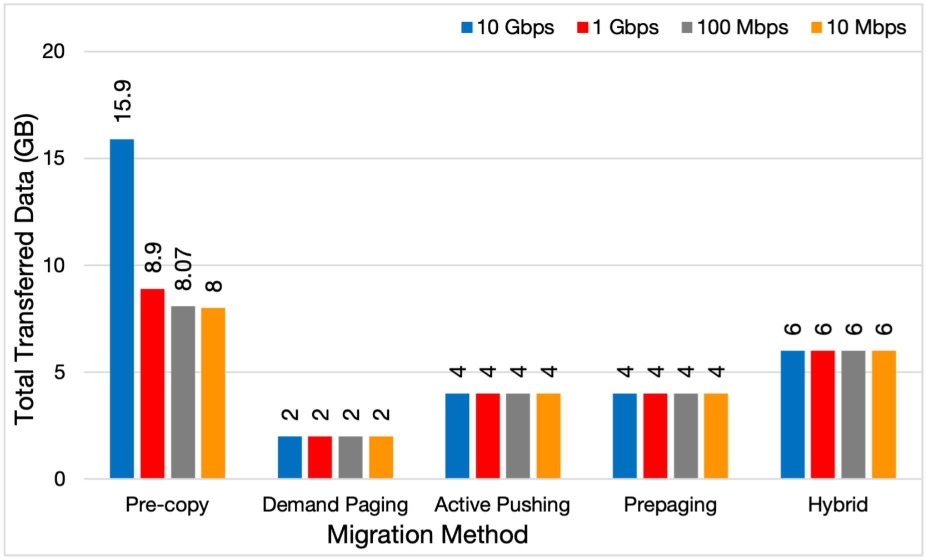


Figure 3.6: Total migrated data of the five migration methods for different link speeds.

speeds. Also, the memory pages that remain after the last iteration and are being copied during the stop and copy phase will be larger.

The Pre-copy method appears to have better migration performance than other methods for higher link speeds, in terms of total migration time but longer downtime and larger total migrated data. The Post-copy methods have the least downtime and total data to transfer, but longer total migration time because of processing the page faults. For lower link speeds, the Post-copy methods outperform the Pre-copy method with lower total migration time. This is because of repeatedly copying memory pages more than one time over a low-speed link when using the Pre-copy and Hybrid-copy methods. The Active Pushing Post-copy method decreases the probability of memory page faults and enhances the total migration time. Also, using the Pre-paging technique to copy the requested memory page with copying the surrounding memory pages in advance will decrease the page fault probability and enhance the total migration time.

The Hybrid-copy method combines the Pre-copy and the Post-copy to have the best of both methods. It transfers the whole memory at the beginning of the migration with one Pre-copy round and then copies the remaining dirty memory pages using the Post-copy method. This method gives a lower total migration time than the Post-copy method but is larger than that of the Pre-copy method for high link speeds. But the Hybrid-copy method gives the downtime of the Post-copy method which is less than what Pre-copy gives.

In the Pre-copy method, the Memory Dirtying Rate directly affects the migration performance parameters. Larger dirtying rates will increase the total migration time and downtime and causes the transfer of more memory pages. Also, the Memory Dirtying Rate should be less than the bandwidth of the link used for migration for the Pre-copy to converge and work properly. If the Memory Dirtying Rate is larger than the migration Bandwidth, the memory pages will be dirtied at a rate larger than that they are copied, which affects the VM's performance, causes to copy more memory pages, and increases the total migration time and downtime.

Another important factor that was shown in the results above is the number of iterations of the Pre-copy method. This value has a direct effect on the migration parameters and should be chosen to have the least total migration time, downtime, and total transferred data.

3.2 Dynamic Hybrid-copy VM Migration Method

In this section, we present the mathematical model for the Dynamic Hybrid-copy live migration method for the total migration time, downtime, and total transferred data during the migration process. We compare the Dynamic Hybrid-copy with other classical live migration methods. We implement the models on JAVA and evaluate the performance of the Dynamic Hybrid-copy and compare it with other methods for different experimental parameters [17]. Based on the performance results, we show how choosing the effective number of Pre-copy iterations leads to better migration performance.

3.2.1 Dynamic Hybrid-copy Method Mathematical Modeling

The original Hybrid-copy migration mechanism has one Pre-copy round which copies all the memory pages, followed by a Post-copy phase to fetch the memory pages that became dirty (modified) during the Pre-copy process. This mechanism will work fine for write-intensive workloads, but having more Pre-copy rounds will significantly reduce the read-faults during the Post-copy phase for read-intensive workloads [16].

The Dynamic Hybrid migration is a migration mechanism that combines the Pre-copy and the

Post-copy methods, where zero, one, or multiple Pre-copy iterations are done before starting the Post-copy phase [88]. The number of Pre-copy iterations is controlled by the three main parameters: The Memory Dirtying Rate, the Memory Size, and the Bandwidth allocated for the migration process. The mathematical model of the proposed migration method is explained. The model is implemented, and a set of figures are presented to show the performance behavior of the proposed method with the change of the three migration parameters.

The Dynamic Hybrid migration method consists of zero, one, or multiple Pre-copy iterations and is followed by the Post-copy for the remaining memory pages after the Pre-copy finishes. The page fetching technique that is considered for the Post-copy phase is the Pre-Paging fetching technique, where a window with multiple memory pages is copied when a page fault is processed. The total migration time, in this case, comes from two parts and is given by Equation 13.

$$T_{mig} = T_{mig-Pre-copy} + T_{mig-Post-copy}, \tag{13}$$

where $T_{mig-Pre-copy}$ is the migration time of the Pre-copy part and is given by Equation 1, excluding the last iteration (The stop-and-copy phase) and rewritten as follows:
$T_{mig-Pre-copy} = \frac{V_{mem}}{R}\frac{1-\lambda^{\mathrm{n}}}{1-\lambda} + \tau\frac{n(1-\lambda)-\lambda(1-\lambda^{\mathrm{n}})}{(1-\lambda)^2}$.

$T_{mig-Post-copy}$ is the migration time of the Post-copy part and is given by Equation 9, and rewritten as follows:
$T_{mig-Post-copy} = (1 - wP_f)\left(\frac{V'_{mem}-V_c}{R}\right) + P_f\left(T_f + \frac{(w-1)P_z}{R}\right)\left(\frac{V'_{mem}-V_c}{P_z}\right)$,
where (V'_{mem}) is the rest of the memory after the Pre-copy iterations and is given by:
$V'_{mem} = V_{mem}\lambda^{\mathrm{n}} + \tau D_{mem}\frac{1-\lambda^{\mathrm{n}}}{1-\lambda}$.

The downtime of the Dynamic Hybrid method is equal to the downtime of the Post-copy method and is given by Equation 14.

$$T_{down} = \frac{V_c}{R} + T_{net}. \tag{14}$$

The total data transferred M has two parts, the Pre-copy part and the Post-copy part, and is given by Equation 15.

$$M = M_{Pre-copy} + M_{Post-copy}. \tag{15}$$

The Pre-copy part of the total transferred data is given by:

$$M_{Pre-copy} = \begin{cases} 0 & \text{, n = 0,} \\ V_{mem} & \text{, n = 1,} \\ V_{mem} + V_{mem}\lambda\frac{1-\lambda^{\text{n-1}}}{1-\lambda} + \tau D_{mem}\left((n-1) - \frac{1-\lambda^{\text{n}}}{1-\lambda}\right) & \text{, n} > 1. \end{cases}$$

The Post-copy part of the total transferred data is the remaining memory after the last Pre-copy iteration and is given by:

$$M_{Post-copy} = V_{mem}\lambda^{\text{n}} + \tau D_{mem}\left(\frac{1-\lambda^{\text{n}}}{1-\lambda}\right).$$

Figure 3.7 shows a timeline for the four migration methods: Pre-copy, Post-copy, hybrid- copy, and the dynamic hybrid-copy. Pre-copy has the highest downtime among the four methods, but it has the least total migration time. Post-copy has a very short downtime, compared to Pre- copy, but it has the longest dependency period on the source host during the memory pages fitch phase. This affects the performance of the running applications on the migrated VM and causes serious failures in case of losing the network connection between hosts during the migration.

3.2.2 Results and Discussion

The equations above were implemented in CloudSim simulator and different experimental environments were tested. Figures 3.8, 3.9, and 3.10 were obtained to compare the Dynamic Hybrid-copy migration method with the Pre-copy, the Hybrid-copy, and the Post-copy. The experiments were conducted for one VM with memory size of 4 G bytes, Bandwidth of 1 Gbps, and four different memory dirtying rates (310 Mbps, 512 Mbps, 720 Mbps, and 925 Mbps which are equivalent to λ values of 0.3, 0.5, 0.7, and 0.9 respectively. A comprehensive integration of the mathematical models within the CloudSim simulator is detailed in Section 3.2.4 of this chapter, which presents a complete cloud simulation environment, including multiple virtual machines (VMs) and hosts.

Total migration times of the four methods are shown in Figure 3.8. At zero Pre-copy iterations, the Dynamic Hybrid-copy has the same migration time as the Post-copy. As the number of iterations increases, the total migration time of the Dynamic Hybrid-copy decreases toward the Pre-copy

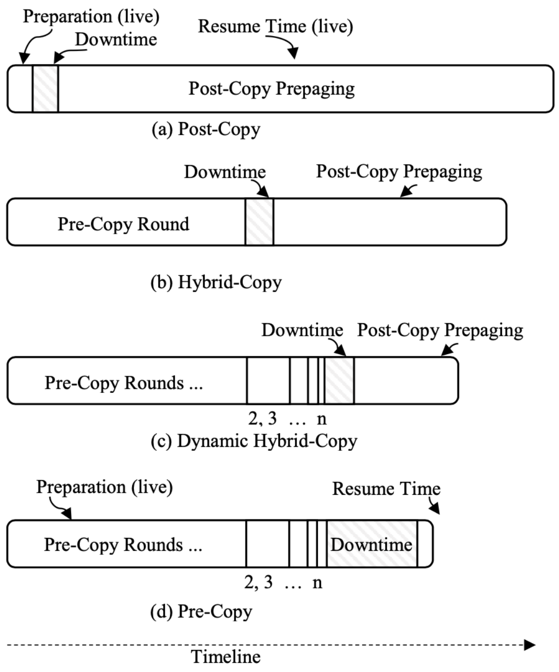


Figure 3.7: Timeline of the four migration methods.

migration time. In contrast, the Pre-copy migration time increases with more iterations done as shown in Figure 3.8 (a), (b), and (c). As dirtying rates go higher, Dynamic Hybrid-copy will start to have an opposite behavior, similar to the Pre-copy behavior, and start to increase with more copy iterations as shown in Figure 3.8 (d). This happens because of the high speed of dirtying memory which leaves greater amount of data for the Post-copy phase which takes more migration time. Post-copy migration time does not change with the number of iterations and with the dirty rate. It has a constant value as shown in Figure 3.8. Hybrid-copy migration time increases with the increase of dirtying rate and does not change with the change of number of iterations.

Figure 3.9 shows the downtime of the four migration methods with the change of the number of pre-copy iterations for different memory dirtying rates. The Post-copy, Hybrid-copy, and Dynamic Hybrid-copy have the same duration for the downtime. Pre-copy has a much higher downtime than the other three methods because in addition to the CPU and network states transferred, a variable amount of memory will also be transferred at the stop-and-copy phase, just after the Pre-copy iterations are done. The downtime of Pre-copy decreases with the increase of the Pre-copy iterations. With the increase of the memory dirtying rate, the downtime of Pre-copy will increase, and this

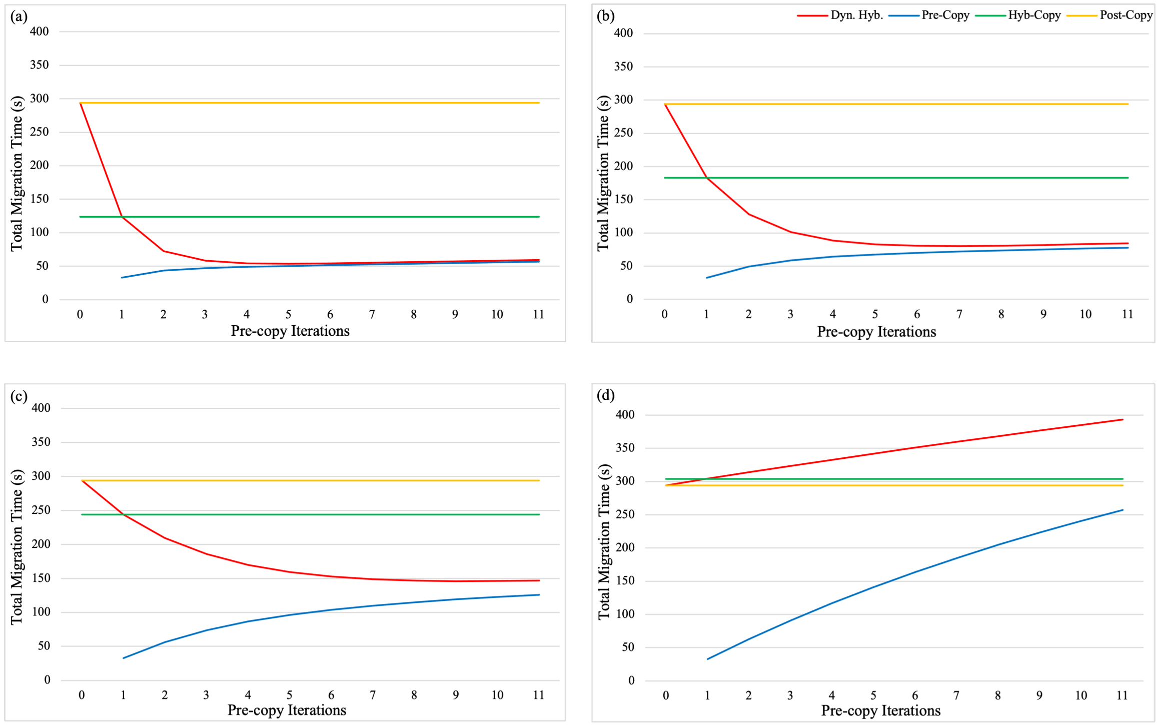


Figure 3.8: Migration times of the four migration methods with bandwidth of 1 Gbps. (a) Dirtying rate = 310 Mbps; (b) Dirtying rate = 512 Mbps; (c) Dirtying rate = 720 Mbps; (d) Dirtying rate = 925 Mbps.

comes from having more dirty memory pages in the last phase after the last Pre-copy iteration.

Figure 3.10 shows the total amount of transferred data over a 1 Gbps link of one VM with 4 GB of memory using the four migration methods for different dirtying rates. Hybrid-copy and Post Copy have a constant amount of transferred data in each case. On the other hand, the total size of data transferred in Pre-copy and Dynamic Hybrid-copy increases as the number of copy iterations increases in each case of the dirtying rates cases.

The difference between Pre-copy and Dynamic Hybrid-copy is in the second phase of the migration. Pre-copy stops the VM, transfers the CPU state and the remaining dirty memory pages to the destination host, and starts the VM there. Dynamic Hybrid-copy stops the VM to copy the CPU state, runs the VM at the destination host, and then starts a Post-copy phase to fetch the remaining dirty memory pages. Although they work differently, the same amount of data is transferred in both methods. As shown in Figure 3.10, the total transferred data increases to more than eight times the VM memory size when the dirtying rate goes closer to the migration network link speed. Table 3.2

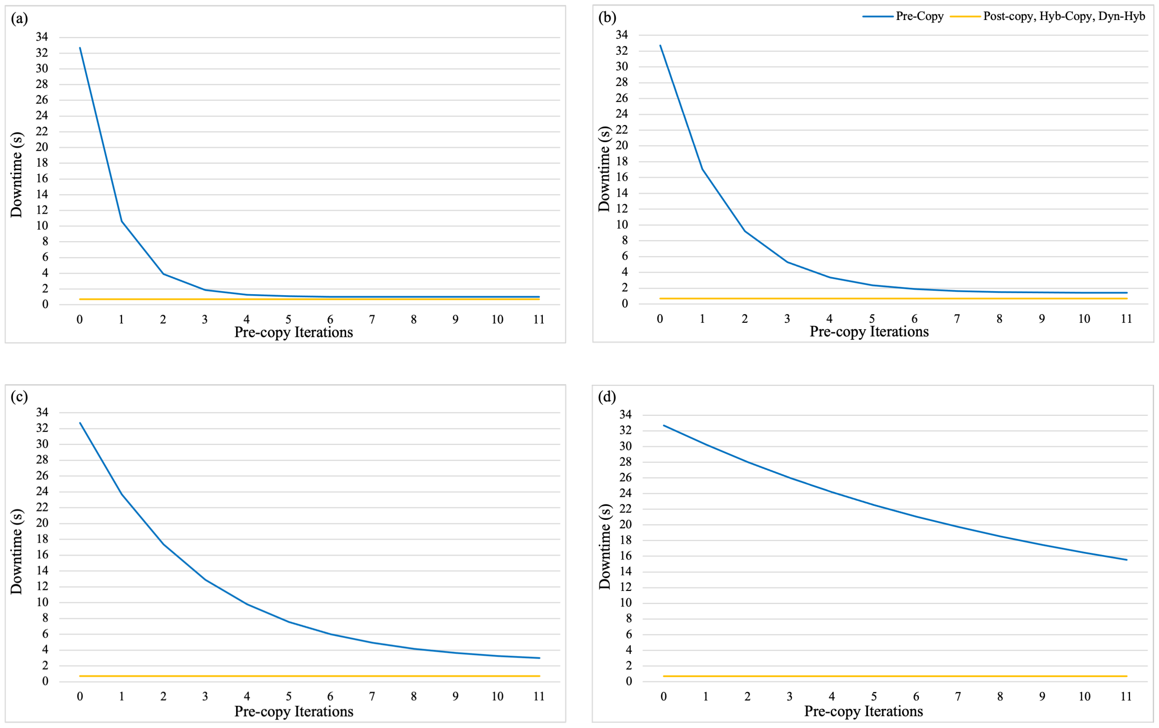


Figure 3.9: Downtimes of the four migration methods with bandwidth of 1 Gbps. (a) Dirtying rate = 310 Mbps; (b) Dirtying rate = 512 Mbps; (c) Dirtying rate = 720 Mbps; (d) Dirtying rate = 925 Mbps.

shows a comparison of the four migration methods.

Dynamic Hybrid-copy can be efficiently used to migrate a certain VM and copy less amount of data with the least migration time by choosing the optimal number of copy iterations in the Pre-copy phase. The lower the number of copy iterations done in the Pre-copy phase, the lower is the amount of memory data to copy. On the contract, having lower copy iterations in the Pre-copy phase will result in having more migration time due to the larger number of dirty memory pages remaining in the Post-copy phase.

Figure 3.11 (a) shows the migration time of migrating a 4 GB VM over a 1 Gbps link for different dirtying rates when using the Dynamic Hybrid-copy migration method. Migration time increases as dirtying rate increases for a fixed number of Pre-copy iterations. However, the downtime, shown in Figure 3.11 (b), is independent of dirtying rate and is constant for any number of iterations. This is because Dynamic Hybrid-copy has a downtime equal to that of Post-copy and it is the delay of copying the VM CPU and network state only without any memory pages. Figure

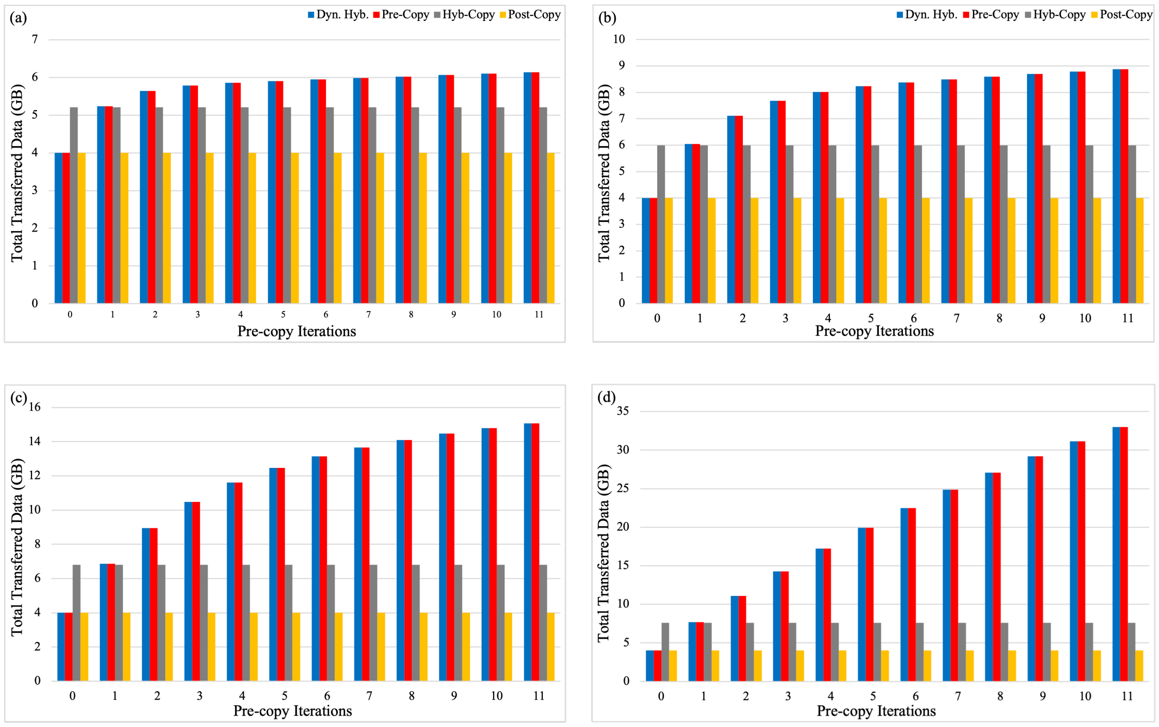


Figure 3.10: Total amount of transferred data of the four migration methods with bandwidth of 1 Gbps. (a) Dirtying rate = 310 Mbps; (b) Dirtying rate = 512 Mbps; (c) Dirtying rate = 720 Mbps; (d) Dirtying rate = 925 Mbps.

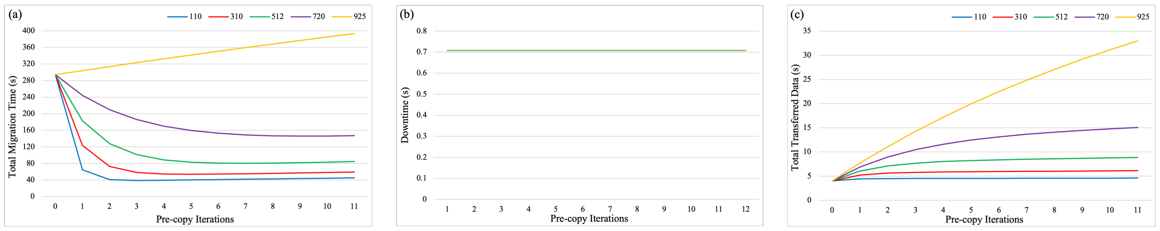


Figure 3.11: Dynamic Hybrid-copy migration of a single VM with 4 GB of memory, over a bandwidth of 1 Gbps, and for five dirtying rates. (a) Migration time; (b) Downtime; (c) Total transferred data.

3.11 (c) shows that the amount of transferred data increases with the increase of copy iterations and dirtying rate. Having more pre-copy iterations means copying more memory pages resulting in increasing the total amount of transferred data. As the dirtying rate increases, more memory pages will be updated and copied to the destination in the Pre-copy and Post-copy phases.

Table 3.2: Comparison of the four migration methods.

Migration Method	Number of Copy iterations	Dependency on Source Host	Downtime	Amount of Transferred Data	Migration time
Pre-copy	Fixed number	During copy iterations (VM runs at source)	High (Delay of transferring CPU and Network states and dirty Memory pages)	High (Repetitive copy of dirty Memory pages)	Depends on dirty rate, number of iterations, and bandwidth
Hybrid-copy	1 round	During Pre-copy phase (VM runs at source) During Post-copy phase (VM runs at destination)	Low (Delay of transferring CPU and Network states only)	Depends on dirty rate (Memory + dirty pages in Pre-copy phase)	Depends on dirty rate and bandwidth
Dynamic Hybrid-copy	Number chosen based on dirtying rate, Memory size, and Bandwidth	During Pre-copy phase (VM runs at source) During Post-copy phase (VM runs at destination)	Low (Delay of transferring CPU and Network states only)	Depends on dirty rate and number of copy iterations	Depends on dirty rate, number of iterations, and bandwidth
Post-copy	No copy iterations	During Memory migration (VM runs at destination)	Low (Delay of transferring CPU and Network states only)	Exactly the memory size	Depends on Page Fault rates at destination

The key in Dynamic Hybrid-copy is choosing the optimal number of copy-iterations in the Pre-copy phase for each VM migrated. Deciding the optimal value of copy iterations is mainly based on the memory size of the VM, the dirtying rate of the memory pages, the network link speed used for the migration, and accessing the memory pages that affect the page faults in the Post-copy phase. The mathematical equations we presented can be used in real systems to find out the best migration method to use for a certain VM to be migrated in order to have the best migration performance with the least migration and interruption delays. Using the Dynamic Hybrid-copy for migration in cloud systems helps in having the best of Pre-copy and Post-copy methods by dynamically approaching the Pre-copy or Post-copy methods by changing the number of copy iterations in the Pre-copy phase.

3.2.3 Optimizing the Dynamic Hybrid-copy VM migration method

The dynamic hybrid-copy method is dynamically controlled to have the optimal number of copy iterations in the pre-copy phase to migrate the VM with the least possible migration time and to transfer the least amount of data over the network. Many methods can be used to predict the best number of pre-copy iterations. Choosing the optimal number of copy-iterations will result in having the best of both pre-copy and post-copy migration methods, with less migration time than the post-copy alone and transferring less amount of data than the pre-copy method alone while having the downtime of the post-copy method which is the least among all migration methods.

The dynamic hybrid-copy migration method presented in [17] has several pre-copy iterations in the pre-copy phase to iteratively copy the dirty memory pages or disk blocks. The number of pre-copy iterations must be chosen such that the least migration time and amount of transferred data are achieved. Figure 3.12 shows the Total migration times and amounts of transferred data when migrating a VM with 15 GB of data (Memory pages and Desk Blocks) over a link speed of 1 Gbps, for different dirtying rates and numbers of pre-copy iterations.

The optimal value of pre-copy iterations is different for each one of the cases in Figure 3.12.

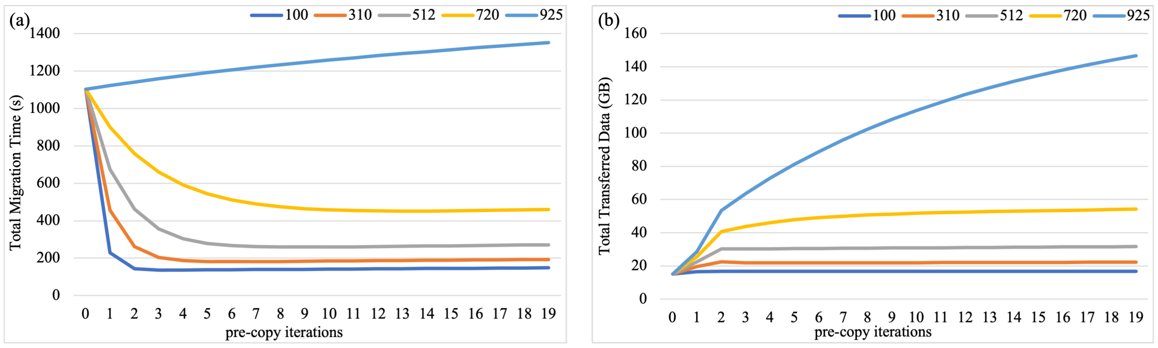


Figure 3.12: Migration performance metrics of a 15 GB VM. (a) Total migration time, (b) Total transferred data.

For example, the optimal number of iterations for the dirtying rate of 100 Mbps would be a value equal to or higher than 2. For dirtying rate of 512 Mbps, the optimal number of iterations would be a number around 6. And for dirtying rate of 720 Mbps, the optimal number of iterations would be more than 11. Keeping in mind that with more copy iteration more data will be transferred, the optimal number should be chosen to have the least possible migration time with minimum data to transfer. The optimal number of pre-copy iterations for the dirtying rate 925 is 0, which means having no pre-copy phase and migrating using only post-copy.

Algorithm 1 finds the number of copy iterations which gives the first minimum total migration time starting from the left of the total migration time graph. Tolerance of a threshold time is applied, meaning if the next value gives a lower migration time with a difference lower than a predetermined threshold it will not be chosen and the previous value with a higher total migration time is chosen to avoid transferring more data by having an extra iteration.

Algorithm 1 Find the best number of pre-copy iterations (n).

Input: Bandwidth (R), Memory size (M), Memory Dirtying Rate (D)
Output: The best number of pre-copy iterations (n)

```
1:  $n \leftarrow 0$
2:  **Compute_**$T_0(n)$
3:  $i \leftarrow 1$
4:  **while** $i < 20$ **do**
5:      **Compute_**$T_i(n+1)$
6:      $delta \leftarrow (T_0 - T_i)$
7:      **if** $delta < 0$ **or** $delta \leq threshold$ **then**
8:          $break$
9:      **end if**
10:     $n \leftarrow i, T_0 \leftarrow T_i, i \leftarrow i+1$
11: **end while**
12: **return** n
```

Applying Algorithm 1 to the data in Figure 3.12 with a threshold of 5 seconds will give the optimal numbers of copy iterations for different dirtying rates shown in Table 3.3. By matching these numbers with the graphs of Figure 3.12, we can say that the number of copy iterations for each case of the dirtying rates cases gives the least possible Total Migration Time with the least possible number of copy iterations to transfer fewer amounts of data.

Memory dirtying rate and provisioned bandwidth play a vital role in the total migration time,

Table 3.3: Optimal number of copy iterations.

Dirtying Rate (Mbps)	**Number of Copy Iterations**	**Total Migration Time (s)**	**Total Transferred Data (GB)**
100	3	136.758	16.673
310	4	186.202	21.757
512	7	261.466	30.613
720	10	457.727	51.717
925	0	1102.968	15

downtime, and amount of data to transfer during the migration process. Prediction of these two parameters is important to find the optimal number of copy iterations (n) for the pre-copy phase of the dynamic hybrid-copy migration method.

The remaining number of memory pages after the pre-copy phase directly affects the duration of the post-copy phase of the dynamic hybrid-copy migration. When migrating over WAN, the post-copy phase should be minimized to decrease the dependency time on the source host and avoid the drawbacks of post-copying the VM data. This parameter is calculated before the migration process and then the migration settings are set with optimal parameters to guarantee the least total migration time and downtime, as well as the least source host dependency time in the post-copy phase.

3.2.4 CloudSim Integration

CloudSim, which is a framework for modeling and simulation of cloud computing infrastructures and services [10] is used to test the mathematical models shown above. The four migration methods: Pre-copy, Post-copy, Hybrid-copy, and the basic Dynamic Hybrid-Copy, without optimizations, are integrated with CloudSim. The Median Absolute Deviation (MAD) VM allocation policy and Minimum Migration Time (MMT) VM selection policy [89] are used to run a simulation scenario of 50 VMs and 50 hosts on one datacenter. The VM memory sizes are chosen from the set: 870, 1740, 1740, 613 MB, the memory dirtying rate was set to 512 Mbps for all VMs. The Hosts are 4 GB, and the bandwidth is 1 Gbps.

The four migration methods: Pre-copy, Post-copy, Hybrid-copy, and Dynamic Hybrid-copy were tested for this experiment, resulting in 5265 migrations in each case. The summations of all migration times, downtimes, and total transferred data were calculated. Figure 3.13 (a) shows the

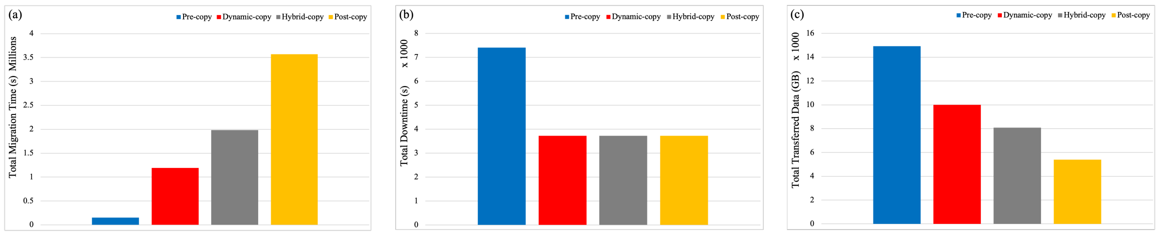


Figure 3.13: Results of CloudSim simulation of 50 VMs, 50 hosts, and 1 data center. (a) Total migration time, (b) Total downtime, (c) Total transferred data.

summation of total migration times for all the migrations for the four migration methods. Figure 3.13 (b) shows the summation of total downtimes for all migrations, and Figure 3.13 (c) shows the summation of total data transferred for all migrations. Figure 3.13 shows that the Pre-copy migration method results in having the least total migration time among all other methods, but the highest downtime and total transferred data.

Post-copy transfers the least amount of data with the least downtime among other methods, but it has a much higher migration time. Hybrid-copy transfers a higher amount of data than Post- copy but less than Pre-copy and Dynamic Hybrid-copy with the same downtime as that of Post- copy. It has a lower total migration time than Post copy but higher than Dynamic and Pre-copy methods. Dynamic Hybrid-copy has the same downtime as that of the Post-copy which is much less than Pre-copy downtime. It has a higher total migration time than Pre-copy but less than the Hybrid-copy and Post-copy methods. For the total transferred data, Dynamic Hybrid-copy transfers less amount of data than Pre-copy but higher than Hybrid-copy and Post-copy methods.

3.3 Chapter Summary

In this chapter, the mathematical models for the classical live VM migration methods Pre-copy, Post-copy, and Hybrid-copy and the Dynamic Hybrid-copy that combines both Pre-copy and Post-copy have been presented. The Dynamic Hybrid-copy method dynamically selects the optimal number of pre-copy iterations in the Pre-copy phase of the method to transfer less amount of VM's memory and storage data with the least migration time and downtime when migrating over WAN connections. Comparisons have been presented between different migration methods for different connection speeds and different VM metrics.

The results showed how the number of iterations of the Pre- copy method affects the total migration time, downtime, and total transferred data for the migration process. It also showed the effect of the Memory Dirtying Rate on the three migration parameters. Pre-copy results in a lower total migration time than other migration methods, but more downtime and a larger amount of transferred data for higher link speeds. For lower link speeds, the Pre-copy was the worst with larger total migration time than other methods. Pre-copy always copied larger memory pages because of repeatedly copying memory pages for a certain number of iterations. Post-copy methods had larger total migration time than Pre- copy and Hybrid-copy methods, but it had the least downtime and copied the least amount of memory pages than other methods. By enhancing the Post-copy methods by using some techniques to decrease the page fault probability, the total migration time was decreased, but it still did not outperform the Pre-copy method. The Hybrid-copy migration method combines the best of the Pre-copy and the Post-copy. It results in a lower total migration time than the Post-copy method, and lower downtime and total transferred data than the Pre-copy method.

The results also showed the effect of number of copy iterations in the Pre-copy phase of the Dynamic Hybrid-copy on the total migration time and total data transferred over the network. They also showed the effect of the memory dirtying rate on these two migration parameters. Downtime of the Dynamic Hybrid-copy is similar to the downtime of the Post-copy which is the least among all migration methods used. Dynamic Hybrid-copy was compared with the three classical migration methods: Pre-copy, Post-copy and Hybrid-copy. Dynamic Hybrid-copy downtime delay is equal to that of Post-copy and Hybrid-copy which is much less than the Pre-copy's downtime. The total migration time of Dynamic Hybrid-copy is higher than the Pre-copy's migration time because of the memory fetching at the Post-copy phase of the Dynamic method. Total migration time of Dynamic Hybrid-copy is less than that of both the Post-copy and Hybrid-copy because of the iterative copy of memory which reduces the amount of remaining dirty memory pages that will be fetched in the Post-copy phase. The total transferred data in the Dynamic Hybrid-copy is less than the Pre-copy when having a smaller number of copy iterations and it is higher than the Post-copy and Hybrid-copy because of iteratively copying memory pages.

The equations of mathematical models of the four migration methods were implemented over the CloudSim simulator, and an experiment for 50 VMs and 50 hosts was conducted. The results of using each of the four migration methods were compared and showed that Pre-copy has the lowest migration time and the highest downtime. It also transfers larger amounts of data over the network than other methods. Dynamic copy transfers a lower amount of data than Pre-copy and has a lower downtime than Pre-copy.

The key in Dynamic Hybrid-copy is choosing the optimal number of copy-iterations in the Pre-copy phase for each VM migrated. Deciding the optimal value of copy iterations is mainly based on the memory size of the VM, the dirtying rate of the memory pages, the network link speed used for the migration, and accessing the memory pages that affects the page faults in the Post-copy phase.The mathematical equations we presented can be used in real systems to find out the best migration method to use for a certain VM to be migrated in order to have the best migration performance with least migration and interruption delays. Using the Dynamic Hybrid-copy for migration in cloud systems helps in having the best of Pre-copy and Post-copy methods by dynamically approaching the Pre-copy or the Post-copy methods by changing the number of copy iterations in the Pre-copy phase.

The mathematical models presented in this chapter are valuable to study the performance of cloud systems before and after migration. However, these models fail to account for all the migration characteristics and performance metrics that can affect the migration process, which makes their performance predictions imprecise for these migration methods. Furthermore, these models do not consider the significance of SLA requirements, which play a crucial role in managing real cloud systems. In the next chapter, regression models for VM migration performance are built using machine learning algorithms. The models account for additional migration parameters and SLA constraints that cannot be mathematically modeled. The accurate prediction of performance for different migration methods can aid in selecting the best approach for a VM with specific characteristics. This is particularly crucial in low bandwidth environments like WAN to minimize migration time, downtime, and data transfer.

Chapter 4

Improving VM Migration Across Cloud and Edge Data Centers

Analyzing global and local VM migration data provides a comprehensive understanding of data center status, enabling accurate decision-making on live VM migration. A centralized management approach, storing data center information in a central database, ensures consistency and effective scheduling. However, this approach comes with challenges such as single point of failure, security concerns, contention access, and potential conflicts when migrating VMs over a WAN [12]. Careful evaluation of these factors and appropriate risk mitigation measures are crucial for successfully dealing with large data models.

A distributed cloud management approach that extends from the data center to edge computing can optimize resource allocation and reduce communication overhead [13]. This approach involves each data center maintaining a duplicate of the status of all other centers. The cloud manager ensures that the duplicate copy is up-to-date with the latest version of VMs and data center status across all data centers and facilitates direct communication between them to enable WAN migration. Integrating blockchain technology into the scheduling process can simplify WAN migration by securely and efficiently sharing the status of all data centers. This approach minimizes the potential negative impacts of WAN migration by involving all data centers in the VM migration process while considering both WAN and LAN VM migration considerations.

In this chapter, we propose a novel distributed cloud management solution that employs blockchain technology to facilitate efficient sharing of VM statuses across multiple data centers. Our blockchain-based model reduces the total messages exchanged for the VM migration with percentages ranging from 0.5% to 22% and the total communication delay by 8% to 72% compared to other distributed technologies. The proposed solution also reduces the number of communication messages by 41.79% to 49.85% and total communication delay by 2% to 12%, as compared to a VPN-based solution. Additionally, we propose a machine learning and deep learning regression-based VM migration prediction service. This service accurately predicts migration metrics for each VM based on its unique characteristics and identifies the most suitable live migration method. We integrate the proposed prediction service over the novel distributed management framework that utilizes blockchain technology to enable efficient VM status sharing among multiple cloud data centers. This approach enhances transparency and reliability in sharing VM status information, thereby improving VM migration. Our proposed solution enhances the speed and accuracy of migration decisions, minimizes migration time and downtime, and ensures SLA compliance. The SLA compliance rate of the proposed solution ranges from 18% to 94.9% for different machine learning algorithms and SLA policies during VM migrations across WAN-connected data centers. The proposed solution reduces the total migration time by 14% to 79% and the downtime by 64% to 99%.

4.1 Blockchain-Based Distributed Framework for VM Migration Across Multi-Cloud Data Centers

Blockchain is a widely used distributed ledger technology that is utilized to create and maintain a tamper-resistant digital record of transactions stored on a peer-to-peer network. The decentralized nature of blockchain technology eliminates the need for a centralized authority, providing a secure and cost-effective way of conducting transactions between untrusted parties. This technology has the potential to significantly transform various socio-economic systems, including but not limited to, cloud computing, Internet of Things (IoT), healthcare, electronic voting, and social networking. In particular, the use of blockchain technology in pervasive environments offers a promising solution to address challenges related to data privacy, network security, and decentralization, while the elastic and scalable nature of such environments can enhance the efficiency of blockchain operations.

The predominant data communication model used in both industrial and academic research for data exchange is the publication of data via REST APIs. In the context of log monitoring, two primary techniques are utilized: on-demand log retrieval and continuous periodic log pushing. The logs are then stored in a centralized management system. For example, Prometheus, an open-source tool, retrieves data at customizable intervals [90]. Additionally, Grafana is employed to visualize and correlate the collected data [91].

In standard cloud management methods, such as Nova [92], compute nodes publish generated logs via REST APIs. Cloud Foundry [93] accesses these low-level logs through the published REST services.

However, employing this architecture in a distributed cloud management model has limitations, such as the security of communication, system scalability in large-scale cloud environments, and the volume of communication messages required for various management tasks. Blockchain-based distributed management addresses these challenges by facilitating efficient and secure communication and status message sharing among numerous communication nodes. This approach enhances communication performance and decision-making processes while reducing the likelihood of errors in management decisions.

The main contribution of this research is to enhance the scheduling and migration of VMs in cloud data centers by enabling effective and secure sharing of VM characteristics between different cloud data centers. Blockchain technology is integrated into a distributed cloud management model. Blockchain nodes are distributed across cloud data centers and act as a secure and efficient registry to data center status information. It streamlines WAN migration by securely and efficiently sharing the status of all data centers. With its immutable nature, it ensures that copies of VMs and data center status across all centers are consistent. In this type of management model, all data centers work together in VM scheduling across data centers, taking into account both WAN and LAN scheduling considerations. This approach minimizes the potentially adverse effects of WAN scheduling operations.

To compare our proposed solution, we implement a REST-based distributed cloud management model and a VPN-based centralized cloud management model that is based on centralized systems' definitions.

4.1.1 Proposed Model Architecture

A data center's cloud manager is made up of specialized modules with distinct functions, including Resources Scheduler Manager, Allocation Manager, Service Manager, Network Manager, and VM Migration Manager. These modules are integral to the smooth operation of any cloud manager, as they handle crucial tasks in the data center.

The Resource Scheduler Module is the backbone of the cloud manager infrastructure. Its primary objective is to regulate the host's resource usage effectively to achieve optimal performance for the entire data center. It assigns VMs to hosts based on resource availability and triggers VM migration to maintain high availability and efficient workload distribution across hosts, preventing host overloading.

Working alongside the Resource Scheduler Module, the Allocation Manager determines when and how to provide the necessary computing and memory resources for each VM. This decision is based on various factors, such as VM characteristics, workload and priority, and resource availability on each host. Optimization algorithms play a critical role in the allocation process to ensure the efficient distribution of VMs on available hosts. The Resource Scheduler and Allocation Manager work together to maximize the data center infrastructure's overall utilization by ensuring efficient and fair resource allocation.

The Allocation Manager reviews the status of the entire data center and generates a list of selected VMs to be migrated along with available destination hosts. This information is then shared with the scheduler, which selects the optimal host to ensure the VM executes efficiently without needing to migrate constantly.

To maintain optimal service quality, the Service Manager closely monitors virtual service behavior. It continuously checks response times and resource scheduling to ensure services are operating effectively. If services fall below SLA standards, the Service Manager triggers migration to maintain optimal performance while adhering to SLA policy guidelines.

The Network Manager is essential in managing communication within a data center. It ensures maximum network performance and efficient resource utilization. The module continuously monitors bandwidth usage to pinpoint and resolve congestion or other network-related issues that may impact virtual machines and other services. Additionally, the Network Manager assigns bandwidth to each VM to ensure efficient task execution, especially in data centers with limited network resources and multiple simultaneous VMs.

The Network Manager is responsible for managing all external channeling links originating from the data center, including internet access, cloud services provided by third-party providers, and connections to other data centers. By efficiently managing these links, the Network Manager ensures that network-related issues do not disrupt data center operations. Additionally, the Network Manager allocates network bandwidth to facilitate VM migrations and improve the performance of the migration process.

The VM Migration Manager module collaborates with the Scheduler and Allocation Manager modules to monitor status and populate lists of VMs and available hosts for local and WAN VM migration. The VM migration manager processes the scheduler decision and executes the migration by direct communication with the cloud manager of the destination data center, communicating with Virtual Machines Manager (VMM) resources and processing an API call to trigger the migration.

Scheduling techniques can be classified into static scheduling such as Min-Min algorithm [94, 95] and PSO based algorithm [96], Dynamic Scheduling such as the Dynamic Round-Robin algorithm [97, 98, 99], and Heuristic Scheduling by applying optimization algorithms such as the Genetic Algorithm (GA) and Particle Swarm Optimization [100]. Deep Reinforcement Learning [101, 102, 103, 104] is used in the development of scheduling strategies for VMs in cloud data centers to optimize the scheduling strategies and to reduce their delays.

This research proposes a new blockchain-based distributed cloud management model that employs blockchain as a secure registry for sharing data center information required for VM scheduling across multiple data centers.

4.1.1.1 Blockchain-based Distributed Management Model

When transferring VMs over a wide area network (WAN), the centralized model may encounter issues with precision and superfluous VM migration. To optimize resource usage and cut down costs, a blockchain-based distributed management model can be utilized, which leverages blockchain and machine learning. This model guarantees secure and adaptable scheduling of migration across all data centers, with uniform contributions from each center. Instead of having a global manager, each data center has its own distributed cloud manager that handles both local and external operations. The blockchain-based distributed cloud management model connects multiple data centers through WAN. All external data center operations are managed within each data center. This model is illustrated in Figure 4.1. Because each data center has a synchronized copy of the states of all data centers, the WAN migration decision is expected to be more precise, which reduces the number of VM migrations and eliminates VM misplacements.

The internal architecture of the blockchain-based distributed cloud manager in a data center is shown in Figure 4.2. It comprises five main modules: Resources Scheduler Manager, Allocation Manager, Service Manager, Network Manager, and VM Migration Manager, in addition to the blockchain service module.

To make a migration decision in the blockchain-based distributed model, the distributed manager follows a set of steps shown in Algorithm 2, triggering VM migration within the same data center or from a data center to another over WAN. The distributed cloud manager starts by getting the states of data centers and hosts from the local blockchain registry. The scheduler and allocation managers apply placement and scheduling algorithms to select candidate VMs and best placement data center and host. The candidate VMs and hosts list is populated to the Network and VM Migration managers. The migration can be completed locally within the same data center or across data centers via WAN. In case of a WAN migration, the source cloud manager communicate with

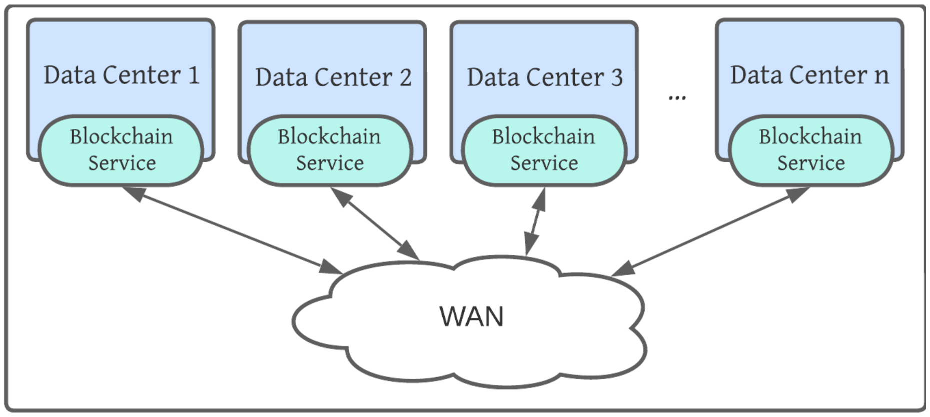


Figure 4.1: Blockchain-based distributed management model view.

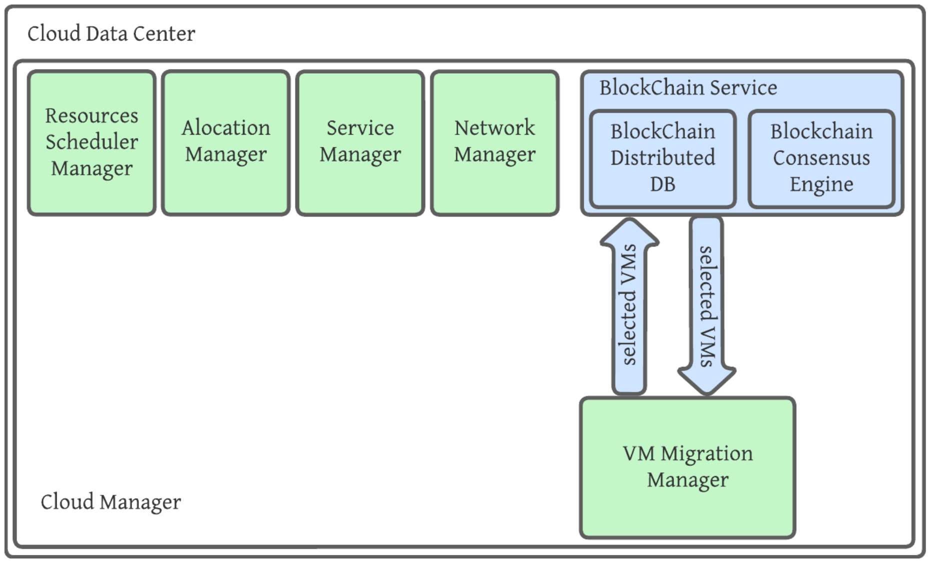


Figure 4.2: Blockchain-based distributed cloud manager internal architecture.

Algorithm 2 Blockchain-based distributed manager algorithm.

```
Get data centers states from the local blockchain register
Apply the dynamic priority-based optimization algorithm to find best placement data center and host, and VMs to migrate (LAN/WAN VM migration scheduler)
Populate list of VMs to hosts, and send notifications to the migration module of data centers
if LAN migration then
    Trigger local VM migration
    Update blockchain register with the new VM state
else if WAN migration then
    Communicate directly with the cloud manager of the destination data center to migrate over WAN
    Trigger WAN VM migration
    Update blockchain register with the new VM state
end if
Perform a Blockchain synchronization operation to update all blockchain nodes in all data centers with the new VM states.
```

the destination cloud manager to trigger the VM migration. In case of a local migration, the VM migration is triggered directly. In both cases, after the migration is done, the blockchain register is updated with the new VM state and the blockchain consensus engine starts the synchronization operation update all blockchain nodes in all data centers with the new VM states.

The blockchain-based distributed model efficiently handles both WAN and local migrations within the data center, effectively preventing VM misplacements. With just one communication message required to process a VM migration and update the blockchain register ($N_m = 1$), the total number of communication messages exchanged in the distributed model (N_{com}) is equivalent to the number of VM migrations, which can be calculated using Equation 16.

$$N_{com} = N_{mig} = N_{mig\_W} + N_{mig\_L}, \tag{16}$$

where N_{mig} is the total number of VM migrations including local and WAN VM migrations.

Equation 17 provides the formula for calculating the total delay (D_{total}) required for processing all VM migrations. It's worth noting that this formula encompasses the convergence time necessary for synchronizing the changes on all blockchain registers (T_{conv}). It's important to mention that the convergence delay comes into play only when dealing with a VM WAN migration.

$$D_{total} = T_{com} \times N_{mig} + N_{mig\_W} \times T_{conv}, \tag{17}$$

where T_{com} is the transmission and processing time for one communication message. In the distributed model, T_{com} is the same for both WAN and local VM migrations ($T_{com} = T_{com\_W} = T_{com\_L}$).

4.1.1.2 Blockchain Service

Our proposed blockchain-based distributed model involves integrating a blockchain registry service module with the VM Migration Manager on top of the cloud manager. By incorporating the Blockchain module within the cloud manager, we can securely share states of cloud data centers with each other, allowing for efficient distributed management.

The blockchain serves as a local registry in the blockchain-based distributed model, saving the state of each data center locally. Each center updates its status to the blockchain register, which then synchronizes the states with other nodes on other data centers. The platform used for the blockchain module is BigchainDB [31], which can be used with decentralized computing platforms and applications. BigchainDB combines the benefits of distributed databases and traditional blockchains, using an enterprise-grade distributed database (MongoDB) [29] with a production-ready consensus engine (Tendermint) [30] to provide high transaction rates, low latency, indexing, querying of structured data, and blockchain properties such as decentralization, immutability, and owner-controlled assets.

The latest version of BigchainDB, version 2.0, incorporates Tendermint for networking and consensus, which ensures Byzantine Fault Tolerance (BFT). Each node is equipped with its own MongoDB database and communication is carried out using Tendermint protocols. Figure 4.3 shows the major components of a four-node BigchainDB 2.0 network and how they interact with one another. The system is highly secure, as the worst possible outcome of a hacking attempt would be corruption or deletion of data in a single local database. The current block proposer in Tendermint changes with each round, ensuring a decentralized network where each node is owned and operated by a unique entity. Ideally, nodes should be situated in various locations and hosting providers to prevent single-point failures. Remarkably, even if one-third of nodes fail, the network will continue to function without interruption.

The BigchainDB network offers a highly effective means of safeguarding stored data, making it extremely difficult to alter or remove. The data is essentially immutable and each node possesses a complete set of information stored in a dedicated MongoDB database. Transactions are also secured through cryptographic signatures, altering the contents of which would affect the signature and trigger detection. These measures collectively ensure the utmost safety and security of the stored data. In addition to that, the transfer of an asset can only be executed by the rightful owner or owners who hold the necessary set of private keys. This implies that even a node operator cannot carry out the transfer of an asset.

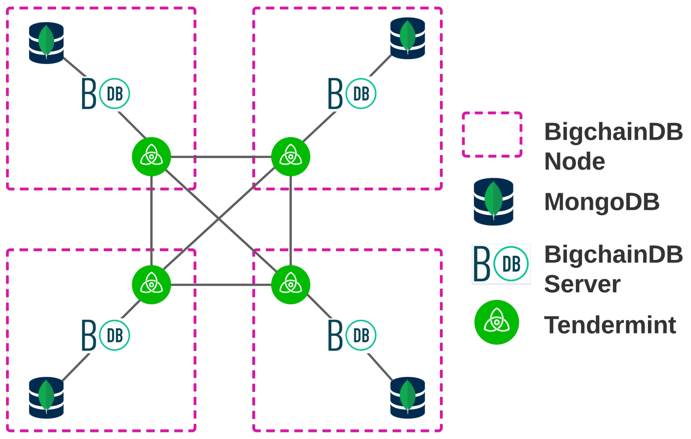


Figure 4.3: Four-node BigchainDB 2.0 network.

The Tendermint consensus protocol has been known for its remarkable performance in terms of transaction processing and commit latencies. It is capable of processing several thousand transactions per second, while maintaining a commitment latency ranging between one to two seconds. The protocol is also known to perform exceptionally well in harsh adversarial conditions, such as validators crashing or broadcasting maliciously crafted votes.

BigchainDB networks, which are based on the Tendermint consensus protocol, have also been observed to exhibit remarkable performance, with transactions being included in new committed blocks in just a few seconds or less. Additionally, the Tendermint consensus protocol ensures that once a transaction is included in a block, it cannot be reverted or considered defunct in the future, as forking is not supported.

Each node in a BigchainDB 2.0 network has its own local MongoDB database, providing node operators with access to the full power of MongoDB for indexing and querying stored data, including transactions, assets, metadata, and blocks, all of which are JSON strings. Node operators have the freedom to decide the extent to which they expose this power to external users, depending on their individual preferences. For instance, some node operators might opt to index geospatial data and offer optimized geospatial queries via a REST Application Programming Interface (API), while others might prefer to offer a GraphQL API. Although BigchainDB 2.0 creates some MongoDB indexes and the BigchainDB HyperText Transfer Protocol (HTTP) API includes some endpoints for basic queries by default, every node operator has the ability to add additional indexes and query APIs.

Unlike some blockchain networks, such as Bitcoin, which allow anyone to add their node to the network, BigchainDB networks are controlled by the governing organization behind the network, making Sybil attacks a non-issue. Bitcoin makes Sybil attacks unlikely by making them prohibitively expensive.

4.1.1.3 REST-based Distributed Model

To compare with another distributed existing solution, we have integrated REST services at each data center to facilitate the publication of logs through on-demand REST GET requests, as illustrated in Figure 4.4. The cloud management systems, distributed across all data centers, exchange messages using two mechanisms: periodic updates scheduled at 10-second intervals and on-demand polling when logs are unavailable or outdated [105].

In this scenario, the number of messages for communication updates between all data centers, assuming we have n data centers, will be $n-1$ for each data center during polling for migration, excluding periodic updates. The total number of messages for migration decision is calculated using the following equation 18. Each data center will exchange information about all participating data centers in the same cloud management domain. Subsequently, the migration decision, specifying the source and destination data centers and local information (such as the selected VMs and destination hosts), will be sent to all data centers.

$$N_{com} = 2 \times n \times (n-1) \times N_{mig\_W} \tag{18}$$

where N_{com} denotes the total number of exchanged communication messages, n denotes the number of data centers, and N_{mig_W} represents the total number of WAN VM migrations.

In the context of web server communication, the total communication delay time is determined by Little's law, which states that the average waiting time equals the average arriving rate multiplied by the average serving time of each REST web service. Here, the statistical web server service time is denoted by θ, and the request arriving rate follows a Poisson distribution with mean parameter λ

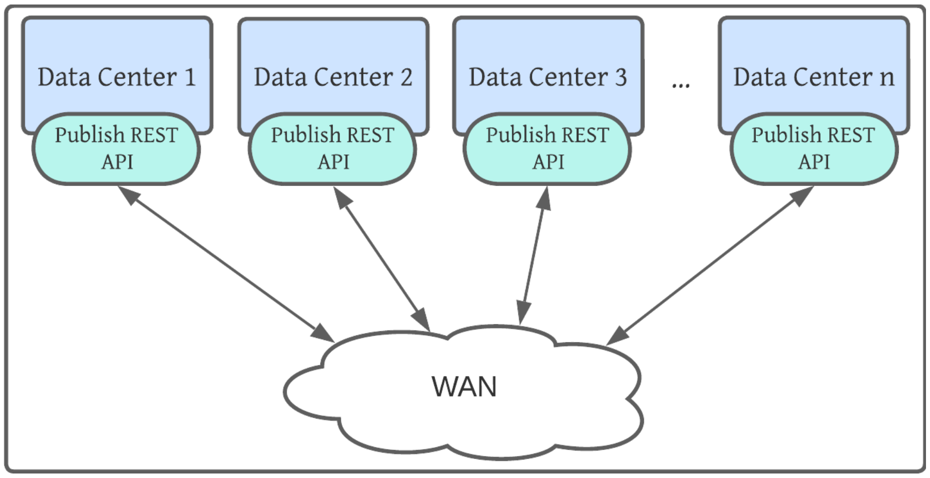


Figure 4.4: REST-based distributed management model view.

as given by Equation 19.

$$D_{total} = N_{com} \times \lambda \times \theta \tag{19}$$

4.1.1.4 VPN-Based Centralized Management Model

In cloud data centers, centralized management involve the appointment of a local cloud manager to oversee the day-to-day operations of each center. Additionally, a global manager is responsible for external operations and communication between the centers. Figure 4.5 provides an illustration of the centralized model, with the global manager linked to multiple data centers via VPN connection over WAN.

The architecture of a local cloud manager in the VPN-based centralized model is illustrated in Figure 4.6. The essential modules of the cloud manager are present in the local cloud manager, and they oversee the fundamental tasks of the cloud data center. A communication module facilitates communication with the global manager and transmits the registered resources states to enable external data center operations such as WAN VM migration.

In the VPN-based centralized model, a software known as the global manager operates on a server that is linked to all data centers via a VPN connection over WAN. The global manager collects data center states and stores them in a database. To initiate a VM migration from one data center to another over the WAN, the global manager follows a set of steps outlined in Algorithm 3.

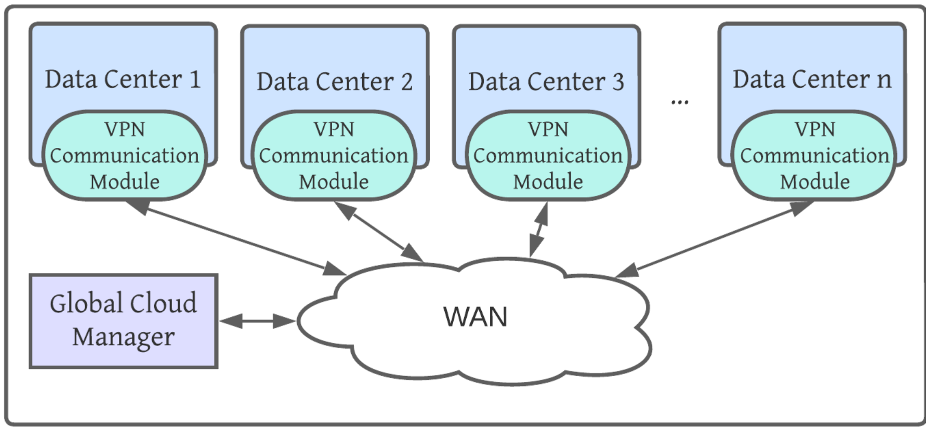


Figure 4.5: Centralized management model view.

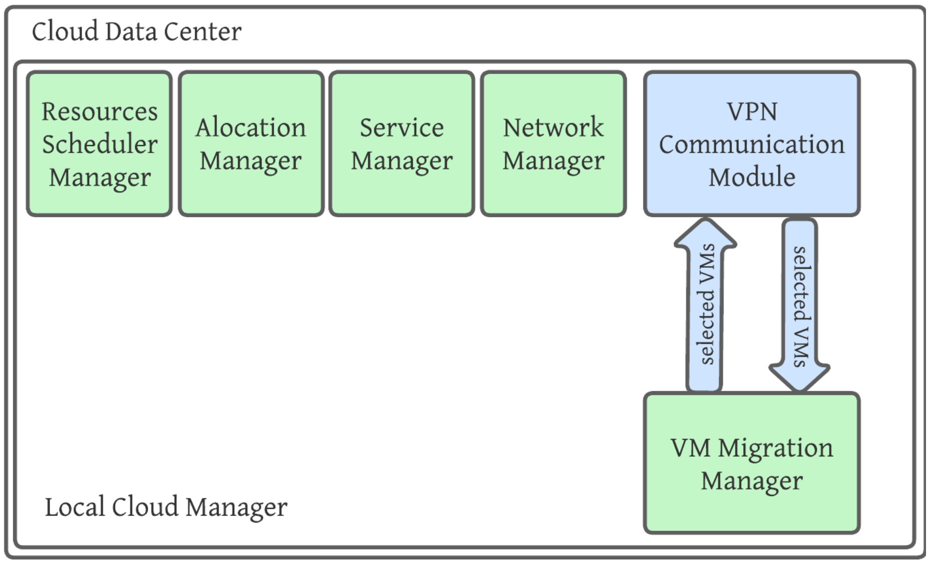


Figure 4.6: Centralized local cloud manager internal architecture.

Algorithm 3 Centralized global manager algorithm.

```
Populate data centers list
for all data centers do
    Get VMs states
    Get list of selected VMs to migrate
    Get list of available physical hosts (can receive VM in multiple data centers)
end for
Apply the dynamic priority-based optimization algorithm to find best placement data center and
host, and VMs to migrate (WAN VM migration scheduler)
Populate the list of VMs to hosts, and send notifications to the migration module of data centers
Trigger the migration
```

The VPN-based centralized database is updated with partial information regarding VMs in data centers that have been triggered for WAN migrations. This occurs when the list of populated hosts is empty, indicating that the VM should be relocated from the current data center. To optimize server consolidation, power consumption, and resource utilization, a dynamic priority-based algorithm [106] with migration facilitator scheduling is utilized. The migration facilitator assigns priorities to VMs and flags them based on their role in host allocation. VMs with higher migration orders are allocated to hosts with fewer available resources.

This approach is implemented in the centralized global manager to determine WAN migration decisions based on the available states of data centers. As the global manager does not have access to the full status of all data centers, the decision may be inaccurate causing a VM misplacement. Additionally, the state of VMs may be changing locally without the knowledge of the global manager while it is making WAN migration decisions, leading to more erroneous decisions and triggering more WAN migration with misplacements, resulting in a large number of unwanted VM migrations to reach a stable state. On the other hand, in the blockchain-based distributed model, the blockchain acts as a local registry, preserving the state of all data centers within it. Each data center updates its status to the blockchain register, which is then synchronized with other blockchain nodes in other data centers using the consensus protocol algorithm, thereby eliminating the possibility of any misplacements.

Equation 20 provides the calculation for the total number of communication messages exchanged in the centralized model, denoted as N_{com}. The total delay to transmit and process all communication messages for all VM migrations, represented by D_{total}, is given by Equation 21. This delay encompasses the transmission and processing delays for all communication messages transmitted, and is dependent upon the network speed and the execution speed of both the local and global cloud managers.

$$N_{com} = N_m \times (N_{mig\_W} + N_{mis}) + N_{mig\_L}, \tag{20}$$

where N_m is the number of exchanged messages needed to process one WAN VM migration, N_{mig_W} is the total number of WAN VM migrations, N_{mis} is the total number of VM misplacements in WAN migrations, and N_{mig_L} is the total number of local VM migrations.

$$D_{total} = N_m \times (N_{mig\_W} + N_{mis}) \times T_{com\_W} + N_{mig\_L} \times T_{com\_L}, \tag{21}$$

where T_{com_W} is the transmission and processing time of one communication message transmitted between the local and global managers. T_{com_L} is the local transmission and processing time for a local message that is processed within the data center without reaching the global cloud manager.

4.1.2 Experimental Setup

To test our proposed models, we utilized a combination of real-world implementation and simulation techniques. We constructed and executed the experimental environment at the esteemed Gina Cody School HPC Facility known as Speed [107]. The infrastructure itself boasts 24 32-core nodes with 512 GB of memory and 1 TB of volatile-scratch disk space, as well as 12 NVIDIA Tesla P6 GPUs with 16 GB of memory and 1 AMD FirePro GPU with 8 GB of memory. Additionally, the facility provides Singularity containers (which are user-managed containers) and support for a variety of machine learning and deep learning frameworks, including TensorFlow, OpenCV, OpenMPI, OpenISS, MARF, and OpenFOAM.

Several containers were constructed to run data center simulators with varying numbers of hosts and distributed VMs. The data center simulator generates both local and across-data-center VM migrations, with an increased focus on scheduling and VM migration over WAN by increasing the percentage of WAN migrations.

The proposed models have been implemented using containers that include all necessary services such as resource scheduling, allocation management, service management, network management, and migration management. To emulate LAN and WAN network connections between data centers, we utilized the open source Wide Area Network emulator WANem [108]. This powerful tool can simulate various WAN conditions including network delay, packet loss, corruption, disconnections, re-ordering, and jitter for data/voice traffic. Additionally, it can act as a transparent application gateway and is licensed under the GPL v2 license.

Figure 4.7 depicts the simulation setup for the blockchain-based distributed model. Blockchain technology was utilized through the BigchainDB platform. We employed containers to run BigchainDB container images as Blockchain nodes in the distributed system. Each container running a data center simulator is connected to a BigchainDB node that runs as a container image. The connections between each data center and a BigchainDB container are established using container bridging through the WANem container.

Extensive testing was conducted on the proposed distributed model using various combinations of data centers, hosts, and VMs as described in Table 4.1. The experiment examined four different

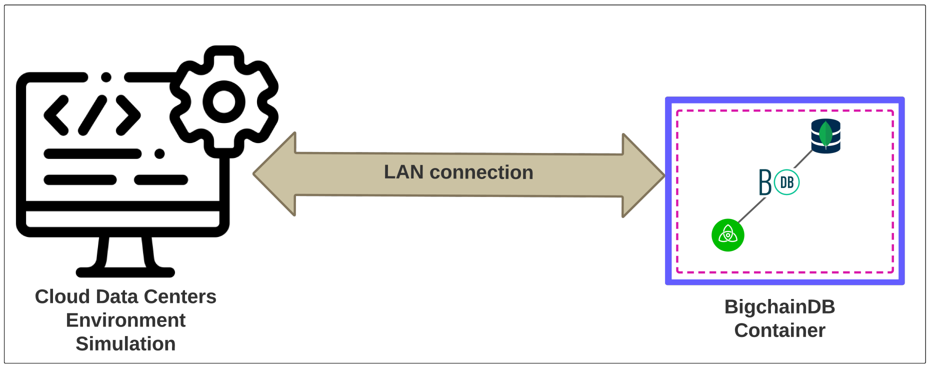


Figure 4.7: Blockchain-based distributed model simulation setup.

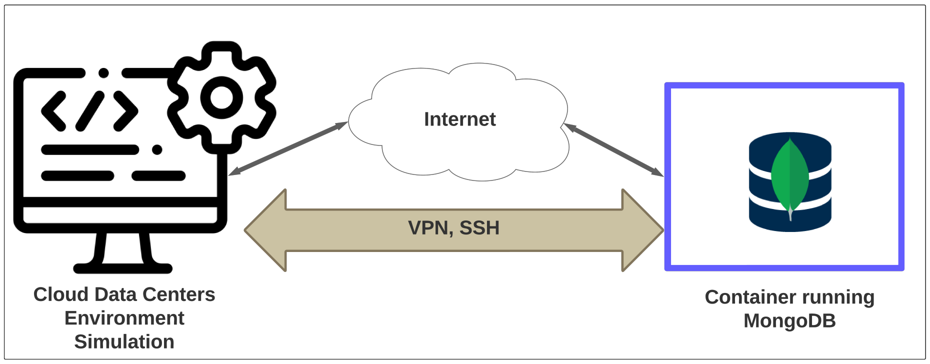
Table 4.1: Simulation scenarios.

Number of data centers	10, 20, 50, 100
Number of hosts	100, 1000, 5000, 10000
Number of VMs	1000, 10000, 30000, 40000

scenarios with 10, 20, 50, and 100 data centers to assess large-scale cloud environments. Each data center consisted of: 10 data centers with 100 hosts and 1000 VMs, 20 data centers with 1000 hosts and 10000 VMs, 50 data centers with 5000 hosts and 30000 VMs, and 100 data centers with 10000 hosts and 40000 VMs. In all scenarios, 10% of the total number of VMs were targeted for migration, with approximately 90% involving WAN migrations.

A container running a MongoDB database is utilized to emulate a centralized global manager. This database kept track of the states of all virtual machines distributed across various data centers and updated their states when migration occurred. The simulation setup for this centralized model can be seen in Figure 4.8. The data center simulators were connected to the global cloud manager via WAN connections emulated through the WANem container. To create a more realistic experiment, all data exchanged was emulated to be sent through VPN and SSH connections.

The proposed blockchain-based distributed model begins by selecting a virtual machine (VM) to migrate from its host. The scheduler manager collaborates with the allocation manager to explore available resources on local hosts for possible migration. If no resources are found, the allocation manager runs allocation optimization algorithms with the migration manager. Then, a potential destination data center and host are identified for VM migration through WAN. The distributed cloud manager communicates with the cloud manager on the destination data center and initiates the


Internet
VPN, SSH
Cloud Data Centers Environment Simulation
Container running MongoDB
Figure 4.8: VPN-based centralized model simulation setup.

VM migration process by making API calls on the Virtual Machine Manager (VMM) hosted on the destination host via network channels. Lastly, the migration process is updated on the BigchainDB node, and a synchronization process ensures all BigchainDB nodes are updated with the change.

In the VPN-based centralized model, when a selected VM cannot be moved to another host within the same data center, the local cloud manager will initiate a request for WAN migration to the global manager. The global manager will then gather status updates from all data centers, utilize optimization algorithms to identify the ideal destination data center and host, and transmit this information to both the original and destination data centers to initiate the migration process. In the event that the status of the destination data center and host changes during the decision-making process and causes a migration error, the optimization algorithms will be rerun until a suitable destination with adequate resources is located.

4.1.3 Results and Discussion

In this section, we introduce and analyze the results of three different implementations: a blockchain-based distributed model, a REST-based distributed model, and a VPN-based centralized model. Each approach is evaluated in terms of the amount of exchanged messages and the communication delay consumed to perform different numbers of cloud management decisions.

4.1.3.1 Blockchain-Based Versus REST-Based Distributed Models

In the blockchain-based distributed model, the decision to move data across WAN is based on the current status of data centers stored in the local blockchain registry. Once a decision is made, the distributed cloud manager sends a message to the corresponding destination distributed cloud manager to initiate the migration process. A single communication message is sent for each VM migration to determine the appropriate course of action. Where in the REST-based model, several number of messages are exchanged to collect the needed information to make the decision. The data presented in Figure 4.9 illustrates the comparison of exchanged messages required for various numbers of VM migrations across different data center quantities. It is evident that our proposed blockchain-based distributed model outperforms the REST-based distributed model in high-scale environments with a variance of 0.5% to 22%. In low-scale cloud environments with fewer data centers (10 and 20 data centers) with lower numbers of VM migrations, our proposed blockchain-based distributed model outperforms the REST-based distributed model. Conversely, in low-scale cloud environments with fewer data centers and higher numbers of VM migrations, the REST approach demonstrates a reduction in communication message count by 4% to 18%.

The relationship between the number VM migrations across different numbers of data centers and the total communication delay in both models is illustrated in Figure 4.10. As the number of VM migrations increases, the exchange of communication messages also increases, leading to longer processing delays for all VM migrations in both models. The results show that the proposed blockchain-based model consistently outperforms the REST-based model across all scenarios, indicating that the blockchain-based solution enables faster communication compared to the

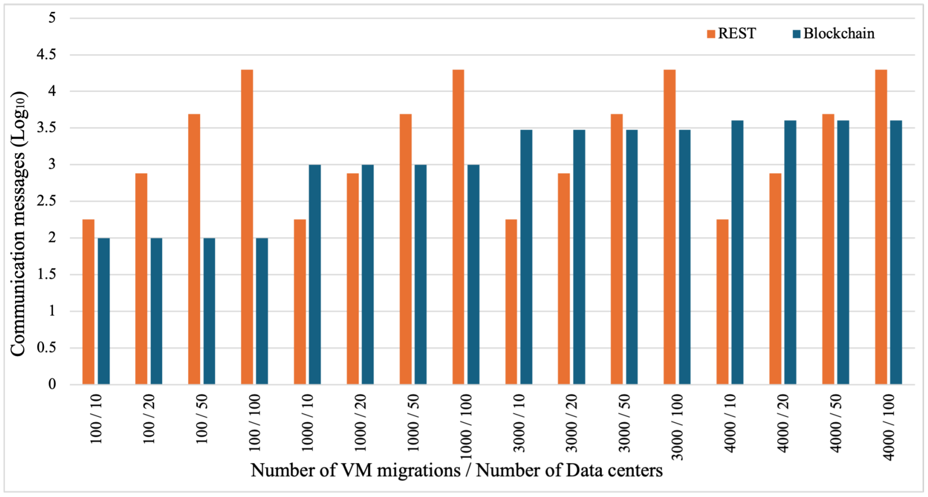


Figure 4.9: Total number of exchanged communication messages.

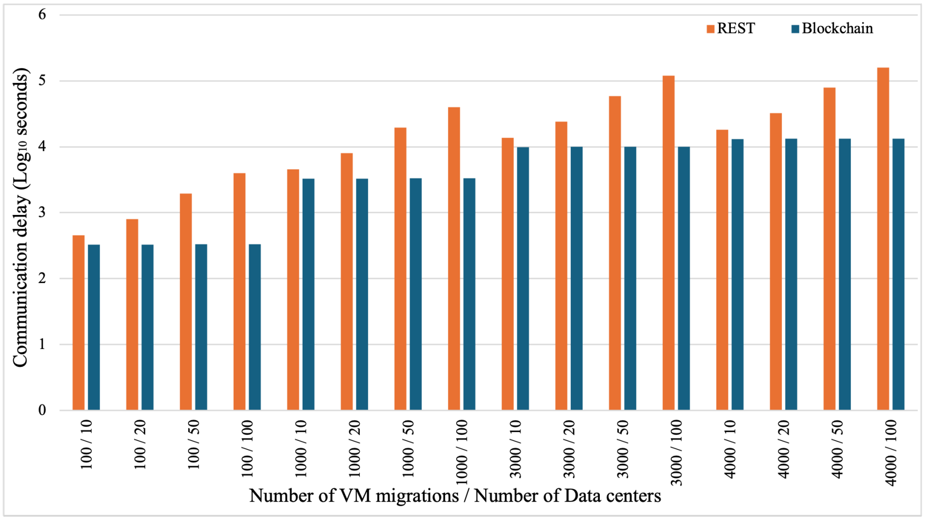


Figure 4.10: Total communication delays.

REST-based model with reduction percentages ranging from 8% to 72%.

4.1.3.2 Blockchain-Based Distributed Model Versus VPN-Based Centralized Model

In the VPN-based centralized management model, the global cloud manager receives WAN migration requests and contacts all data centers to assess their local status. Based on the reports received from each data center, the global cloud manager determines the destination data center and host for the migration. This decision is then communicated to both the source and destination data centers. This communication process is repeated for each WAN migration, resulting in a significant exchange of messages between the data centers and the global manager. Figure 4.11 illustrates the total number of messages exchanged for different scenarios involving VM migrations with varying numbers of data centers and hosts in the VPN-based centralized management model.

In the blockchain-based distributed management model, the decision of a WAN migration is determined by the available data center states in the blockchain registry, locally. Subsequently, the distributed cloud manager sends a message to the destination data center's distributed cloud manager to commence the migration process. For each VM migration, only one communication message is transmitted to make the appropriate decision. The total count of communication messages required for the blockchain-based distributed management model, reflecting the number of VM migrations, is provided in Figure 4.12.

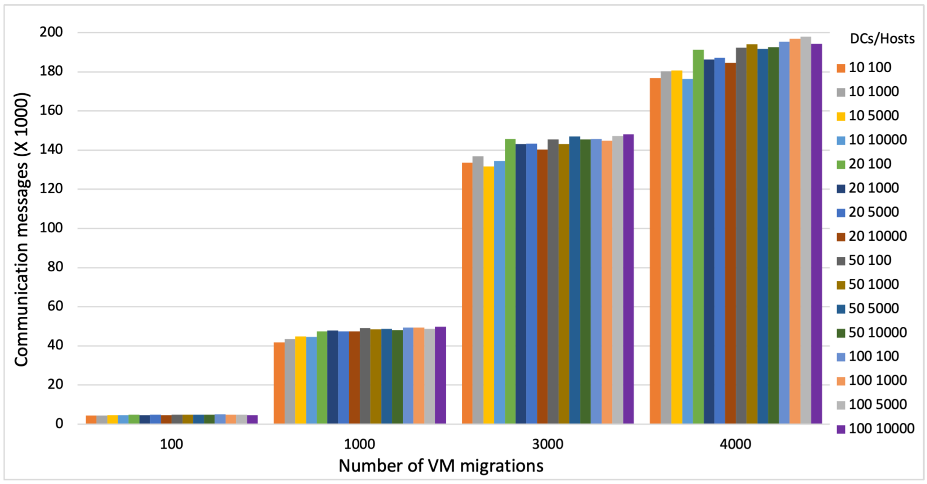


Figure 4.11: Total number of exchanged communication messages in the VPN-based centralized model.

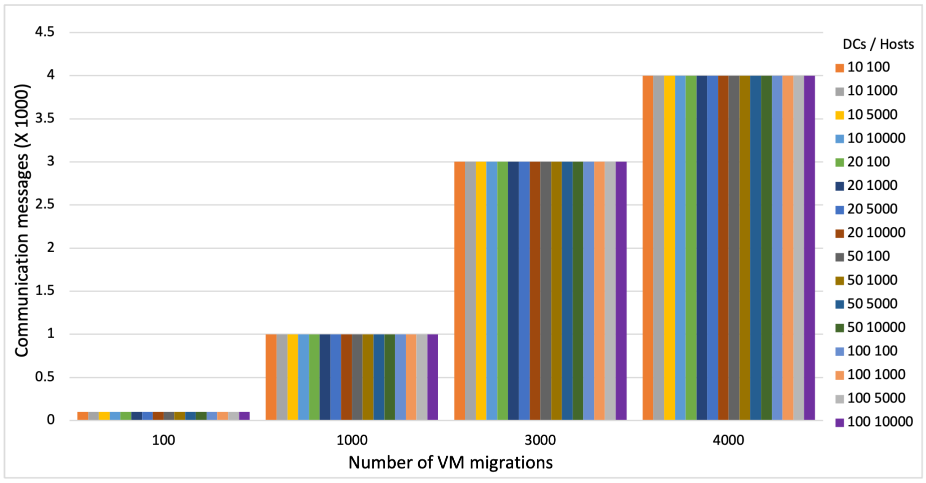


Figure 4.12: Total number of exchanged communication messages in the blockchain-based distributed model.

Figure 4.13 illustrates the cumulative delay experienced during communication message exchanges in the VPN-based centralized model. In order to process WAN VM migrations, the global manager and data centers exchange numerous communication messages. As the number of VM migrations increases, more messages are required to handle these movements, resulting in longer delays to process all transactions. This delay is further compounded by VM misplacements, which cause disruptions in various cases. VM misplacements occur when the global manager makes inaccurate migration decisions, leading to multiple migrations of the VM before it reaches a stable state.

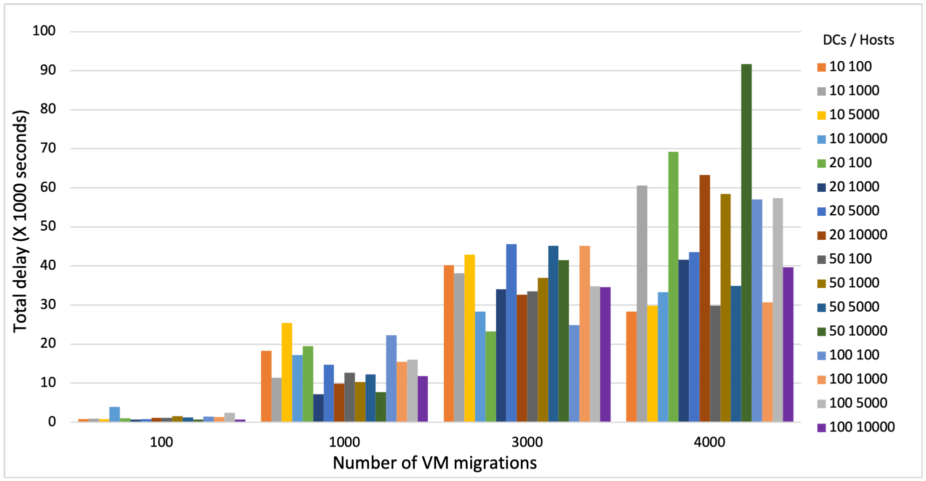


Figure 4.13: Total communication delay in the VPN-based centralized model.

The relationship between the total VM migrations and the total delay of all communication messages exchanged in the blockchain-based distributed model is illustrated in Figure 4.14. This behavior is identical to that of the VPN-based centralized solution, with varying numbers of VMs distributed across different numbers of hosts and data centers. As the number of VM migrations increases, more communication messages are exchanged, resulting in longer processing delays for all VM migrations. However, we observe that the total delay for each case is significantly lower than the VPN-based centralized model. The total delay for each VM migration process includes the migration decision's processing time and the blockchain node's convergence time to synchronize the states of all blockchain nodes distributed across all data centers. Tendermint-based networks, like BigchainDB networks, can include a transaction in a new committed block within just a few seconds or less [31]. Therefore, the total delay for processing a VM migration is less than that in the VPN-based centralized model, where more communication messages are exchanged, resulting in a higher total delay. The absence of VM misplacements in the blockchain-based distributed management model, along with the fixed number of communication messages equal to the number of VM migrations as depicted in Figure 4.12, indicates that there is no disruption in the total communication delay across different cases. This stands in contrast to the VPN-based centralized management model behavior shown in Figure 4.13.

Figure 4.15 presents a side-by-side comparison of the total communication messages exchanged between blockchain-based distributed and VPN-based centralized models. The results indicate that

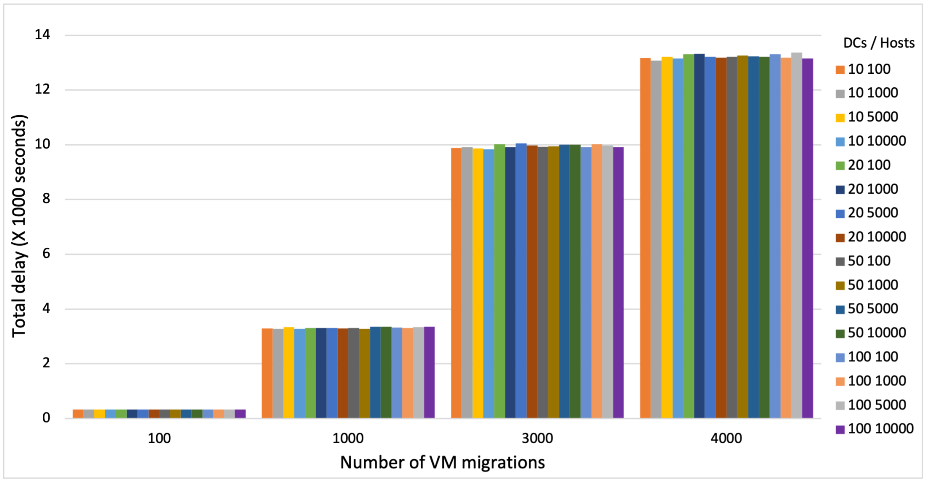
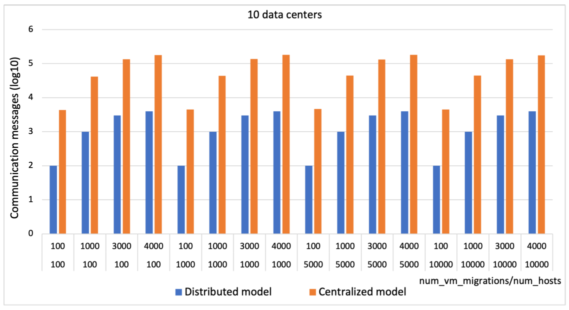


Figure 4.14: Total communication delay in the blockchain-based distributed model.

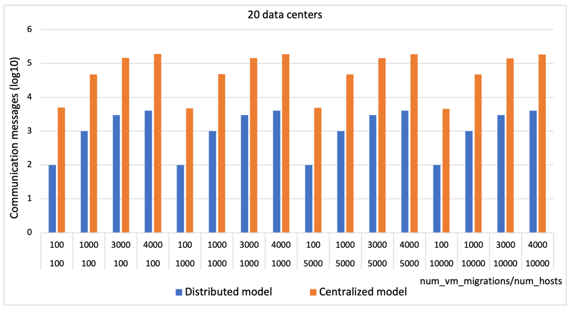
the VPN-based centralized model necessitates a significantly larger number of communication messages to complete all migrations for a given number of VM migrations across various hosts and data centers. In contrast, the blockchain-based distributed model achieves an impressive reduction of 41.79% to 49.85% in exchanged communication messages compared to the VPN-based centralized model. This is due to the centralized global migration manager needing to exchange numerous messages with the data centers to process each VM migration.

In comparing the VPN-based centralized and blockchain-based distributed models of VM migration, it becomes clear that the total delay of the former is significantly greater due to a higher number of communication messages exchanged between data centers and the centralized global migration manager. Further analysis, as shown in Figure 4.16, reveals that the blockchain-based distributed model boasts a significant reduction in total delay, ranging from 2% to 12%, when compared to the VPN-based centralized model. It is worth noting that the extent of this reduction depends on several factors, including the total number of WAN migrations and the number of VM misplacements that occur in the VPN-based centralized management model.

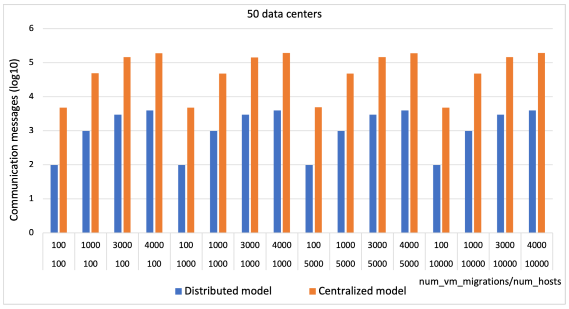
During WAN migration in the VPN-based centralized model, any VM misplacements can cause the migration processes to be repeated, which in turn leads to an increase in communication messages that need to be exchanged. This, in turn, results in additional delays to the total processing time for a specific number of WAN VM migrations. Figure 4.17 illustrates the number of misplacements that arise from various numbers of VM migrations across different data centers and



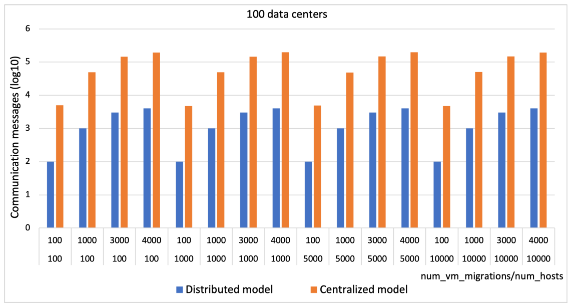
(a) 10 data centers.



(b) 20 data centers.

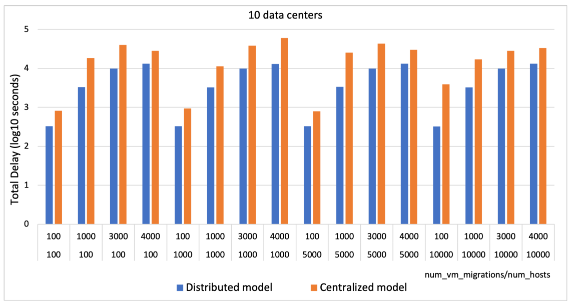


(c) 50 data centers.

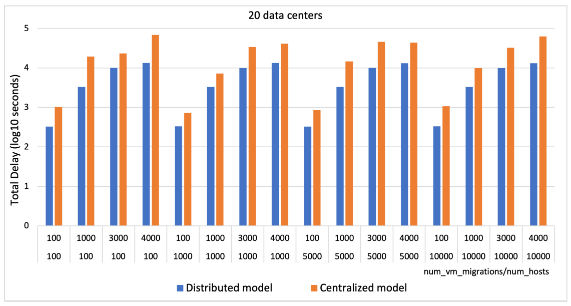


(d) 100 data centers.

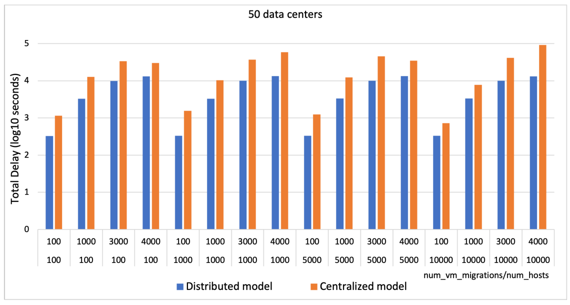
Figure 4.15: Total number of communication messages for different numbers of data centers.



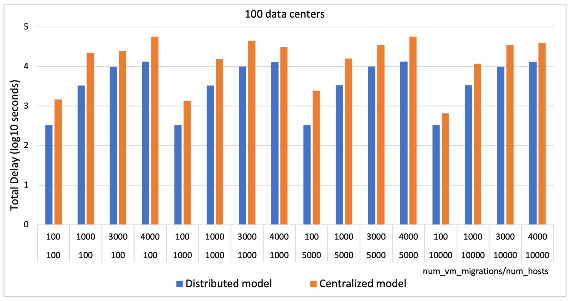
(a) 10 data centers.



(b) 20 data centers.



(c) 50 data centers.



(d) 100 data centers.

Figure 4.16: Total communication delay for different number of data centers.

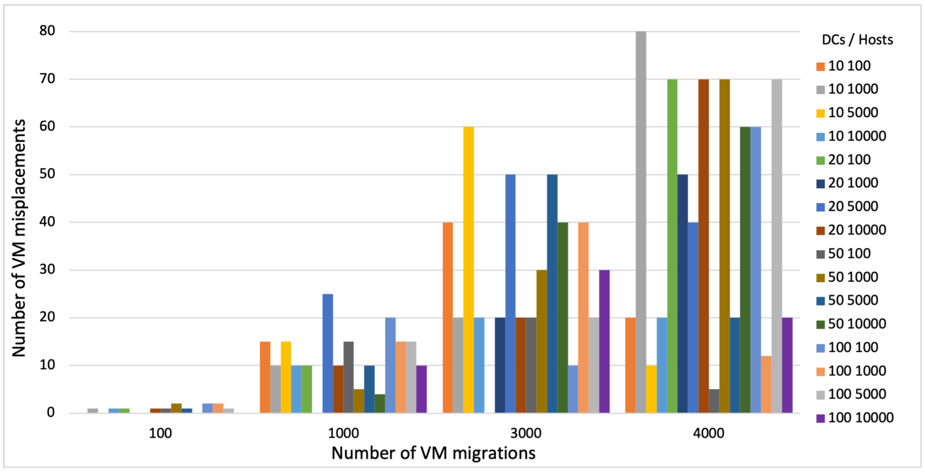
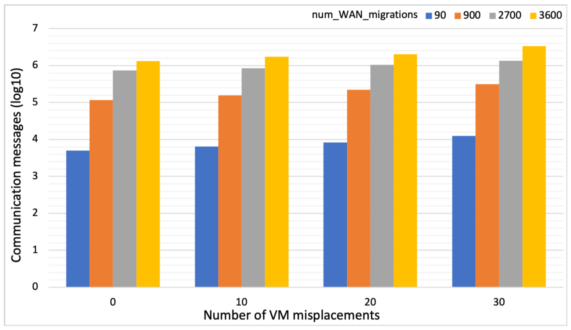


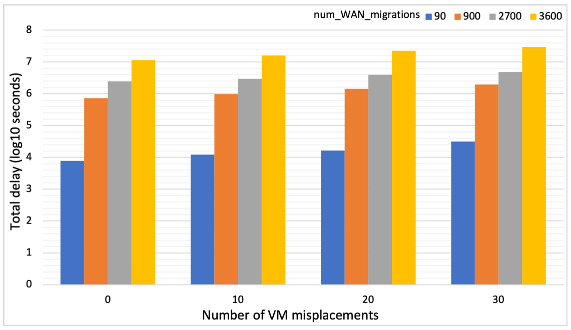
Figure 4.17: VM misplacements for different number of VM migrations in the VPN-based centralized model.

hosts. The occurrence of VM misplacements may be attributed to the inability to access the most up-to-date and comprehensive status of VMs in data centers. This can lead to inaccurate decisions regarding VM migrations, resulting in a series of migrations in order to achieve a stable state. The random nature of these VM misplacements can cause significant fluctuations in the total number of misplacements, as illustrated in Figure 4.17. However, as shown in the figure, the average number of VM misplacements generally rises with an increase in the number of VM migrations, as well as with the number of data centers and hosts. These three metrics are crucial in determining the optimal destination for VM migration. When considering a larger number of VM migrations across multiple data centers and hosts, a balanced decision on the final destination for the migrated VM must be made, which takes into account the impact on scheduling decisions.

Figures 4.18a and 4.18b show the impact of VM misplacements on the total number of communication messages and the total communication delay to process all VM migrations in the VPN-based centralized model. The number of misplacements has a direct impact on the number communication messages, as well as the overall delay, for a specific number of VM migrations, regardless of the number of data centers and hosts involved. In each scenario of VM migrations, the total communication delay and number of messages both increase proportionally with the number of VM misplacements.



(a) Total number of communication messages.



(b) Total communication delay.

Figure 4.18: VM misplacements impact in the VPN-based centralized model.

4.1.4 Results Validation

Through the use of statistical and random variable analysis, we have examined two models based on the central limit theorem. This theorem allows us to describe the behavior of the two systems by calculating the Z-score of the normal random variable distribution, which can accurately predict and match the characteristics of both the centralized and distributed models.

Our analysis measured the probability of the Z-score for various factors, including the number of VMs migrated, communication message exchanges, and delay. To determine the total delay and number of exchanged messages for each simulation case, we conducted an experiment for each scenario. As per the theorem, if a random sample of size n is taken from a population with a mean of μ and a finite variance of σ^2, and X represents the sample mean, then the distribution in Equation 22 approaches the standard normal distribution as n approaches infinity [109].

$$Z = \frac{X - \mu}{\frac{\sigma}{\sqrt{n}}}. \tag{22}$$

If we have two independent populations with means μ_1 and μ_2 and variances σ_1^2 and σ_2^2, and if X_1 and X_2 are the sample means of two independent random samples of sizes n_1 and n_2 from these populations, then the sampling distribution of the Z-statistic is given by Equation 23:

$$Z = \frac{\bar{X}_1 - \bar{X}_2 - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}. \tag{23}$$

In statistical hypothesis testing, the Z-statistic, denoted by Z, is a fundamental tool for assessing the significance of observed differences between sample means and population means. The validity of this approach hinges on the application of the Central Limit Theorem (CLT), which provides a framework for approximating the distribution of sample means as normal, given certain conditions. Specifically, when the underlying populations are normal, the sampling distribution of the Z-statistic is known to follow a standard normal distribution, with all the associated properties of mean and variance. This property underpins the use of Z-tests in a variety of statistical applications, including quality control, experimental design, and survey analysis.

We utilized statistical analysis to validate the system by employing a histogram that was mapped to KDE. From the output probability density function (PDF), we were able to determine the CDF of the distribution. To estimate the probability density function of a continuous random variable based on a sample, we utilized the Kernel Density Estimation (KDE) formula. This formula utilizes kernels (window functions) to smooth the data and approximate the distribution underlying the sample. Assuming $X_1, X_2, \ldots, X_n$ are the data points observed from a continuous random variable X, we used Equation 24 to estimate the probability density function $f(x)$ using KDE.

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right), \tag{24}$$

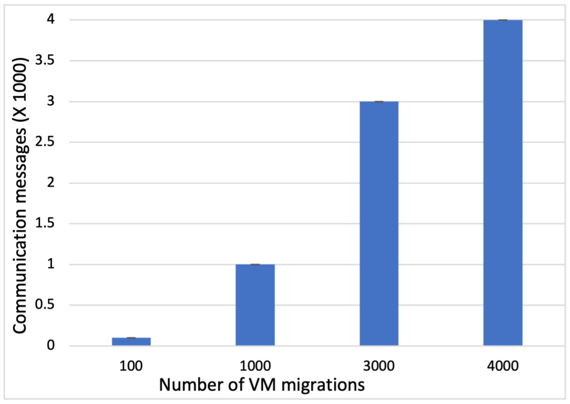
where: $\hat{f}(x)$ is the estimated probability density function at point x, $K(u)$ is the kernel function, h is the bandwidth, and n is the number of data points in the sample.

The probability density function, represented by the symbol $f(x)$, is an essential tool for describing the probability distribution of a continuous random variable X. In situations where a specific range is more likely to contain a value for X, the probability is higher, and in turn, $f(x)$ takes on larger values. To calculate the probability that X will fall within a particular range, such as between a and b, we can integrate $f(x)$ from a to b. The Probability Density Function (PDF) for a continuous random variable X is denoted by $f(x)$ and is defined in Equation 25.

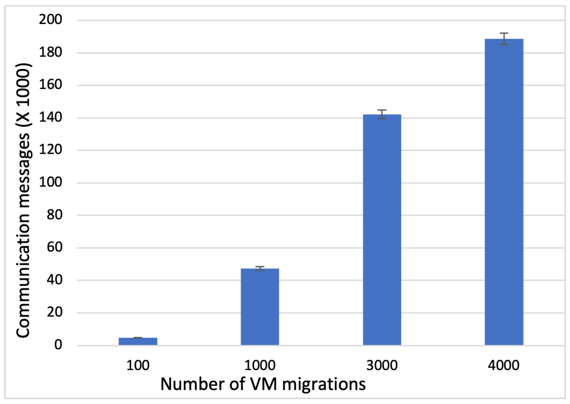
$$P(a \leq X \leq b) = \int_a^b f(x)\, dx, \tag{25}$$

where a and b are any two real numbers such that $a \leq b$.

The confidence interval for the two model measurements' population sampling validation was



(a) Blockchain-based distributed model.



(b) VPN-based centralized model.

Figure 4.19: Total number of exchanged communication messages with confidence interval.

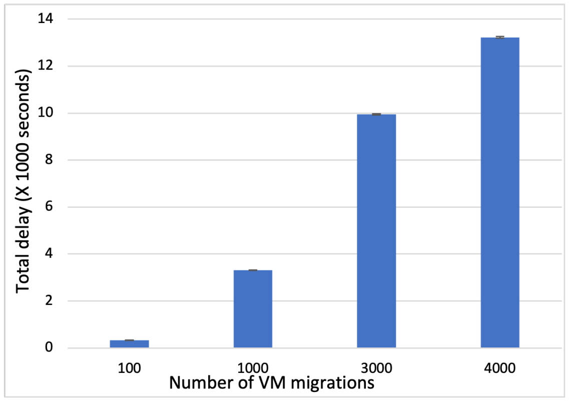
determined and the interval for the population mean was calculated using Equation 26, assuming knowledge of the population variance. These findings contribute to a more accurate and robust understanding of the underlying population parameters and can aid in improving the reliability of research outcomes.

$$\text{Confidence Interval} = \left(\bar{X} - Z\frac{\sigma}{\sqrt{n}}, \bar{X} + Z\frac{\sigma}{\sqrt{n}}\right), \tag{26}$$

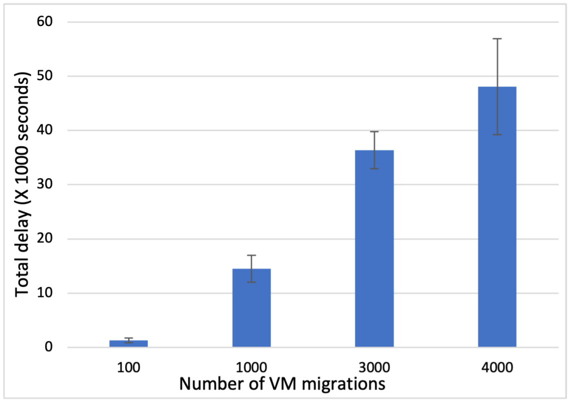
where $\bar{X}$ is the sample mean, σ is the population standard deviation, n is the sample size, and Z is the critical value from the Z-distribution corresponding to the desired confidence level.

Figure 4.19a displays the mean of the total number of communication messages in the distributed model, along with its confidence interval. The sampled data is an exact match with the population, resulting in a 100% confidence level for the system. The mean of the total number of communication messages in the centralized model is shown in Figure 4.19b, with a sampling error falling within the acceptable range of 0.018 to 0.024.

Figure 4.20a displays the mean total delay of communication messages along with confidence intervals in the distributed model, which has an error range of 0.002 to 0.004. Similarly, Figure 4.20b shows the mean total delay of communication messages with confidence intervals in the centralized model, which has an error range between 0.09 to 0.30. The results clearly indicate that the error range in the distributed model is narrower than the centralized model, making it more precise and less uncertain.

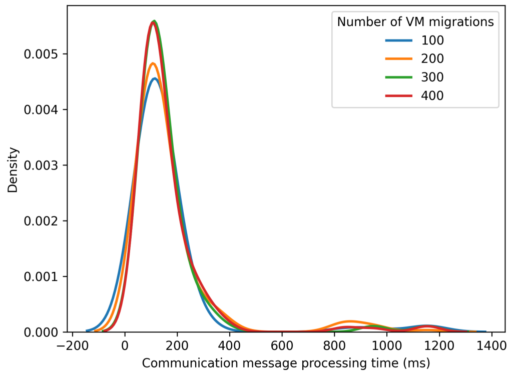


(a) Blockchain-based distributed model.

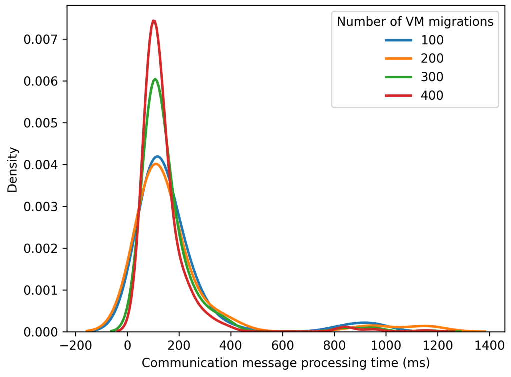


(b) VPN-based centralized model.

Figure 4.20: Total delay of exchanged communication messages with confidence interval.



(a) Blockchain-based distributed model.



(b) VPN-based centralized model.

Figure 4.21: KDE density diagram for different number of VM migrations.

We gathered data on the processing time of every communication message exchanged in both models and used it to calculate the KDE of the observations for four different cases of VM migrations (100, 200, 300, and 400 migrations). As depicted in Figure 4.21a, the KDE density plot for each case of VM migration is strikingly similar, with the majority of observations concentrated on the left side of the graph, indicating that they require less processing time.

Similarly, the centralized model yields the same result. Figure 4.21b displays the KDE density plot for the same four cases of VM migrations, with all cases exhibiting highly comparable processing delay distributions, again skewed towards the left side of the graph, indicating the need for less processing time.

4.1.5 Discussion and Limitations

Our innovative solution provides a highly efficient and simplified approach that effectively minimizes management complexities and inefficiencies. Blockchain technology is utilized to facilitate seamless data centers information sharing, minimizing communication messages, and thereby reducing processing delays. Moreover, our VM migration service employs advanced machine learning algorithms that demonstrate both swift training times and exceptional predictive accuracy, surpassing other approaches documented in the literature.

Our experimental results demonstrate the benefits of employing a blockchain-based distributed management model for facilitating data sharing and communication across multiple data centers in diverse cloud environments. The use of the blockchain-based model surpasses other industry technologies, particularly in large-scale cloud environments, in terms of the number of communication messages exchanged and the time delays incurred for making various management decisions (such as VM migrations in our case). Moreover, the inherent security feature of transmitted messages in blockchain, already integrated into the model, mitigates the need for additional security layers in other management models, thereby safeguarding the privacy of shared data.

The limitations of our proposed approach become apparent when implementing the solution in low-scale environments where a larger number of messages are transmitted to make the necessary management decisions. Additionally, there is no customization of the blockchain model to improve its performance; it is applied as a single unit with no control over its low-level functionality.

4.2 Neural Network Based Regression Model for VM Migration Method Selection

In this section, we propose utilizing ANN to construct a regression model capable of predicting the critical performance metrics of VM migration for both the Pre-copy and Post-copy methods, as well as various application workload types running on the VMs [110]. Based on the prediction, one of the two migration methods can be selected to migrate a specific virtual machine. We use Neural Networks from the TensorFlow platform to build sequential learning models for the migration performance of the two classical methods: Pre-copy and Post-copy. We use the 15-input CSAP dataset to build learning models for the four migration performance metrics: migration time, downtime, the amount of data transferred, and the degradation in the performance of the VM.

4.2.1 Dataset for Machine Learning

We utilized the CSAP VM Live Migration Dataset [62] to train and evaluate our machine learning models for the VM migration prediction. CSAP Virtual Machine Live Migration Dataset provides over 40,000 samples of VMs live migration log collected over a period of several months in the CSAP lab's cluster. The cluster is constructed by four identical servers with four cores processor Skylake i5-6600 and 16 GB of RAM. For the migration traffic, 1 gigabit dedicated NIC is used. For the shared storage, Solid State Disks (SSDs) are used and connected by Network File System (NFS) protocol.

37 unique applications and benchmark workloads, covering a wide parameter space of cloud workloads, were run to explore the characteristics and performance metrics of various live migration methods. The workload types include database, webserver, multicore applications, Java applications, large data compression application, online streaming simulation applications, and many others [61]. The dataset includes 15 features for each migration process, elaborated in Table 4.2. We conducted a comprehensive examination of the 15 input features to evaluate their individual impact on training the regression models, which indicated that all features are mutually independent, prompting us to include all of them in training the regression models.

Five migration methods were used in QEMU [111, 112] to generate the dataset. The generated migration performance metrics are the six metrics: 1) the total migration time measured in QEMU, 2) the downtime measured also in QEMU, 3) the total transferred data, which is a result in QEMU's migration information, 4) the performance degradation of the VM measured by number of retired instructions per second (IPS), 5) the overhead CPU consumption of the host, and 6) the extra memory consumption of the host.

Table 4.2: Dataset Features.

Feature	Description
Workload type	Application type (integer from 0 to 9)
VM size	Virtual memory size of VM
Page dirtying rate	Number of pages modified in a second
Memory working set size	Number of modified pages every 20 seconds
Memory working set entropy	Entropy (compressibility) of memory working set
Memory nonworking set entropy	Entropy of memory non-working set
Dirty words per page	Number of dirty words per page every second
Instructions per second	Retired instructions executed (performance)
Page transfer rate	Network bandwidth reserved for migration traffic
VM's CPU utilization	Average CPU utilization of VM in 20 seconds
VM's Network utilization	Average utilization of VM network
Host's CPU utilization[a]	Hosts CPU utilization
Host's Memory utilization[a]	Hosts Memory utilization

a: Two features (for the source and destination hosts).

4.2.2 Evaluation and Discussion

Mathematical models for the two classical migration methods and a hybrid technique of both were presented in [16, 17]. Comparisons are also presented that show the performance of different migration methods when migrating a VM with specific parameters. Different migration techniques have different performance when migrating a VM. The mathematical models proposed are good in simulation environments to study different cloud system's metrics, but they do not include all migration environment parameters and they lack SLA requirements which are important in the management of real cloud systems.

Machine learning is a good choice to build accurate prediction models that are trained using datasets of migration samples data. The trained models can then be used to predict the migration performance metrics and then classify the predicted metrics to choose the best migration technique to migrate a certain VM. A machine learning model is proposed to predict the key performance metrics of live VM migration for different migration methods and application workloads. In order to provide more accurate prediction models for different migration metrics, we use neural networks to build prediction models for the four migration performance metrics: the migration time, downtime, the total transferred data, and the performance degradation of the VM. The models in this work were built for two migration methods. In the next section, we expand this to include more migration performance metrics and more migration methods.

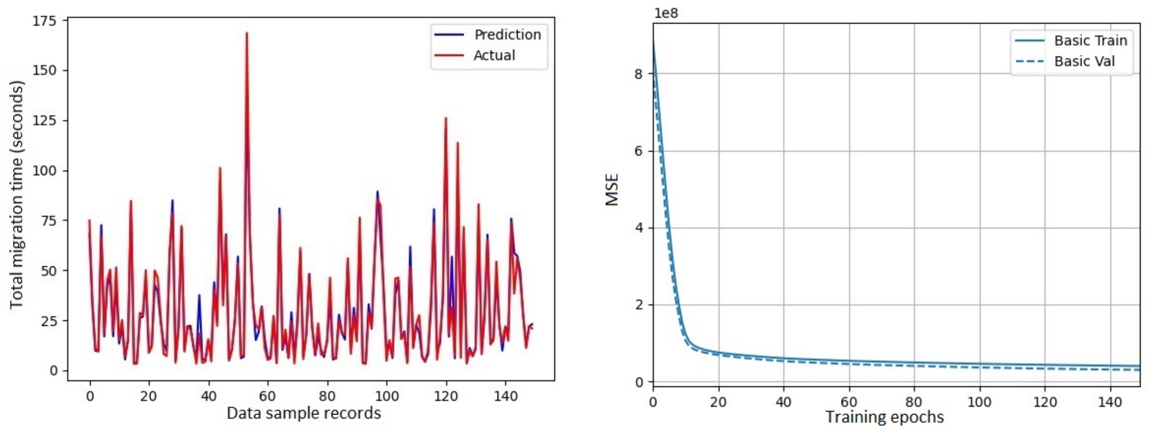
We use TensorFlow platform to build an artificial neural networks regression model for the migration performance of the two classical methods: Pre-copy and Post-copy. We use the 15-input CSAP dataset to train and test the regression model for the four migration performance metrics: the migration time, the downtime, the amount of data transferred, and the degradation in the performance of the VM. The dataset is divided into training and testing sets with fractions of 80% and 20% respectively. In total, using ANN regression model, 8 networks are built for the two migration methods and the four performance metrics, and they all have 2 hidden layers, with densities: 32 and 16 neurons. Table 4.3 summarizes the characteristics of the regression networks.

The accuracy of the proposed ANN model for the four migration performance metrics is shown in different ways. The next figures show the accuracy of each of the four metrics for the two migration methods. For each migration method, we show two figures. The first figure shows the prediction and the actual values of a testing sample of 150 records from the testing dataset. The higher accurate the model is, the better the two lines fitting over each other. The second figure shows the Mean Square Error (MSE) of the basic training and validation of the prediction model over the epochs of learning. The higher the accurate the model is, the quicker the MSE falls down and approaches zero.

Figure 4.22 presents the prediction accuracy of the total migration time for the two migration methods. Figure 4.22 (a) shows a sample of 150 records of the testing dataset with the predicted values of total migration time for the Pre-copy migration method. An excellent prediction is shown from the fitting of both the red and blue lines. The same accuracy is also achieved in Post-copy as

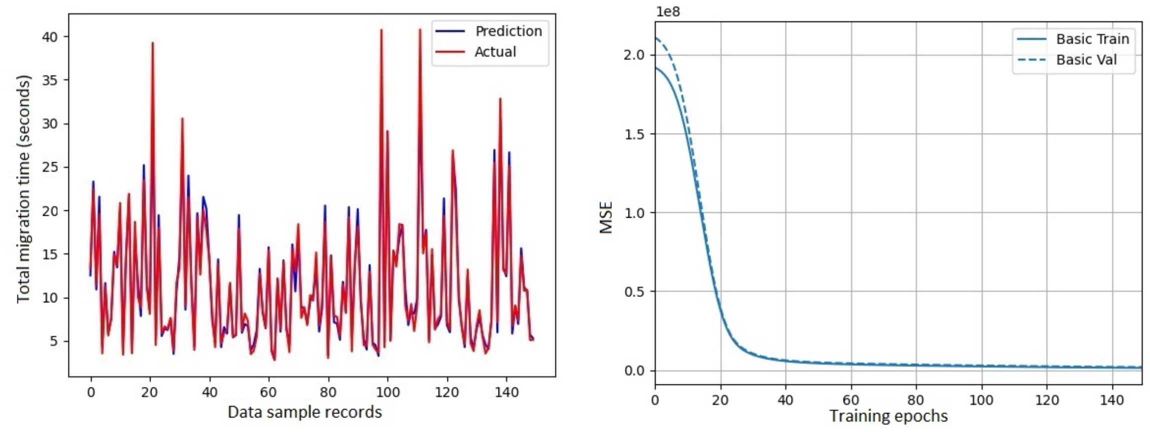
Table 4.3: ANN model networks characteristics.

Migration Method	**Model**	**Input Layer Density**	**Epochs**	**Batch Size**
Pre-copy	Migration time	15	150	5
	Down time		500	25
	Transferred data		150	5
	Performance		500	25
Post-copy	Migration time	15	150	30
	Down time		500	45
	Transferred data		500	30
	Performance		500	30



(a) Prediction accuracy for Pre-copy.

(b) MSE for Pre-copy for 150 epochs.



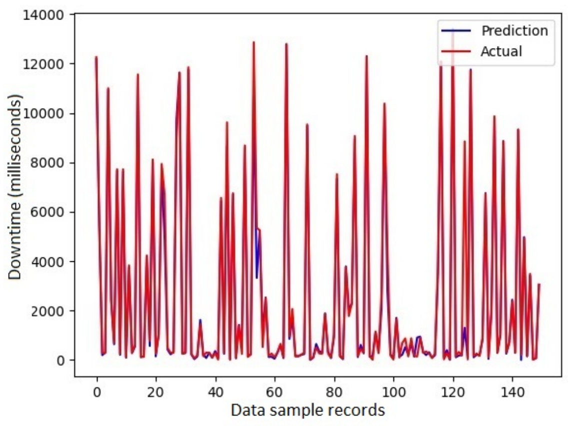
(c) Prediction accuracy for Post-copy.

(d) MSE for Post-copy for 150 epochs.

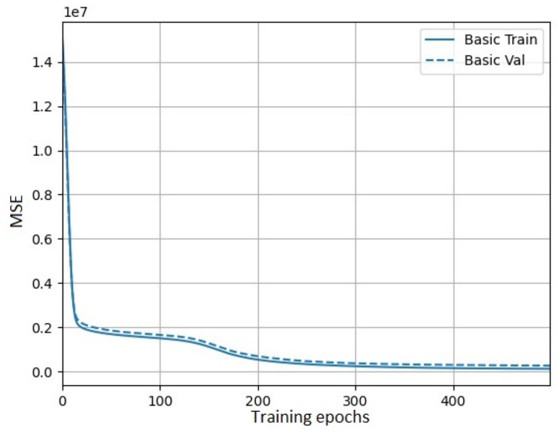
Figure 4.22: Performance of total migration time prediction.

shown in Figure 4.22 (c). The MSE of the prediction for the two migration methods is shown in Figures 4.22 (b) and 4.22 (d). The prediction model for the two methods is trained over 150 epochs and different batch sizes shown in Table 4.3. In both migration methods, MSE falls down quickly towards zero. The MSE of the total migration time of Pre-copy becomes steady on a value that is less than 0.5×10^8, and that is because of not removing the outliers from the training dataset.

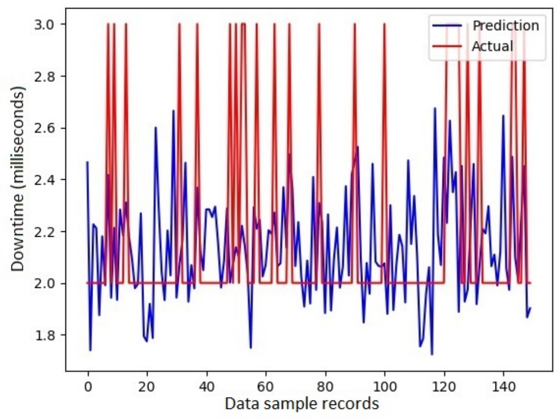
The prediction accuracy of the downtime is shown in Figure 4.23. Figure 4.23 (a) shows a sample of 150 records of the testing dataset with an excellent prediction of migration downtime for the Pre-copy method. On the contrary, as shown in Figure 4.23 (c), the prediction accuracy of the Post-copy is not as accurate as of that in the Pre-copy. The MSE of the prediction model for the



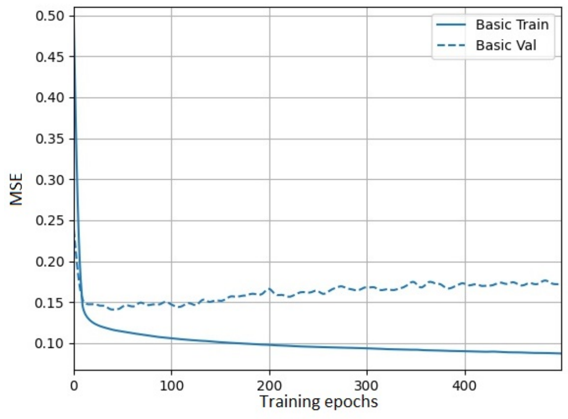
(a) Prediction accuracy for Pre-copy.



(b) MSE for Pre-copy for 500 epochs.



(c) Prediction accuracy for Post-copy.

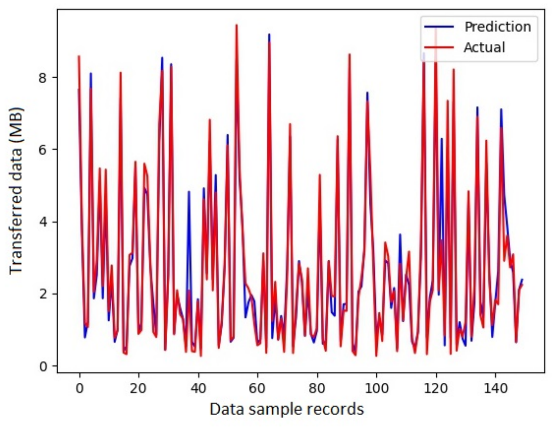


(d) MSE for Post-copy for 500 epochs.

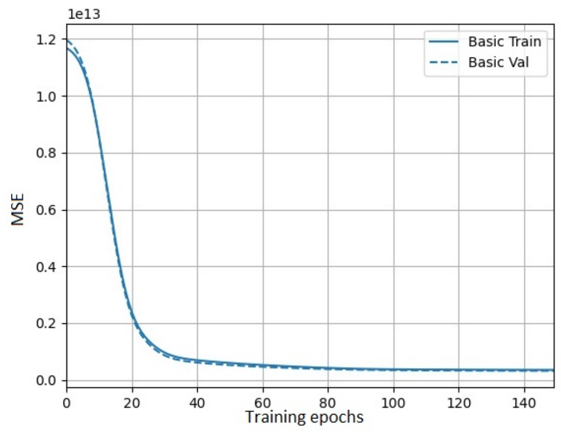
Figure 4.23: Performance of downtime prediction.

two migration methods is shown in Figures 4.23 (b) and 4.23 (d). The models for the two migration methods is trained over 500 epochs and different batch sizes shown in Table 4.3. In both methods, MSE falls down quickly towards zero. The MSE of the downtime of Pre-copy becomes steady on a value that is less than 0.05×10^7. The prediction in the Post-copy model is still valid as the range of the actual values after removing the outliers is from 2 to 3 seconds and the range of predicted values is from 1.8 to 2.7.

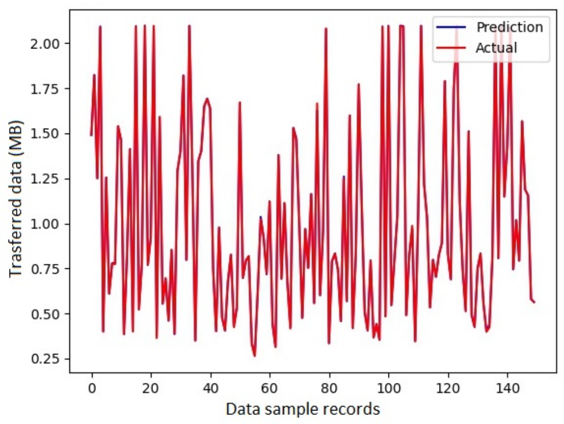
The prediction accuracy of the migration total data transferred for the two migration methods is shown in Figure 4.24. A sample of 150 records of the testing dataset with an excellent prediction for the Pre-copy method is shown in Figure 4.24 (a) and for the Post-copy in Figure 4.24 (c). The



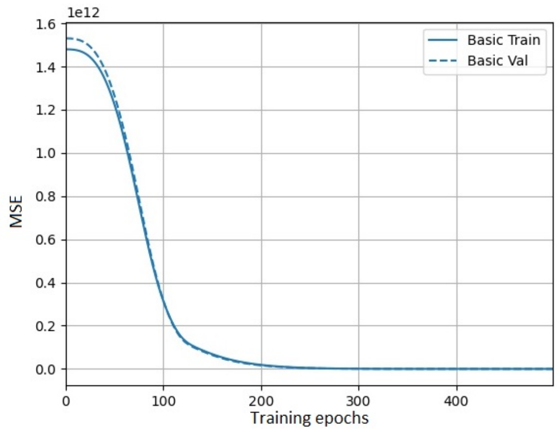
(a) Prediction accuracy for Pre-copy.



(b) MSE for Pre-copy for 150 epochs.



(c) Prediction accuracy for Post-copy.

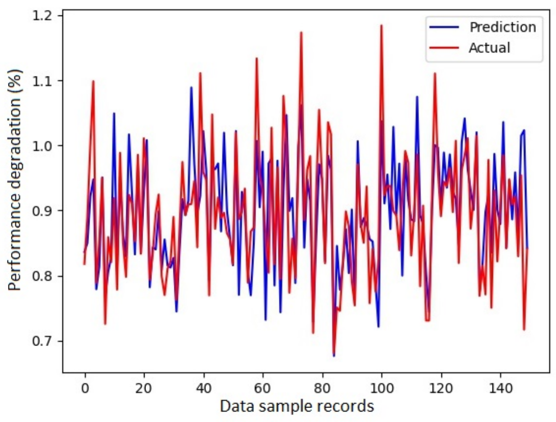


(d) MSE for Post-copy for 500 epochs.

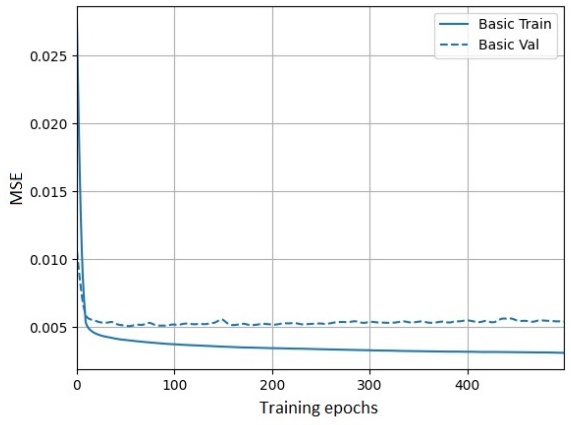
Figure 4.24: Performance of data transferred prediction.

MSE of the prediction model for the two methods is shown in Figures 4.24 (b) and 4.24 (d). For this metric, the prediction model for Pre- copy is trained over 150 epochs and Post-copy is trained over 500 epochs. Different batch sizes are used for the training as shown in Table 4.3. The MSE of the total data transferred of Pre-copy falls down towards zero faster than that of the Post-copy method, and it becomes steady on a value that is less than 0.05×10^{13}, but as the data size used for training is bytes, this MSE value can be considered low when it is converted to megabytes.

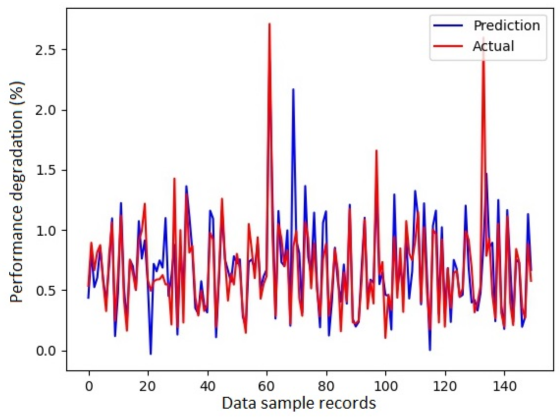
Figure 4.25 shows the accuracy of the VM performance degradation prediction model for the two migration methods. The prediction model for both migration methods is trained over 500 epochs. And different batch sizes shown in Table 4.3 are used. A sample of 150 records of the



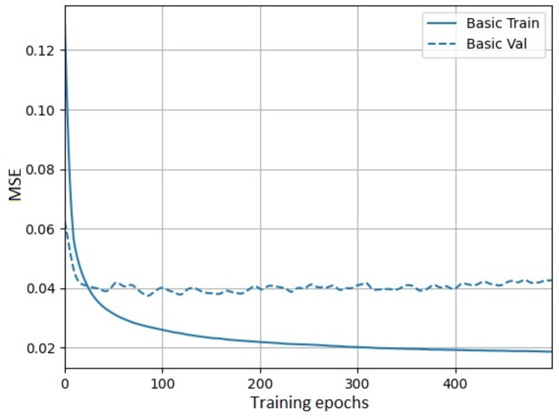
(a) Prediction accuracy for Pre-copy.



(b) MSE for Pre-copy for 500 epochs.



(c) Prediction accuracy for Post-copy.



(d) MSE for Post-copy for 500 epochs.

Figure 4.25: Performance of VM performance prediction.

testing dataset with the prediction values in the Pre-copy method is shown in Figure 4.25 (a) and for the Post-copy in Figure 4.25 (c). The prediction model of this migration metric for both migration methods provides a very good prediction accuracy with very low MSE as shown in Figures 4.25 (b) and 4.25 (d).

Table 4.4 provides a comparison of the Mean Absolute Error (MAE) between our proposed ANN-based regression model and the 3 linear regression models presented in [16] for the four-migration metrics we consider in our model: the total migration time (Mig. Time), Downtime (Down Time), Transferred Data (Data), and the performance degradation (Perf.). Our proposed model of the total migration time, for Pre-copy and Post-copy migration methods, has higher MAE,

Table 4.4: Accuracy of ML models, comparison of MAE for two migration methods and four migration characteristics.

Model	**Pre-copy**				**Post-copy**			
	Mig. Time	**Down Time**	**Data**	**Perf.**	**Mig. Time**	**Down Time**	**Data**	**Perf.**
Proposed ANN Model	3279	114	0.34	0.04	795	0.21	0.002	0.09
Linear Regression [61]	3127	152	59	4.3	1206	1	0.7	14.4
SVR [61]	1197	128	94.4	3.5	289	1	18.5	6.9
SVR with bagging [61]	1053	96	70.7	3	309	1	16.9	6.4

and this is expected as we did not fully remove all outliers in the dataset. The model we propose to predict the downtime, for Pre-copy and Post-copy migration methods, has lower MAE than the linear regression and the SVR models for the Pre-copy method. For the Post-copy migration method, our model has a very low MAE compared to other models. The model predicting the total transferred data and the VM Performance degradation, for both migration methods Pre-copy and Post-copy, has much lower MAE than other models for both Pre-copy and Post-copy migration methods.

A comparison of the processing delay for the training and the prediction of testing dataset records is provided in Table 4.5. Our proposed model has an average training processing delay and a low prediction delay compared to other models.

Table 4.5: Training and prediction overhead of ML models.

Model	**Training (s)**	**Prediction (s)**
Proposed ANN Model	716.3	1.05
Linear Regression [61]	8	0.74
SVR [61]	239	5.11
SVR with bagging [61]	6617.1	188.63

4.3 Machine Learning-Enabled Distributed Model for VM Migration Across Multi-Data Centers of Collaborative Cloud-Edge Environments

The cloud and edge computing is focused on improving the performance of distributed computing systems and making the most efficient use of resources available. The cloud-edge collaboration approach combines edge computing resources like edge servers and IoT devices with cloud resources like data centers and virtualized environments [113]. This allows for optimization of workload distribution and processing based on various factors such as proximity to data sources, latency requirements, and network bandwidth availability.

In cloud computing field, virtualization plays a crucial role in cloud data centers as it enables the operation of multiple VMs, leading to more effective hardware utilization. With cloud resources being diverse and dynamic, scheduling need to be advanced to efficiently allocate tasks while meeting application demands and optimizing resources [114]. Live VM migration is a crucial technique that enables the seamless movement of VM instances between physical servers or data centers without causing any interruption to their operations. Its importance lies in its ability to efficiently manage workloads, maintain load balance, ensure fault tolerance, and scale resources [115].

The power of edge computing lies in its ability to reduce latency by processing data closer to the source or destination [116]. With the help of cloud computing, edge devices can offload intensive computational tasks or data processing to cloud servers, based on real-time demand and resource availability, ensuring speedy and efficient execution [117]. Live VM migration enables this collaboration by allowing smooth transfer of workloads between edge devices and cloud infrastructure in accordance with workload dynamics and performance requirements.

Collaboration between cloud and edge computing, combined with live VM migration, provides the ability to manage computing resources in a scalable and flexible manner. During periods of high demand or resource limitations, edge devices can dynamically scale their computing capacity by utilizing cloud resources [118]. By enabling flexible allocation and reallocation of resources across the cloud-edge continuum, VM migration optimizes resource usage and adapts to changing workload patterns.

When moving a VM to a different data center using WAN networks, it is essential to choose the optimal migration method to minimize downtime and migration time while meeting the SLA requirements [119]. Therefore, various factors need to be considered, and mathematical models have been used to study live migration methods [15, 16, 17]. However, these models may not consider all performance metrics and could overlook SLA requirements. However, machine learning regression models can be incorporated to predict various migration performance metrics and SLA policies that cannot be mathematically modeled. This insight can help determine the most suitable migration technique for any given VM. By forecasting performance metrics, migration time, downtime, and data transfer more accurately, the efficiency and performance of the VM migration process can be significantly improved.

In this section, machine learning algorithms are utilized to enhance VM migration in cloud data centers and edge nodes. The previous work in section 4.2 is extended to include three distinct migration methods, namely pre-copy with CPU throttling, delta compression, and data compression. Moreover, we have incorporated two additional performance metrics, i.e., the extra CPU consumption and additional memory consumption of the source hosts during the migration process. Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Random Forests Regression models are used for even more precise predictions. This results in accurate prediction of VM migration performance metrics, facilitating the selection of the most efficient migration method to reduce downtime and total migration time. It is worth noting that the Hybrid and Dynamic-Hybrid migration methods, which were introduced in Chapter 3, have not been incorporated into this study. The reason behind this omission is that the dataset utilized to develop and train the machine learning models in this study does not provide support for both migration techniques.

A VM migration method selector is developed using the predicted performance metrics and predefined SLA policies and considering factors that traditional mathematical models do not consider. This is particularly important when migrating across data centers or edge nodes over WANs. The prediction service is integrated over the novel blockchain-based distributed cloud management model that is implemented using blockchain technology for secure and authentic information sharing across multiple data centers and edge nodes. This, in turn, has significant implications for enhancing server consolidation and load balancing, ultimately optimizing the allocation of computational resources.

4.3.1 Proposed Model Architecture

When migrating VMs across data centers or edge nodes, traditional management models can sometimes suffer from imprecision and unnecessary VM movement, negatively impacting the overall efficiency and effectiveness of the migration process. To optimize resource utilization and reduce costs, a distributed management model is proposed that utilizes blockchain and machine learning. This model ensures secure and adaptable VM migration across all data centers and edge nodes, with each data center and edge node contributing uniformly. Rather than relying on a global manager, each data center and edge node has its own distributed manager that manages both local and external operations. The blockchain-based distributed model presented in Section 4.1.1 has been modified as a new cloud/edge management model that connects multiple data centers of the cloud layer through WAN, as well as the multiple nodes of the edge layer. Figure 4.26 depicts this model. As each data center possesses a synchronized copy of the states of all other data centers, the WAN migration decision is expected to be more precise, reducing the number of VM migrations and eliminating the migration errors present in other models.

The internal architecture of the distributed cloud/edge manager in a data center or edge node is shown in Figure 4.27. It is similar to the internal architecture of the proposed model of Section 4.1.1 comprising the Blockchain Service modules and the five main modules: Resources Scheduler Manager, Allocation Manager, Service Manager, Network Manager, and VM Migration Manager. In addition, the VM Migration Prediction Service is integrated into the blockchain-based distributed model to cooperate with the VM migration manager.

The VM migration prediction service module is responsible for selecting the best migration method for a VM with specific characteristics. It takes into account the limited bandwidth networks such as WAN and aims to minimize migration delays and dependencies between source and destination hosts. Figure 4.28 illustrates the internal architecture of the cloud manager's VM migration prediction service module. The proposed module comprises two sub-modules, namely the machine learning regression part and the migration method selector part.

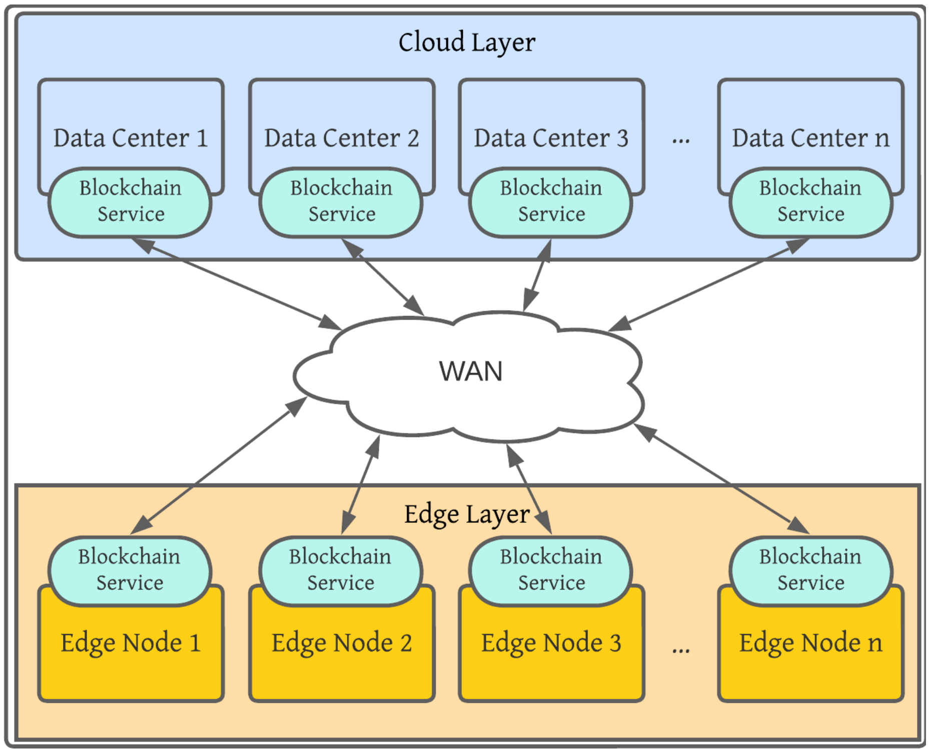
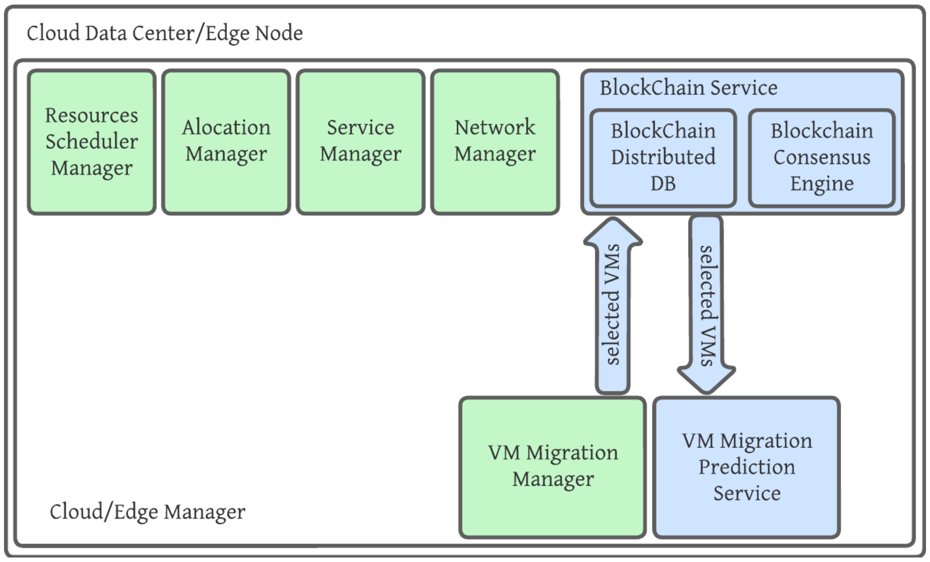


Figure 4.26: Blockchain-based distributed management model.



Cloud Data Center/Edge Node
Resources Scheduler Manager
Alocation Manager
Service Manager
Network Manager
BlockChain Service
BlockChain Distributed DB
Blockchain Consensus Engine
selected VMs
selected VMs
VM Migration Manager
VM Migration Prediction Service
Cloud/Edge Manager

Figure 4.27: Distributed cloud/edge manager model architecture.

The regression sub-module employs regression models to predict VM migration metrics that include total migration time, downtime, total transmitted data, VM performance during migration, additional CPU consumption, and additional memory consumption of the source hosts during the

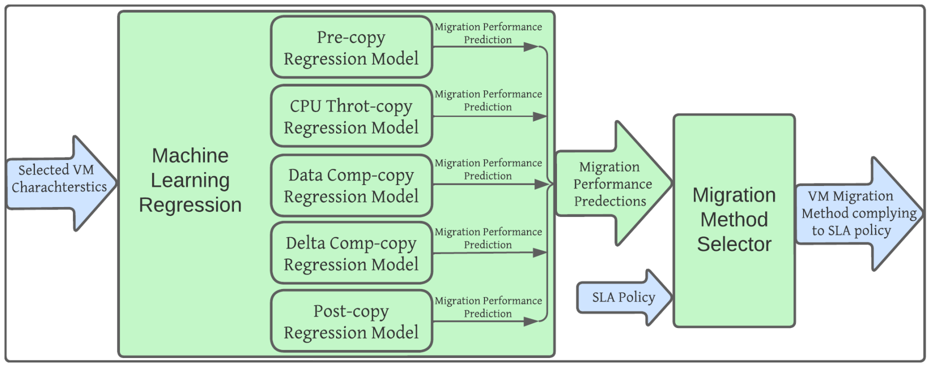

Selected VM Charachterstics
Machine Learning Regression
Pre-copy Regression Model
CPU Throt-copy Regression Model
Data Comp-copy Regression Model
Delta Comp-copy Regression Model
Post-copy Regression Model
Migration Performance Prediction
Migration Performance Predections
SLA Policy
Migration Method Selector
VM Migration Method complying to SLA policy

Figure 4.28: VM migration prediction service module internal architecture.

migration process. These models are built for five migration methods, including classical pre-copy and post-copy methods, and three enhanced pre-copy methods, i.e., pre-copy with CPU throttling, delta compression, and data compression.

Three regression algorithms are used in this study: Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Random Forest Regression. ANN can model complex, non-linear relationships between input and output data by automatically learning hidden features from input data [18]. CNN is ideal for image processing as it recognizes patterns in data and reduces data size by considering the correlation between multivariate variables [19]. Random Forest Regression is an ensemble of decision trees that corrects overfitting and provides better prediction accuracy [20]. These algorithms are widely established and successful in various applications.

The prediction models' output is fed into the second part of the module, which uses a different algorithm to select one of the five migration methods based on specific VM parameters while complying with defined migration SLA policies from the service manager. As part of the proof of concept, the SLA policies have been defined by mixing a range of VM migration goals and constraints. Table 4.6 shows the selected SLA policies. To assess the system's performance, two migration objectives were tested: minimizing the total migration time and minimizing the downtime. Four migration SLA restrictions were examined: ensuring that the total migration time is under 5 seconds, the downtime is under 3 seconds, the total transferred data is less than 500 MB, and that the relative performance during the migration remains above 85%.

The migration method selector selects a migration method by following Algorithm 4 to choose

Table 4.6: Migration SLA policies.

		Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6
Migration objectives	Minimize the total migration time	✓	✓	✓			
	Minimize the total downtime				✓	✓	✓
Migration SLA constraints	Migration time less than 5 seconds				✓		
	Downtime is less than 3 seconds	✓					
	Transferred data less than 500 MB		✓			✓	
	Performance is above 85%			✓			✓

one of five strategies: pre-copy (0), pre-copy with CPU throttling (1), pre-copy with data compression (2), pre-copy with delta compression (3), and post-copy (4). The selection algorithm searches the regression results from the regression part to determine the migration method that best satisfies the chosen SLA policy. It takes as input the characteristics of the VM to be migrated and an SLA policy from Table 4.6. The algorithm outputs a number, ranging from 0 to 4, indicating the selected migration strategy, as well as a violation error in the event that none of the strategies meet the constraints of the input SLA policy. The SLA compliance rate serves as a key performance indicator for evaluating the efficacy of the migration method selector. This metric is utilized to determine whether the selected migration methods align with the predetermined SLA policies.

Algorithm 4 Selecting the best migration strategy.

```
Input: VM characteristics, SLA policy
Output: mig_method, violation_error
  for all migration strategies regression models do
      Predict migration metrics (VM characteristics)
  end for
  for all prediction results do
      $models \leftarrow min(method(SLA\_constraint))$
      $models \leftarrow max(method(SLA\_constraint))$
  end for
  if $length(models) = 0$ then
      $mig\_method \leftarrow min(model(SLA\_objective))$
      $violation\_error \leftarrow mig\_method - SLA\_constraint$
  else
      $mig\_method \leftarrow min(models(SLA\_objective))$
      $violation\_error \leftarrow 0$
  end if
```

4.3.2 Results and Discussion

We utilized the CSAP VM Live Migration Dataset [62] to train and evaluate our Machine Learning (ML) models for the VM migration prediction service module. We used the 15 features for each migration process included in the dataset, elaborated in Table 4.2.

We use regression models that leverage advanced neural networks and regression techniques to achieve unparalleled accuracy. Through extensive experimentation, we optimized the number of layers and estimators to create these models. ANN models consist of three hidden layers with node densities of 46, 23, and 14, while CNN models comprise a 2D convolution layer, a Flatten layer, and three hidden layers with node densities of 64, 64, and 5. The random forest regression models are employed with 65 estimators to ensure precise predictions. We used the ReLU activation function and the RMSProp optimization algorithm to train the ANN models. CNN models utilized ReLU activation and the Adam optimizer. Finally, the random forest regression models employed the MSE criterion for splitting and the Gini index for impurity measurement.

To validate our proposed models, we employed the similar combination of real-world implementation and simulation techniques as that employed in the work of Section 4.1. We have constructed several containers to run data center simulators, with varying numbers of hosts and VMs. Our simulator generates both local and across-data-center VM migrations, with an increased focus on VM migration over WAN. To implement our proposed distributed mode, we have included all necessary services such as resource scheduling, allocation management, service management, network management, and migration management within containers. To emulate network connections between data centers, we have utilized the open source Wide Area Network emulator WANem [108].

To evaluate the performance of the proposed VM migration prediction service for cloud management, we calculated key common regression performance metrics including the mean absolute error (MAE), the mean squared error (MSE), the mean absolute percentage error (MAPE), the root mean squared error (RMSE), and the R-squared (R^2) score. Our prediction performance results are shown in Table 4.7 for the three models we developed: artificial neural networks (ANN), convolutional neural networks (CNN), and random forest regression (RFR). We also compare our models to other models presented in the literature like multiple linear regression (MLR), support vector regression (SVR), bagging support vector regression (BSVR), K-nearest neighbors (KNN), and ridge regression (RR).

The Mean Absolute Error (MAE), as defined in Equation 27, serves as a popular metric to measure the precision of regression models. It computes the average absolute difference between predicted and actual values, treating all errors equally. The lower the MAE, the higher the model's performance. Table 4.7 demonstrates that the suggested models outperform others in terms of lower MAE values, with the Random Forest Regression having the least MAE value.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |predicted_i - actual_i|. \tag{27}$$

The Mean Squared Error (MSE) is a widely used metric for assessing the accuracy of regression models. It calculates the average squared difference between predicted and actual values, with greater emphasis on larger errors, making it responsive to outliers. Equation 28 defines MSE, and Table 4.7 illustrates that the suggested regression models excel against others in the literature in terms of MSE. The Random Forest Regression Model is the most precise predictor.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (predicted_i - actual_i)^2. \tag{28}$$

The Mean Absolute Percentage Error (MAPE) is a frequently utilized loss function for regression problems and model evaluation due to its intuitive interpretation in relative error. Equation 29 defines MAPE, while Table 4.7 showcases that our proposed regression models excel compared to

Table 4.7: Machine learning regression models performance comparison.

Model	MAE	MSE	MAPE	RMSE	R^2
MLR	0.4646	0.7774	8.1486	0.8817	0.2850
SVR	0.3074	0.6507	6.4888	0.8066	0.4087
BSVR	0.2943	0.5926	6.1564	0.8040	0.5634
KNN	0.3461	0.6069	11.2842	0.7790	0.4499
RR	0.4646	0.7774	8.1484	0.8817	0.2850
ANN	0.1887	0.2744	3.2121	0.5238	0.7755
CNN	0.1586	0.2477	2.4450	0.4977	0.7992
RFR	0.1256	0.2343	3.3496	0.4841	0.8112

other models in the literature in terms of MSE. Based on this evaluation metric, the CNN model yields the most accurate prediction.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{predicted_i - actual_i}{actual_i}|. \tag{29}$$

The Root Mean Squared Error (RMSE) is a metric that measures the average variance between predicted values from a model and their corresponding actual values. It offers insight into the model's ability to accurately predict target values. Equation 30 defines the RMSE. In Table 4.7, it is evident that the proposed regression models outshine their literary counterparts in terms of MSE. Notably, the random forest regression model achieves the highest level of prediction accuracy.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{n}}. \tag{30}$$

The R-squared (R^2) score, also known as the coefficient of determination, is a statistical metric used to assess the accuracy of a regression model. It measures the extent to which the independent variables included in the model can predict the variance in the dependent variable. In other words, it determines how well the regression model accounts for the fluctuations in the data. The value of the R^2 score ranges from 0 to 1, as shown in Equation 31. When the score is 0, the model is unable to explain any changes in the dependent variable, and its predictions are equivalent to the average of the dependent variable. Conversely, a score of 1 indicates that the model can perfectly explain all the changes in the dependent variable, and its predictions match the actual data points exactly. If the R^2 score is negative, it means that the model performs worse than the horizontal line of the mean of the dependent variable, which can occur if the model is overfitting.

The R^2 scores attained by the proposed models have been compared to those of other models in Table 4.7. The proposed models have exhibited superior results. Notably, the random forest regression model has achieved the highest score, indicating its suitability for accurately predicting the dependent variables of the model.

$$R^2 = 1 - \frac{SSR}{SST}, \tag{31}$$

where SSR is the sum of squared residuals (differences between predicted and actual values), and SST is the total sum of squares (total variance of the dependent variable).

The average duration for both training and testing of our regression models is presented in Table 4.8. Training duration refers to the time taken to train the model, while testing duration represents the time required to predict the entire testing dataset. All durations are measured in seconds.

Our models have significantly longer training times than linear regression algorithms but shorter times than non-linear algorithms. This is due to the complexity of neural networks and the ensemble of decision trees in the random forest model.

When it comes to testing time, our models perform acceptably compared to other regression models. Predicting the entire testing dataset, which includes thousands of records, is a time-consuming process. However, our models provide reliable predictions within reasonable time-frames.

The VM migration prediction service includes a migration method selector module that determines the most appropriate migration method from five options based on a set of pre-defined SLA policies. Figure 4.29 displays the module's SLA compliance rate, which was calculated for a sample of 1000 VM instances across three proposed regression models. The compliance rate was determined by dividing the number of instances meeting the SLA by the total number of instances, resulting in a range of 18% to 94.9% depending on the specific SLA policy implemented.

Figure 4.30 shows the total migration time of 1000 VM migration processes. The blue bars represent the total migration time when using a fixed migration method for all migrations of the five migration methods: Pre-copy, Pre-copy with CPU throttling, Pre-copy with data compression,

Table 4.8: Average training and testing times for regression models.

Regression Model	Training time (s)	Testing time (s)
MLR	0.0366	0.002
SVR	476.02	99.47
BSVR	30464	3960
KNN	0.008	0.7
RR	0.018	0.002
ANN	233.312	0.48
CNN	164.767	0.5
RFR	46.3	0.197

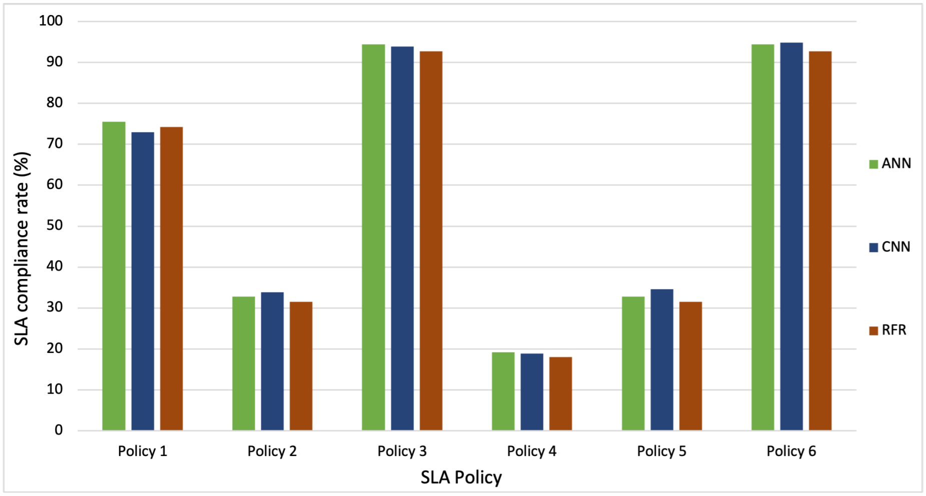


Figure 4.29: SLA compliance rate for the three regression models and different SLA policies.

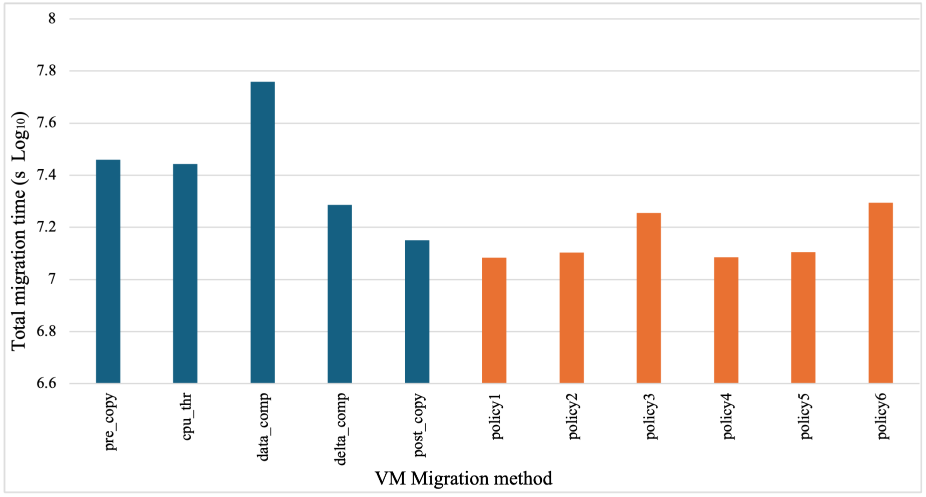


Figure 4.30: Total migration times for fixed migration methods and the proposed selector method.

Pre-copy with delta compression, and Post-Copy. The Orange bars represent the total migration times achieved by applying our proposed system to select the best migration method, out of the five methods, that best complies with the SLA policies applied in the our system. For all SLA policies considered, our proposed solution reduces the total migration time comparing to using a fixed migration method with reduction percentages range from 14% to 79%.

Figure 4.31 shows the total downtime for using a fixed migration method, from the five methods, and for our proposed system when applying the six SLA policies. Our proposed solution reduces the total downtime of the the 1000 migrations with reduction percentages range from 64% to 99%.

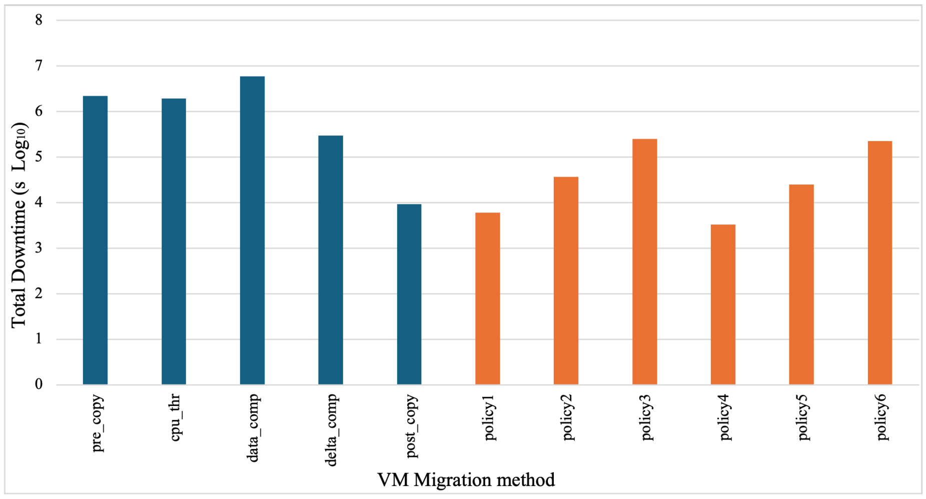


Figure 4.31: Total downtimes for fixed migration methods and the proposed selector method.

4.4 Chapter Summary

In this chapter, a blockchain-based distributed cloud resource management model that employs the BigchainDB framework as a blockchain registry is introduced in section 4.1. This registry allows for secure and efficient information sharing about data centers, facilitating quick and dependable scheduling decisions for various cloud data centers. Our blockchain-based model reduces the total messages exchanged for the VM migration with percentages ranging from 0.5% to 22% and the total communication delay by 8% to 72% compared to another REST-based distributed model. The proposed blockchain-based distributed model has also shown a remarkable decrease of 41.79% to 49.85% in exchanged communication messages, as well as a significant reduction in total delay ranging from 2% to 12%, when compared to the centralized model.

By using machine learning, more migration parameters that affect the migration process can be included in the prediction and more migration performance metrics can be predicted. ANN-based prediction model was initially presented in section 4.2 to predict the four live VM migration metrics: total migration time, downtime, total data transferred, and VM performance degradation. The proposed model predicts the metrics with high accuracy compared to other models and it has an average acceptable training and prediction delays. The proposed model can therefore be used in different cloud data centers for VM's migration performance metrics prediction prior to the migration process, and then the best migration technique can be used for the migration to migrate the VM with the required migration time and downtime and transfer less amounts of data with the least VM performance degradation while complying to the existing SLAs.

Artificial Neural Networks, Convolutional Neural Networks, and Random Forest prediction models have been presented in section 4.3 to predict the six live VM migration metrics: total migration time, downtime, total data transferred, VM performance, host CPU utilization, and host Memory utilization during the migration process. Additionally, we integrated the machine learning models into the blockchain-based distributed management model proposed in section 4.1 as a VM migration service that selects a migration method based on specific VM characteristics and SLA policies to comply with. Furthermore, our machine learning algorithms for the VM migration service demonstrate acceptable training time and SLA compliance rate, and significantly more accurate prediction than other methods presented in the literature. The SLA compliance rate of the proposed solution ranges from 18% to 94.9% for different machine learning algorithms and SLA policies during VM migrations across WAN-connected data centers. The proposed solution reduces the total migration time by 14% to 79% and the downtime by 64% to 99%. These results demonstrate the effectiveness of the proposed model and its potential to improve the efficiency of cloud systems management.

Our innovative solution provides a streamlined and effective approach that reduces management burdens and inefficiencies, outperforming conventional centralized methods. We utilize blockchain technology to facilitate data center information sharing, minimizing the amount of communication messages exchanged and thereby reducing processing delays. Additionally, our VM migration service employs advanced machine learning algorithms that demonstrate both swift training times and exceptional predictive accuracy, surpassing other approaches documented in the literature.

The next chapter introduces a novel approach to optimizing the allocation and scheduling of containers in cloud platforms. Our strategy employs advanced optimization techniques to enable the efficient allocation of containers to computing nodes while ensuring the secure migration of previously allocated containers to new nodes. By achieving superior server consolidation and reducing energy consumption, our proposed solution holds the potential to revolutionize container management in the cloud.

Chapter 5

Optimizing Secure Scheduling of Cloud Containers

Over the past few years, containers have become popular for simplifying application deployment and management. This virtualization technology provides a consistent runtime environment, enabling seamless application mobility across diverse systems. Additionally, containers are lighter than traditional virtual machines, translating to faster initiation and termination and superior resource utilization.

Container technology is a virtualization method that provides a runtime environment for applications and their dependencies at the process level [2]. The container is composed of two primary layers: the application layer and the base image layer. These layers work in tandem to ensure the smooth operation of the application or microservice [120]. The application layer contains the code for the application itself, and dependencies include codes, binaries, system libraries, and any configuration files needed for the application to run. The base image consists of these files and common system files. The container's ability to run relies on the necessary binaries and libraries in the base image layer. A container engine must be installed on the machine for the container to work. Docker, LNX, OpenVZ, and Containerd are some examples of container engines. Containers can function on bare metal servers or virtual machines, and multiple containers with similar dependencies can be utilizing a common base image. The container platform architecture makes it possible to quickly deploy, migrate, terminate, and replicate containers, boosting the adaptability and scalability of computing systems [3].

Security of containers is important for cloud system providers and end users. There are three levels of data protection for containers: securing the saved container base image (data at rest), securing the running container memory (data in-use), and securing the container data while being migrated (data in-transit). In this research, we specifically focus on ensuring the security of container data while being migrated by encrypting it prior to the migration process at the source host, and decrypting it upon its arrival at the destination.

When migrating between hosts within the same data center or across different ones with varying resource capabilities, it's crucial to have an adaptive encryption algorithm and key size. We utilize attribute-based encryption, encompassing Attribute-Based Access Control (ABAC) and Attribute-Based Encryption (ABE). In our proposed approach, a customized ABE method has been effective for our needs, as we rely on the cloud management portal to handle authorization control between cloud data centers. To clarify, we cannot consider a static portal configuration for the container migration data block using one encryption type and key size due to the diversity in data center hardware and running container characteristics with user SLA requirements. This highlights the need for an on-the-fly system to set a key size and encryption method type as needed.

ABAC is an access control approach that authorizes users based on their specific attributes. ABE is an encryption technique that allows data owners to set access control policies for encrypted data. This method ensures that only individuals with the necessary attributes can decrypt the protected data. As a result, ABE is considered a promising access control mechanism [121]. There are two types of ABE: Key Policy Attribute Based Encryption (KP-ABE) and Ciphertext-Policy ABE (CP-ABE) [122]. KP-ABE limits access to encrypted data to a chosen few users with keys. It uses policies in users' private keys. On the other hand, CP-ABE is a type of public-key encryption that defines a user's private key using attributes. The ciphertext is tied to an access policy, and decryption is only possible if the policy and user attributes match. This access control is based on user attributes, which can be issued by attribute authorities in one or multiple domains [123]. CP-ABE schemes are advantageous as they can cover a large number of users. Multi-authority CP-ABE schemes are well-suited for applications like e-health and e-government where users have attributes from multiple domain authorities.

KP-ABE in cloud environments has gained significant attention in recent years due to its potential to granularly secure cloud resources and data. This technology allows administrators and end-users to control cloud systems and applications by matching cloud-available resources. The adoption of KP-ABE is expected to enhance the security posture of cloud-based services and enable the secure sharing of resources across different cloud providers. Our novel proposal for optimizing cloud resources while prioritizing security involves using KP-ABE with container orchestration. This innovative approach will significantly impact cloud management and resource utilization.

In this chapter, we propose an innovative approach to optimizing container allocation and scheduling in cloud platforms. We use advanced optimization techniques to efficiently allocate containers to computing nodes while ensuring the secure migration of already allocated containers to new nodes. By achieving better server consolidation and reducing energy consumption, our solution promises to revolutionize container management in the cloud.

5.1 Optimized Encryption-Integrated Strategy for Containers Scheduling and Secure Migration in Multi-Cloud Data Centers

Efficiently scheduling VMs and containers in Dynamic Resource Management (DRM) is crucial for reducing energy consumption in cloud computing systems. This, in turn, lowers data center operating costs and helps mitigate their impact on the environment. Optimization modeling, a widely used approach for simulating complex problems, can be applied to translate the scheduling objective into a mathematical problem that can be solved analytically. The container placement problem involves various factors, such as decision variables, constraints, objective functions, and model assumptions. For instance, the objective function may aim to minimize latency, overall service cost, energy usage, load imbalance, or maximize SLA satisfaction for the best results.

When optimizing container-based scheduling, it is essential to address the intricate challenge of factoring in security overhead as a crucial aspect of the optimization objective. This consideration becomes more needed when containers are migrated between hosts, as secure container migration is necessary to ensure the safe relocation of sensitive information.

Combining multiple objectives into an optimization model can significantly affect the ability to find a solution, especially when time management is critical. Therefore, it is essential to have a model that considers all relevant characteristics and can deliver optimal solutions within a reasonable timeframe. In general, power consumption, consolidation trade-off with load balancing, efficiently deploy containers to computing nodes, and migrate them between different nodes with minimum costs must all be considered with encryption overhead as one optimization cost function.

In this section, we tackle the pressing issues surrounding scheduling virtualized cloud resources. Our proposed solution presents a novel two-stage container scheduling solution that addresses node imbalances and efficiently deploys containers. The proposed solution formulates the scheduling process as an optimization problem, integrating various objective functions and constraints to enhance server consolidation and minimize energy consumption. The confidentiality of migrated containers is ensured through encryption, and the associated costs are incorporated into the optimization constraints [124]. This approach ensures security in container scheduling, considering container attributes as input features in our proposed attributes-based encryption model. By carefully selecting containers and destination nodes, this work seeks to establish balance within cloud-based clusters. This contributes to the improvement of container orchestration systems and their effectiveness in real-world scenarios. The proposed solution's efficacy is demonstrated in its ability to efficiently deploy containers in multi-data center cloud environments and seamlessly migrate them between hosts within the same data center or across different data centers. Our results show optimal consolidation with a reduction in the number of running hosts, ranging from 4% to over 18%. Additionally, the solution promotes minimal total power consumption with savings ranging from 3.5 to 16.25 megawatts, while also ensuring balanced server loads.

5.1.1 Proposed work

Our research presents a comprehensive two-stage scheduling solution for containers in cloud platforms, designed to optimize energy consumption while enhancing balance across computing servers. Our power-aware optimization model incorporates support for container encryption during migration. In Stage 1, containers are deployed on the minimum number of running hosts, without factoring in additional optimization objectives. This facilitates rapid deployment and immediate customer service. Stage 2 involves a conservative optimization model that balances load on running hosts, with minimal increases in power consumption and migration. Containers are migrated to other hosts, either within the same data center or in different data centers, with encryption to guarantee data confidentiality. The encryption and decryption schemes' processing is combined with scheduling optimization constraints to provide the best possible QoS for running containers and their scheduling.

Figure 5.1 presents the interaction architecture diagram for the proposed system. The diagram depicts the optimization goal as an ellipse, the state as a rectangle, and the event as a rounded rectangle. The system's constraints and objective functions are explicitly shown in the architecture. From the starting point operation of SLA and the data center configuration, the placement algorithms work towards achieving the optimization goal. The model design is highly structured and well-organized, ensuring the desired outcome is achieved.

The optimization model process starts by carefully analyzing a set of predefined SLA Policy requirements. The optimization conditions take into account a variety of factors, including objectives, constraints, application response time, Quality of Service (QoS), end-user encryption specification, and supported encryption technologies. By using this information, the optimizer can fulfill all conditions in a selective priority check that matches optimization cost functions and constraints. This

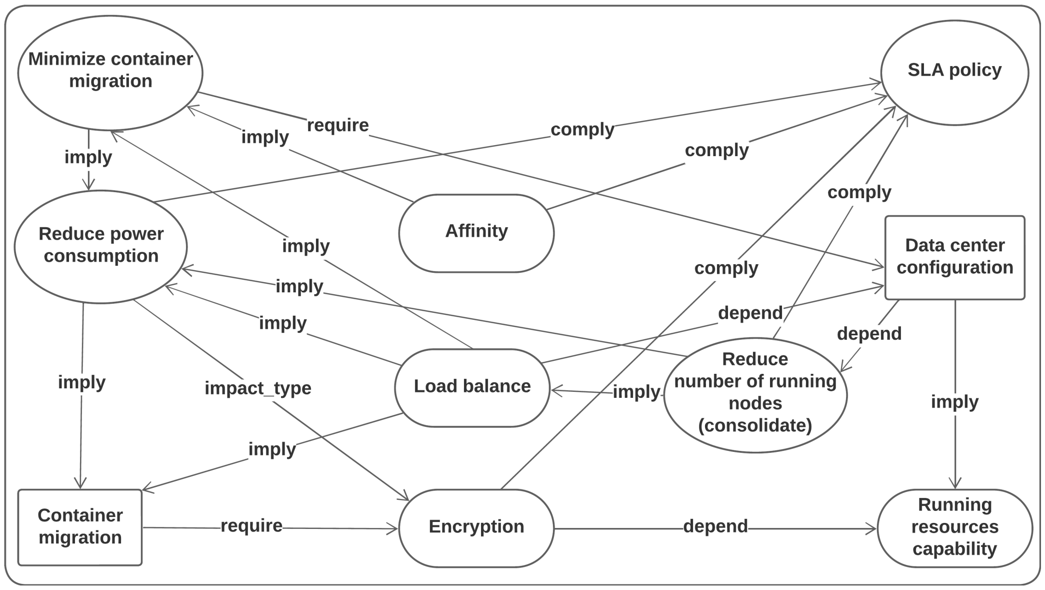


Figure 5.1: Proposed system interaction diagram.

process is broken down into two stages.

During the first stage (stage 1), the container placement scheduler is formulated to comply with the SLA and optimization goals, which prioritizes fast placement with minimum number of running hosts (hybrid first fit and bin packing algorithms). This objective is closely tied to a set of constraints that are addressed in the next stage.

During stage two, the focus shifts towards container scheduling, with the primary goals of minimizing container migrations, power consumption, and running hosts. Different events are taken into consideration when determining scheduling actions, including SLA terms, affinity, load balance, encryption cost, data-center configuration, and capabilities. In the orchestration stage, the workload on data-center resources is redistributed according to the best possible data center hardware reconfiguration while keeping the aforementioned goals in mind. This process ensures that all conditions are met while prioritizing optimization goals and compliance with the SLA requirements.

5.1.2 Containers Encryption

It is crucial to ensure that container migration is done securely, by using encryption-enabled migration based on the KP-ABE technique. When deciding where to place containers, one should take into consideration their encryption characteristics, such as the encryption algorithm used, the size of the encryption key, and the time it takes to encrypt. The time required for encryption can have an impact on the secure migration and placement of containers, especially since the hardware and server capabilities in data centers can vary. Using the same encryption technique for all containers may lead to delays and performance issues.

To address this concern, our solution incorporates custom encryption attributes based on the KP-ABE technique, such as the key size and encryption algorithm, into the cost function used to determine container placement. This approach utilizes an attribute-based encryption technique that fulfills a specific set of requirements as conditions for selecting which encryption algorithm to use and a proper key size.

Our model uses reflective learning to establish an adaptive baseline measurement for three types of encryption algorithms (AES, RSA, ECC) and different key sizes. By evaluating the container size, target host capabilities, network bandwidth, and security conditions SLA, the model determines which encryption algorithms can be used with the appropriate key size.

In our proposed solution, we leverage the robustness and security of the RSA and ECC public key architectures, along with the efficiency and effectiveness of the AES block encryption algorithm. Through the combination of these cryptographic elements, we aim to establish a highly secure and reliable container migration.

The steps involved in the hybrid AES-RSA encryption and decryption schemes are illustrated in Algorithm 5. During encryption, the container image file is first encrypted with the AES key. Subsequently, RSA encrypts both the AES encrypted file and the AES key. Decryption, on the other hand, begins with RSA decrypting both the encrypted container image file and encrypted AES key. AES then decrypts the resulting decrypted file from RSA decryption, and returns the original container image file.

Algorithm 6 outlines the steps involved in the hybrid AES-ECC encryption and decryption processes. Initially, an elliptic curve is created for the purpose of keys generation. A private ECC key is randomly generated, and then it is multiplied with the elliptic curve generator point (G) to produce the ECC public key. These two keys are utilized later to generate the encryption and decryption keys.

Algorithm 5 Hybrid AES-RSA encryption and decryption.

```
Initialization:
  1: Generate $AES\_key, RSA\_public\_key, RSA\_privatekey$
  2: Key_exchange($RSA\_private\_key$)
Encryption:
Input: Container_image_file, AES_key, RSA_public_key
Output: Encrypted_container_file, Encrypted_AES_key
  1: $AES\_encrypted\_file \leftarrow AES\_Encrypt(Container\_image\_file, AES\_key)$
  2: $Encrypted\_container\_file \leftarrow RSA\_Encrypt(AES\_encrypted\_file, RSA\_public\_key)$
  3: $Encrypted\_AES\_key \leftarrow RSA\_Encrypt(AES\_key, RSA\_public\_key)$
  4: return Encrypted_container_file, Encrypted_AES_key
Decryption:
Input: Encrypted_container_file, Encrypted_AES_key, RSA_private_key
Output: Container_image_file
  1: $AES\_encrypted\_file \leftarrow RSA\_Decrypt(Encrypted\_container\_file, RSA\_private\_key)$
  2: $AES\_key \leftarrow RSA\_Decrypt(Encrypted\_AES\_key, RSA\_private\_key)$
  3: $Container\_image\_file \leftarrow AES\_Decrypt(AES\_encrypted\_file, AES\_key)$
  4: return Container_image_file
```

Algorithm 6 Hybrid AES-ECC encryption and decryption.

```
Initialization:
1: Create $Elliptic\_curve$
2: Generate $ECC\_Keys$ :
3: $ECC\_private\_key \leftarrow Generate\_Random\_Number()$
4: $ECC\_public\_key \leftarrow ECC\_private\_key * Elliptic\_curve\_generator\_point\_G$
5: Key_exchange($ECC\_private\_key$)
Encryption:
Input: Container_image_file, ECC_public_key
Output: Encrypted_container_file, Ciphertext_public_key
1: $Ciphertext\_private\_key \leftarrow Generate\_Random\_Number()$
2: $AES\_key = Ciphertext\_private\_key * ECC\_public\_key$
3: $Encrypted\_container\_file \leftarrow AES\_Encrypt(Container\_image\_file, AES\_key)$
4: $Ciphertext\_public\_key \leftarrow Ciphertext\_private\_key * Elliptic\_curve\_generator\_point\_G$
5: return Encrypted_container_file, Ciphertext_public_key
Decryption:
Input: Encrypted_container_file, Ciphertext_public_key, ECC_private_key
Output: Container_image_file
1: $AES\_key \leftarrow ECC\_private\_key * Ciphertext\_public\_key$
2: $Container\_image\_file \leftarrow AES\_Decrypt(Encrypted\_container\_file, AES\_key)$
3: return Container_image_file
```

The encryption process begins by generating a key that AES will use to encrypt the container file. This key is created by multiplying a random number with the ECC public key. This is a speedy multiplication process that doesn't require much time. Another cyphertext public key is created at this stage by multiplying the randomly generated encryption key with the elliptic curve generator point (G). This key is sent along with the encrypted container file and will be used later to create the decryption key.

During the decryption process, the AES key is first generated by multiplying the ECC private key with the received cyphertext public key. After that, AES decrypts the file and returns the original container file. The encryption and decryption processes in AES-ECC hybrid encryption method, unlike the hybrid AES-RSA method, includes only one stage of encryption and decryption, which makes it faster.

Table 5.1 provides a ranking of the security level for the two hybrid encryption schemes: AES-RSA and AES-ECC. The ranking was determined by using different key sizes and referring to information presented by NIST in [43]. The security level rank is later utilized as part of the security

Table 5.1: Security level ranks of hybrid encryption schemes.

Encryption type	Key size (AES-RSA/ECC)	Security Level Rank
AES-RSA	128 - 1024	1
	128 - 2048	4
	128 - 4096	10
	192 - 1024	2
	192 - 2048	5
	192 - 4096	11
	256 - 1024	3
	256 - 2048	6
	256 - 4096	12
AES-ECC	128 - 256	7
	128 - 384	13
	128 - 512	16
	192 - 256	8
	192 - 384	14
	192 - 512	17
	256 - 256	9
	256 - 384	15
	256 - 512	18

SLA policy for the encryption cost function.

Based on the available host capacity, and in compliance with the SLA policy, we determine the level of encryption complexity based on the machine's capability. Then, we choose the maximum supported encryption ranking that corresponds to a specific encryption type and key-pair size, in accordance with the end-user encryption requirement.

We have listed all the SLA policies in our system in Table 5.2. A policy is established by selecting a set of performance QoS terms by the customer. The QoS policies include the application and service response times, whether they need to be strict or tolerant, and the level of security the customer requires for their containers (high, medium, low, or no security). The technology used for cloud container management is also taken into account in selecting the security SLA policy. Specifically, if the technology supports elliptic curve processing or not.

The security level ranks are classified into four encryption complexities, as shown in the state diagram in Figure 5.2. Each complexity state has a set of encryption schemes with key-pair sizes. The encryption schemes are ordered based on their processing time costs. Based on the required security SLA, an encryption complexity is first chosen. After that, the set of encryption schemes

Table 5.2: Encryption SLA policies.

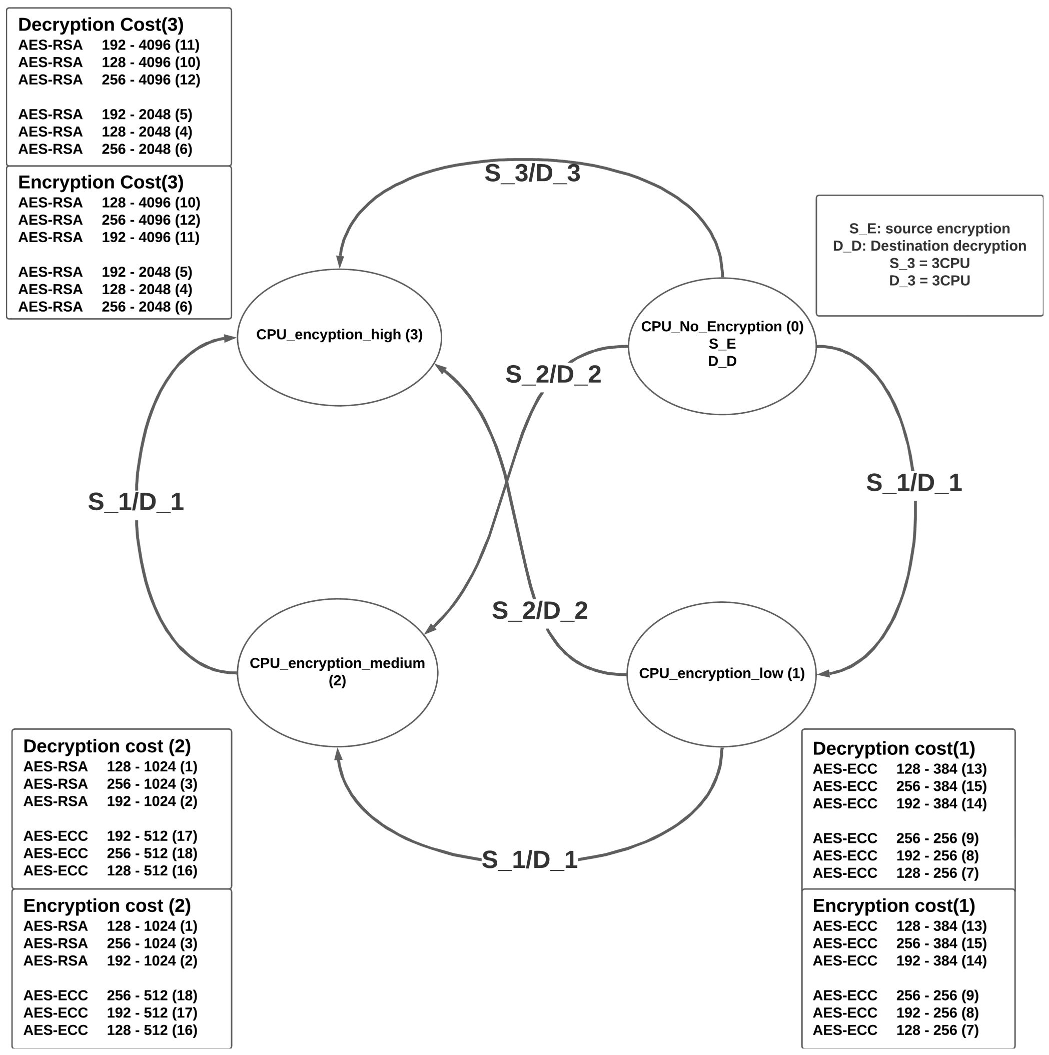
		Policy 1	Policy 2	Policy 3	...	Policy 32
Objectives	Minimize number of running nodes Minimize power consumption Minimize number of container migrations	✓	✓	✓	...	✓
Constraints	Affinity Load balancing Encryption resources requirements	✓	✓	✓	...	✓
Application QoS (Response time)	1. Strict	✓	✓	✓	...	
	2. Tolerant				...	✓
Service QoS (Response time)	1. Strict	✓	✓	✓	...	
	2. Tolerant				...	✓
End-user Encryption Requirement	1. High Encryption	✓	✓		...	
	2. Medium Encryption			✓	...	
	3. Low Encryption				...	
	4. No Encryption				...	✓
Encryption technology support	1. ECC Enabled	✓		✓	...	
	2. ECC Not Enabled		✓		...	✓

with their key-pairs are searched from top to bottom, looking for the first one that complies with the security SLA policy. In the migration process, the source host will act as the encryptor, and the destination host will act as the decryptor. The security level rank is selected based on this.

Every host in the data center has four different states that are determined by its available resources and encryption computation cost in addition to the running workload. Initially, each host starts with a state of no encryption load, which is denoted as state 0. As the host begins to receive encryption load, it progresses to either state 1, 2, or 3, depending on the associated encryption computation cost.

5.1.3 Stage 1: Relaxed Quick Optimization Placement

In stage 1, an initial placement process is performed to distribute the containers among the computing host nodes. This is done with the objective of ensuring that none of the hosts are overloaded. To achieve this, an optimization method is used that can provide a quick solution, without adding



Decryption Cost(3)

AES-RSA	192 - 4096 (11)
AES-RSA	128 - 4096 (10)
AES-RSA	256 - 4096 (12)
AES-RSA	192 - 2048 (5)
AES-RSA	128 - 2048 (4)
AES-RSA	256 - 2048 (6)

Encryption Cost(3)

AES-RSA	128 - 4096 (10)
AES-RSA	256 - 4096 (12)
AES-RSA	192 - 4096 (11)
AES-RSA	192 - 2048 (5)
AES-RSA	128 - 2048 (4)
AES-RSA	256 - 2048 (6)

Decryption cost (2)

AES-RSA	128 - 1024 (1)
AES-RSA	256 - 1024 (3)
AES-RSA	192 - 1024 (2)
AES-ECC	192 - 512 (17)
AES-ECC	256 - 512 (18)
AES-ECC	128 - 512 (16)

Encryption cost (2)

AES-RSA	128 - 1024 (1)
AES-RSA	256 - 1024 (3)
AES-RSA	192 - 1024 (2)
AES-ECC	256 - 512 (18)
AES-ECC	192 - 512 (17)
AES-ECC	128 - 512 (16)

Decryption cost(1)

AES-ECC	128 - 384 (13)
AES-ECC	256 - 384 (15)
AES-ECC	192 - 384 (14)
AES-ECC	256 - 256 (9)
AES-ECC	192 - 256 (8)
AES-ECC	128 - 256 (7)

Encryption cost(1)

AES-ECC	128 - 384 (13)
AES-ECC	256 - 384 (15)
AES-ECC	192 - 384 (14)
AES-ECC	256 - 256 (9)
AES-ECC	192 - 256 (8)
AES-ECC	128 - 256 (7)

Figure 5.2: Encryption model state diagram.

overhead to the scheduler operation. The goal is to maximize the total number of containers allocated by placing them on all available hosts, without exceeding the resource limitations of each host. Additionally, dependent containers are placed on the same host in order to consider the affinity constraint. The objective function for this stage is to maximize the number of allocated containers while ensuring that the total CPU and memory allocated for containers on a host do not exceed the resource limitations of that host. The objective function for this stage is given by Equation 32.

$$Maximize \sum_{h \in H} \sum_{c \in C} X_{ch}, \tag{32}$$

where H is the list of all host nodes in all data centers, C is the list of containers, and X is a binary matrix for allocations of containers on all hosts. X_{ch} is 1 if container c is allocated to host h.

The constraints for stage 1 are given by Equations 33, 34, 35, and 36.

$$\sum_{h \in H} X_{ch} = 1, \forall c \in C. \tag{33}$$

This constraint is used to guarantee that each container is assigned to exactly one host.

$$\sum_{c \in C} CPU_{ch} \leq CPU_h, \forall h \in H, \tag{34}$$

where CPU_{ch} is the allocated CPU for container c on host h, and CPU_h is the total CPU available at host h. This constraint is used to guarantee that the total allocated CPU in each host does not exceed the available CPU at that host.

$$\sum_{c \in C} M_{ch} \leq M_h, \forall h \in H, \tag{35}$$

where M_{ch} is the allocated memory for container c on host h, and M_h is the total memory available at host h. This constraint is used to guarantee that the total allocated memory in each host does not exceed the available memory at that host.

$$X_{ih} = 1, if \; X_{jh} = 1, \forall i, j \in D, \tag{36}$$

where D is the dependency matrix for all containers in C. Dependent containers are allocated on the same host for better application and service performance considering affinity approach.

The scheduling algorithm in our proposed scheme has prior knowledge of the migration encryption overhead that matches the source host's encryption cost requirement. The optimized scheduler takes into account the encryption cost each time a container is migrated to a destination host. The encryption constraint is dependent on the security level rank group (complexity level) assigned to each container based on the security SLA policy. Before allocating a container to a host, the host must have a certain amount of available memory and CPU based on the encryption complexity level. The security constraint is expressed in Equation 37.

Table 5.3: Encryption reserved resources percentage.

Encryption complexity	Reserved resources percentage (v)	
	Encryption	Decryption
3	20%	30%
2	10%	20%
1	5%	10%
0	0%	0%

$$\sum_{c \in C} CPU_{ch} \leq v[complexity(w)]CPU_h, \forall h \in H, w \in C, \tag{37}$$

where v denotes the percentage of available resources that must be reserved for encryption processing on host h, based on the encryption complexity level of container w as shown in Table 5.3.

5.1.4 Stage 2: The Orchestration Stage

In the second stage of the proposed solution, containers will be transferred between nodes to enhance server consolidation. This process will be executed through dynamically adaptive server load balancing. The key benefit of server consolidation is the reduction of energy consumption. Typically, a data center's power consumption is influenced by resources like CPU, memory, storage, and network, with CPU being the most significant factor. Power consumption of a computing node can be described by a linear relationship of CPU utilization on that node [125]. The migration of containers will be modeled as an optimization problem with the following objectives: (1) minimizing total energy consumption, (2) maximizing server consolidation by minimizing the number of operating nodes, and (3) minimizing the total number of container migrations.

The load of a container (L_c), given by equation 38, is the total resources reserved by this container multiplied by a load factor (α).

$$L_c = \alpha(CPU_c + M_c), \forall c \in C. \tag{38}$$

The load of a host is the total load of containers allocated to that host. This can be represented by Equation 39.

$$L_h = \sum_{c \in C} X_{ch} L_c, \forall h \in H. \tag{39}$$

The constraints of stage 2 are the constraints of stage 1 in addition to the constraints in Equation 40 and 41.

$$\sum_{h \in H} L_h \leq \hat{L}, \tag{40}$$

where $\hat{L}$ is the load threshold for the host, which is given by the mean of the load of all hosts in the data center. This constraint is used to keep the load of each host in balance with all other hosts running in the data center.

$$Y_h = \begin{cases} 0 & , if \sum_{c \in C} X_{ch} = 0, \forall h \in H \\ 1 & , if \sum_{c \in C} X_{ch} \geq 1, \forall h \in H \end{cases}, \tag{41}$$

where Y_h is a binary value for whether host h is used or not. This constraint marks the host as used or not used, so it can be later decided which hosts can be shutdown.

In this stage of the optimization process, the encryption constraint defined in Equation 37 is applied to decrypt the container migration. To ensure a smooth migration process, a percentage of the available resources (v) from Table 5.3 is reserved for hosts that act as a destination.

The power consumption for each host can be calculated as a function of CPU utilization as shown in Equation 42.

$$P(u) = KP_{max} + (1 - K)P_{max}u, \tag{42}$$

where P_{max} is the maximum power of a running host, K is the fraction of power consumed by an idle host, and u is the CPU utilization of the host. P_{max} is set to 250 W and K is set to 0.7, which are common values for recent servers [126]. The total energy consumption for any host over a period of time $[t_1, t_2]$ is then calculated as in Equation 43.

$$E = \int_{t_1}^{t_2} P(u(t))dt. \tag{43}$$

During the scheduling process, when a container is migrated, it is assigned to a different host than the one allocated in stage 1. If this new host is located in the same data center as the host from stage 1, it is considered a LAN migration. However, if the two hosts are in different data centers, it is a WAN migration. To determine the total number of container migrations, the number of containers that are allocated to different hosts in stages 1 and 2 is calculated using Equation 44.

$$N_{mig} = \sum_{h \in H} \frac{\sum_{c \in C} |X_{ch\_stage1} - X_{ch,stage2}|}{2}. \tag{44}$$

The objective functions of this stage are set to minimize the total energy consumption, the total running nodes, and the total number of container migrations. Objective functions are given by Equation 45, 46, and 47.

$$Minimize(\sum_{h \in H} E_h), \tag{45}$$

$$Minimize(\sum_{h \in H} Y_h), \tag{46}$$

$$Minimize(N_{mig}), \tag{47}$$

where E_h is the energy consumption for host h and is given by Equation 43.

5.1.5 Implementation and Experimental Setup

The proposed model was tested using a combination of real implementation and simulation. An experimental environment was built and run using the Gina Cody School HPC Facility named ”Speed” [107]. The hardware architecture and scheduler grid engine (Univa grid engine) information can be found at the Speed GitHub link. The infrastructure includes 24 nodes with 32 cores, each having 512 GB of memory and 1 TB of volatile-scratch disk space. Additionally, there are 12

NVIDIA Tesla P6 GPUs with 16 GB of memory and 1 AMD FirePro GPU with 8 GB of memory. Singularity containers are provided, which are user-managed containers with the ability to convert from Docker containers to Singularity. Various machine learning and deep learning frameworks, including TensorFlow, OpenCV, OpenMPI, OpenISS, MARF, and OpenFOAM are also supported.

To study the performance of the used hybrid encryption schemes: AES-RSA and AES-ECC, we emulated the encryption of a 100 MB file. We tested different key-pair sizes for the two hybrid encryption schemes. For AES, we tested key sizes of 128, 192, and 256 bits. For RSA, we tested key sizes of 1024, 2048, and 4096 bits. For ECC, we tested key sizes of 256, 384, and 512 bits. The emulation experiments were conducted 100 times for each of the two encryption schemes, and the mean values, with confidence intervals, were calculated.

In order to apply RSA encryption, we utilized the Python-RSA library [127], a pure-Python RSA implementation. This library facilitates encryption, decryption, signature generation, signature verification, and key generation in accordance with PKCS#1 version 1.5 [128]. In our AES implementation, we incorporated the AES library from the PyCryptodome Python package, which provides access to low-level cryptographic primitives [129]. Additionally, random secret encryption keys of 128, 192, and 256 bits were generated using the Python Secrets library [130]. In our ECC implementation, we employed the Tinyec library [131] for conducting arithmetic computations on elliptic curves, as well as the Secrets library for the generation of random secret encryption keys. These libraries were instrumental in facilitating the execution of essential operations within our implementation.

To validate the scheduling part of the solution, the latest Google Borg clusters trace log version 3 [132, 133] was integrated with the CloudSim simulator [10, 11]. The proposed optimization scheduler solution was then implemented in the CloudSim simulator. We applied the same resource amounts from the trace to simulate a huge data center with 96.4K hosts, memory, and CPU capacities. We then replicated that data center into 5 data centers to simulate the behavior of multiple data centers and measure the impact of container migration across data centers via WAN with network link speed of 1 Gbps. We applied it on six different cases of varying container numbers, including 500K, 750K, 1M, 1.5M, 1.75M, and 2M containers. These containers had different resource and security SLA requirements, and we analyzed the performance of our solution under each scenario.

The two stages of our method are modeled as a mixed-integer linear programming (MILP) optimization problem. Solving these optimization problems must be done in a relatively short period of time to avoid adding additional overhead to the scheduler operation. To preserve the linearity of the problem, we used the CP-SAT (Constraint Programming - Satisfiability) solver available in the Google OR-Tools library [134]. CP-SAT is particularly well-suited for problems where discrete decision variables and complex constraints are involved, making it ideal for efficient container placement and migration.

The optimization scheduler was run in different run cases, partially (scenarios C1 and C2) and then fully (scenario C3), as shown in Table 5.4. This was done to check and compare the performance of different parts of the scheduler and the whole system by enabling different optimization objective functions and constraints in different steps to check the impact of each optimization function and constraint on the scheduling optimization solution. We first ran Stage 1 (S1) and saved the data center's status. Then we ran Stage 2 and gradually applied different optimization objectives to have the three scenario cases: C1, C2, and C3, explained in Table 5.4, and compared the performance of different objective functions of our proposed solution.

5.1.6 Results and Discussion

In this section, we will display the outcomes of the two parts of the solution independently. Firstly, we will showcase the results of data encryption using two hybrid encryption schemes,

Table 5.4: Scheduling optimization run cases.

Run case	Applied objective functions and constraints
S1	Stage 1 objectives and constraints
C1	1- Minimize number of migrations 2- Encryption constraint
C2	1- Minimize number of migrations 2- Minimize power consumption 3- Encryption constraint
C3 (Full Solution)	1- Minimize number of migrations 2- Minimize power consumption 3- Minimize number of running nodes 4- Encryption constraint 5- Server load balancing constraint

namely AES-RSA and AES-ECC. Subsequently, we will present the outcomes of the scheduling optimization part.

A. Containers Data Encryption

We carried out a standardized baseline experiment to assess the key generation, encryption, and decryption of migrated containers of varying sizes. The baseline was set at 100 MB to provide performance benchmarks for the encryption and decryption stages. We conducted 100 iterations for each of the two encryption schemes utilized (AES-RSA and AES-ECC). Mean values were calculated and validated for accuracy using confidence error ratios. The AES-RSA experiments showed confidence error ratios between 6% and 15% for key generation, while the AES-RSA encryption method yielded ratios between 0.1% and 1%. The AES-RSA decryption ratios were between 0.1% and 0.3%. Key generation for AES-ECC encryption experiments yielded error ratios ranging from 1% to 5%, while encryption and decryption stages showed ratios between 0.1% and 2%.

Figure 5.3 illustrates the delay associated with the generation of encryption keys within the hybrid AES-RSA encryption scheme. Notably, the delay of generating encryption keys escalates proportionally with the size of the RSA key, as evidenced by the significant rise in generation time when using a 4096-bit RSA key.

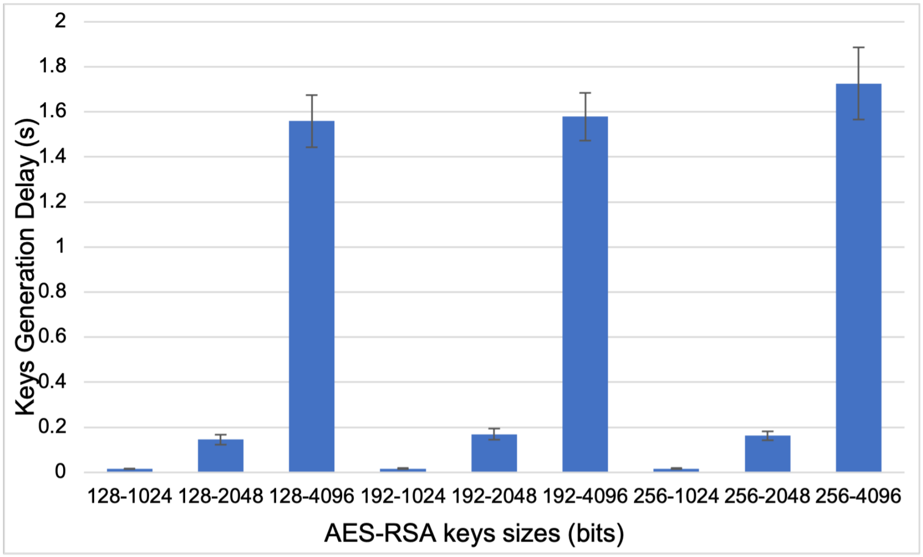


Figure 5.3: AES-RSA keys generation delay.

Figure 5.4 presents the encryption delay observed in the context of a 100 MB file utilizing the hybrid AES-RSA encryption scheme. It is worth noting that the AES encryption process shows superior speed compared to RSA encryption, especially when larger RSA keys are employed, as the latter requires significantly longer durations. This difference in encryption times becomes more noticeable, thus highlighting the significant impact of RSA encryption on the overall encryption delay within the AES-RSA hybrid scheme.

The decryption delay for a 100 MB file in the hybrid AES-RSA encryption paradigm is illustrated in Figure 5.5. Similar to the RSA encryption procedure, the decryption phase incurs longer processing times with increasing RSA key sizes. The data depicted in the figure shows a clear trend of increasing decryption times with larger RSA key sizes, which exceeds the processing durations observed during encryption.

Furthermore, it is worth noting that RSA decryption generally requires more time compared to RSA encryption. This is primarily due to the larger size of the private exponent used in the decryption process of RSA. The computational complexity of raising the ciphertext to the power of the private exponent is greater than that of raising the plaintext to the power of the public exponent during encryption.

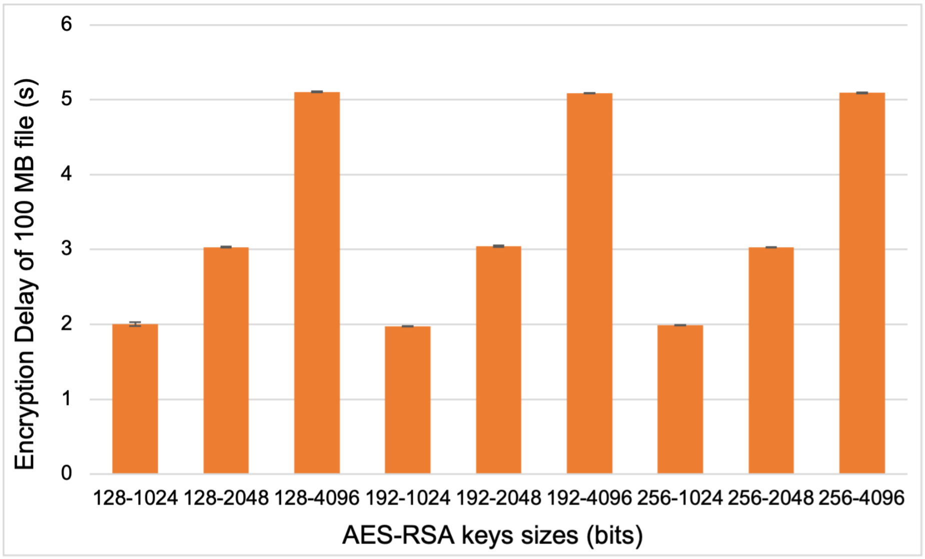


Figure 5.4: AES-RSA encryption delay of 100 MB file.

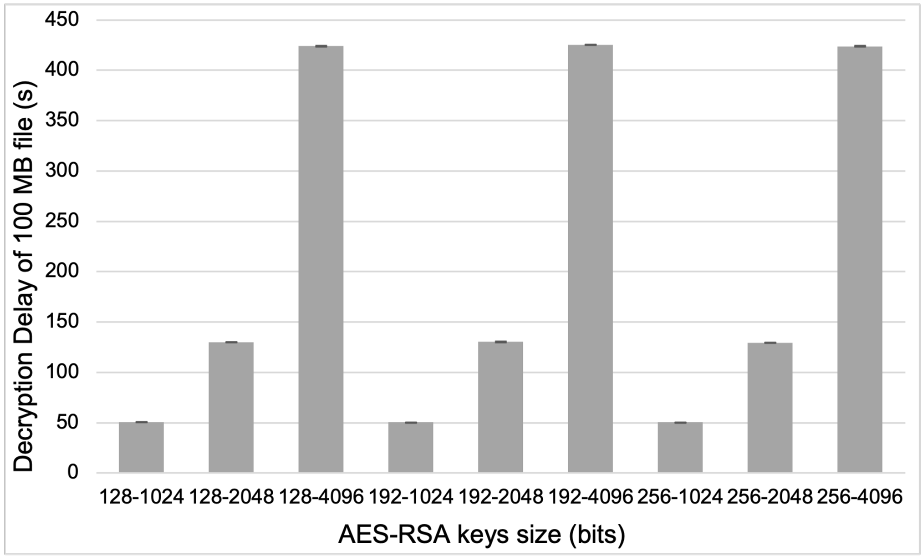


Figure 5.5: AES-RSA decryption delay of 100 MB file.

Elliptic Curve Cryptography (ECC) utilizes an elliptic curve multiplication process for generating keys. This results in a notably faster key generation procedure for the hybrid AES-ECC encryption scheme compared to AES-RSA. Figure 5.6 demonstrates the delay incurred during key generation for the hybrid AES-ECC encryption scheme. While larger key sizes may cause longer processing delays, it is important to note that the key generation process is generally quick, with a maximum duration of 5 milliseconds.

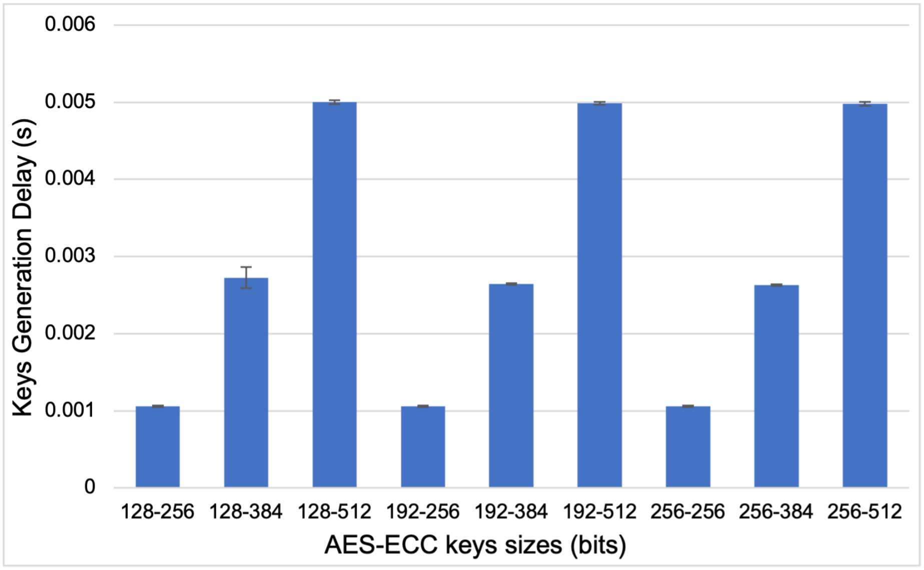


Figure 5.6: AES-ECC keys generation delay.

The encryption process within the AES-ECC scheme demonstrates superior speed compared to the AES-RSA scheme due to the singular application of AES for encryption, with ECC exclusively employed for key generation. Figure 5.7 illustrates the encryption delay associated with a 100 MB file within the hybrid AES-ECC encryption scheme. It is observed that the encryption processing delay escalates proportionally with the size of the key pair, indicating that larger keys introduce longer processing durations.

The decryption phase in the AES-ECC scheme works the same way as the encryption phase. When decrypting a 100 MB file using the hybrid AES-ECC encryption scheme, the decryption delay is directly proportional to the key pair size, just like the encryption phase, as shown in Figure 5.8. It is important to note that larger keys require longer processing times. However, unlike the AES-RSA scheme, the AES-ECC decryption process is characterized by shorter processing delays compared to its encryption counterpart.

B. Containers Scheduling Optimization

Figure 5.9 depicts the aggregate number of running hosts across the five data centers. The figure illustrates the overall percentage of utilized hosts relative to the total available hosts in these data centers, considering various optimization objectives and diverse quantities of containers. The

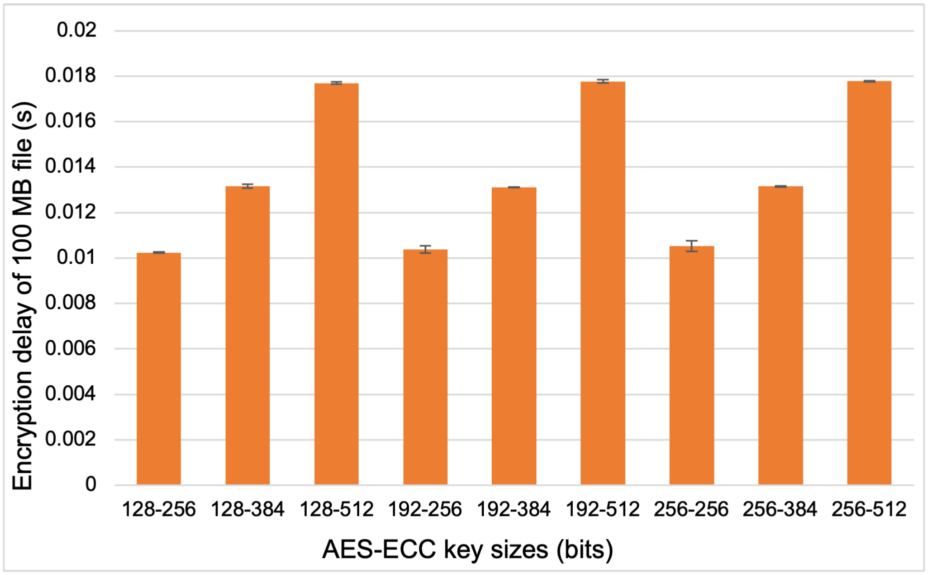


Figure 5.7: AES-ECC encryption delay of 100 MB file.

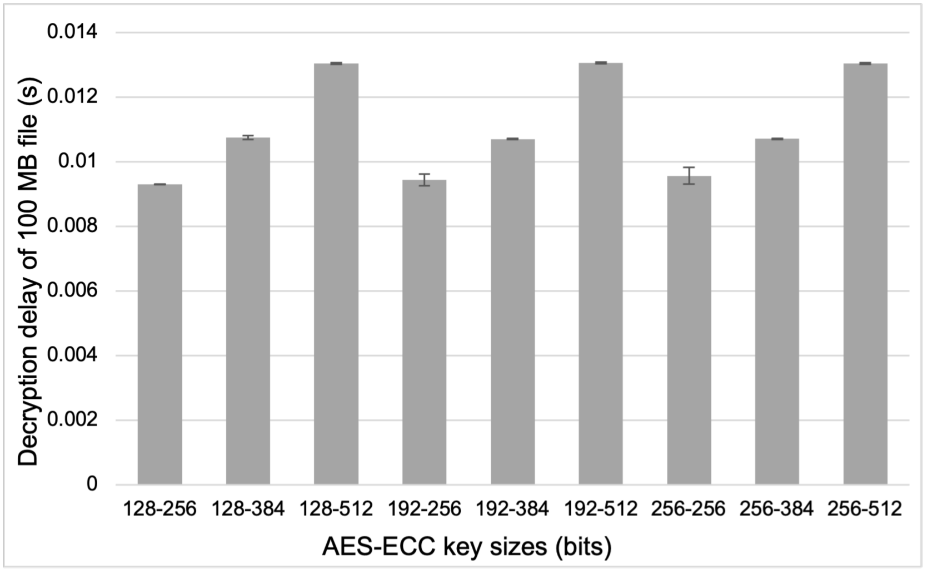


Figure 5.8: AES-ECC decryption delay of 100 MB file.

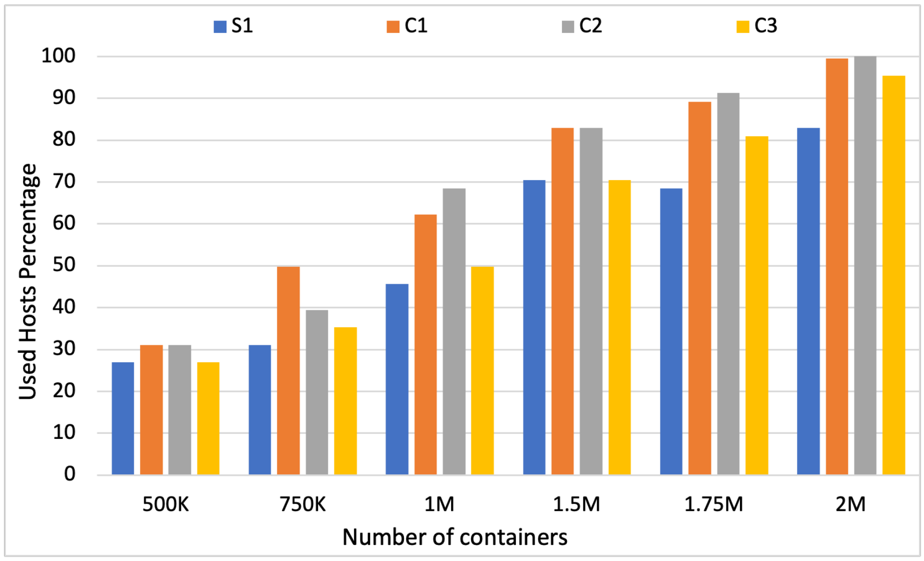


Figure 5.9: Used hosts percentage for different run case scenarios (S1, C1, C2, and C3).

scheduling unfolds in two stages, commencing with Stage 1 (S1), followed by the application of three distinct objective scenarios in Stage 2 (C1, C2, and C3).

Our proposed full consolidation optimization objectives solution, denoted as case scenario (C3), resulted in a remarkable decrease in the number of running hosts, ranging from 4% to over 18% with respect to the maximum number of running hosts in all other cases. This improvement has significant implications for the five data centers tested, where each 1% represents a huge number of hosts, which results in considerable impacts on the overall power consumption.

Figure 5.10 depicts the aggregate power consumption of all data centers, which varies based on the number of containers and objectives applied. When correlated with Figure 5.9, which shows the total number of utilized hosts, it is evident that our proposed optimization scheduling solution can reduce the number of active hosts in data centers. This reduction ultimately results in significant power savings, ranging from 3.5 to 16.25 megawatts when using our proposed scheduling solution in optimization scenario C3.

Figure 5.11 presents the total count of migrations across all data centers, encompassing both LAN and WAN migrations. It is worth noting that migrations take place solely during the second stage of the solution. Therefore, the outcomes presented in this study are directly relevant to the objective scenarios implemented in stage 2. Due to the application of different objectives in the scheduling process, varying numbers of LAN and WAN migrations occur. However, it is important to note that the overall number of migrations remains the same for each unique set of objectives applied across different quantities of containers. This consistency results from the optimization objective that aims to minimize the total number of migrations, a feature that remains constant in all three scenarios presented in the diagram.

The cumulative volume of migrated data resulting from both LAN and WAN migrations across

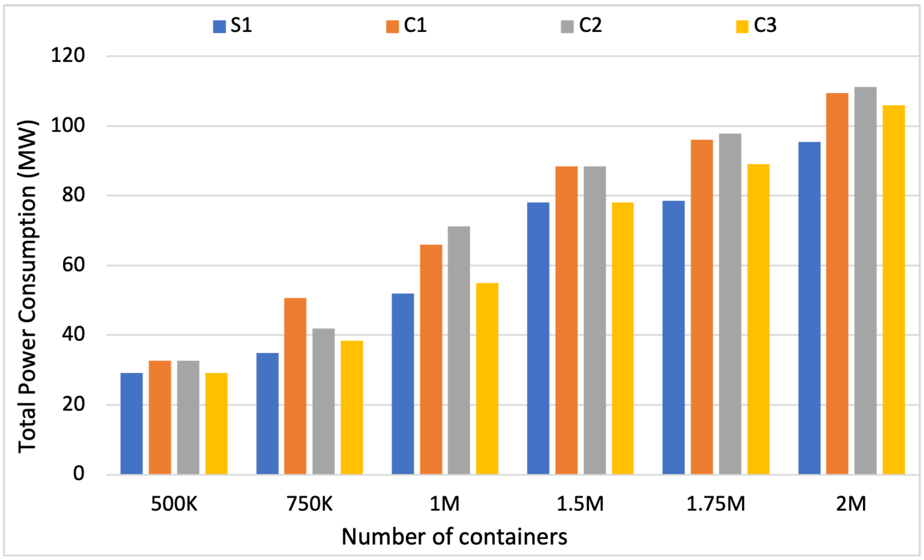


Figure 5.10: Power consumption for different run case scenarios (S1, C1, C2, and C3).

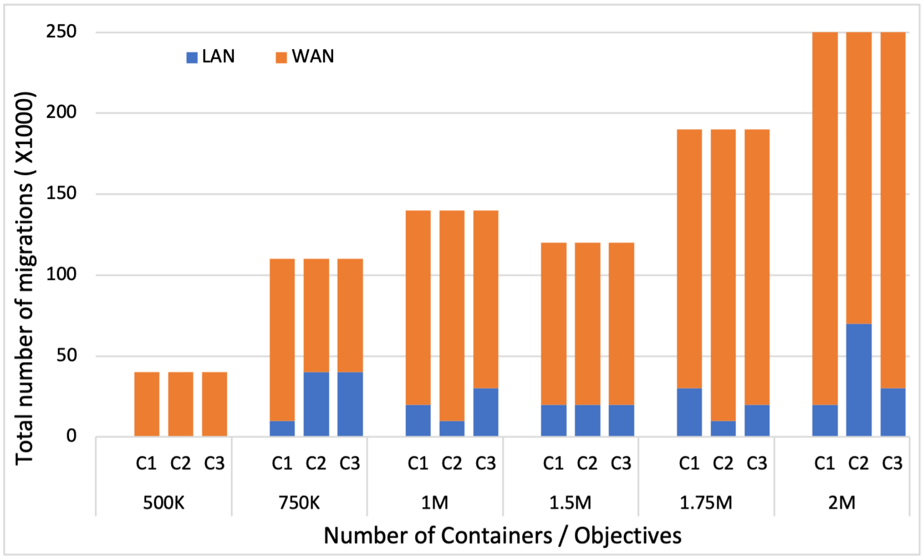


Figure 5.11: Total number of migrations for different case scenarios (C1, C2, and C3) and numbers of containers.

all data centers is depicted in Figure 5.12. The total migrated data is closely tied to the number of container migrations and the sizes of the migrated containers. As different optimization objectives result in the selection of different containers for migration, the total volume of migrated data varies in each scenario.

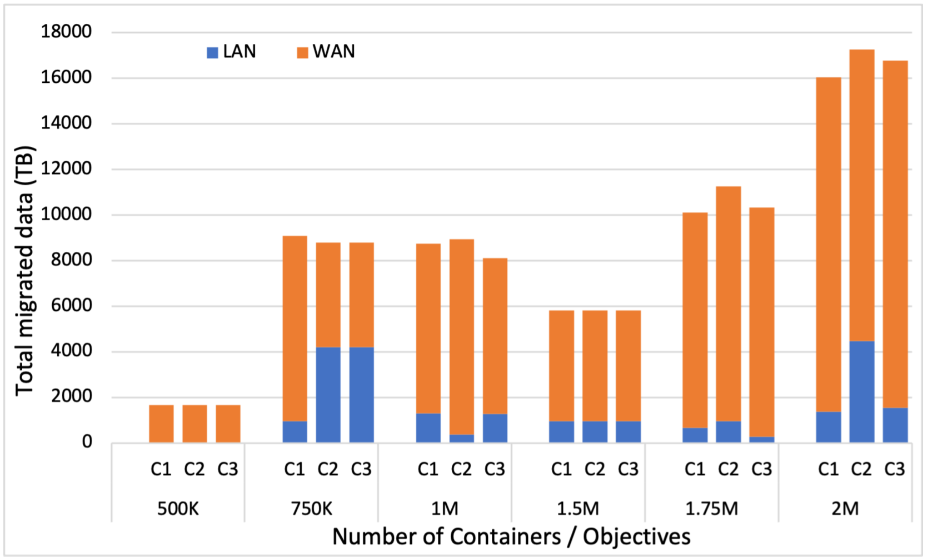


Figure 5.12: Total migrated data for different case scenarios (C1, C2, and C3) and numbers of containers.

Figure 5.13 shows a detailed analysis of the total migration time, taking into consideration LAN and WAN migrations across all data centers. This illustration factors in differences in container quantities and optimization goals. The total migration time is affected by the combined impact of the total amount of data migrated and the operational speed of the network links involved in the migration process. It is important to note that the careful selection of containers for migration, based on specific optimization objectives in each instance, leads to noticeable fluctuations in the total migration time for various container quantities.

Figure 5.14 shows a detailed overview of the total delays that occur during encryption for all migrations, including both LAN and WAN migrations, across various data centers. These delays cover the entire processing timespan, including the generation of encryption keys, container encryption, and decryption for each container migration. Encryption processes can cause significant delays, particularly in cases with a higher number of migrations and larger volumes of container data. As a result, more migrations and larger volumes of data will lead to increased encryption processing delays.

Figures 5.15 and 5.16 present a visual depiction of memory and CPU utilization in data centers. These values are measured by running the proposed optimised scheduler with the scenarios (S1, C1, C2, and C3) for different number of containers used. The results are systematically presented,

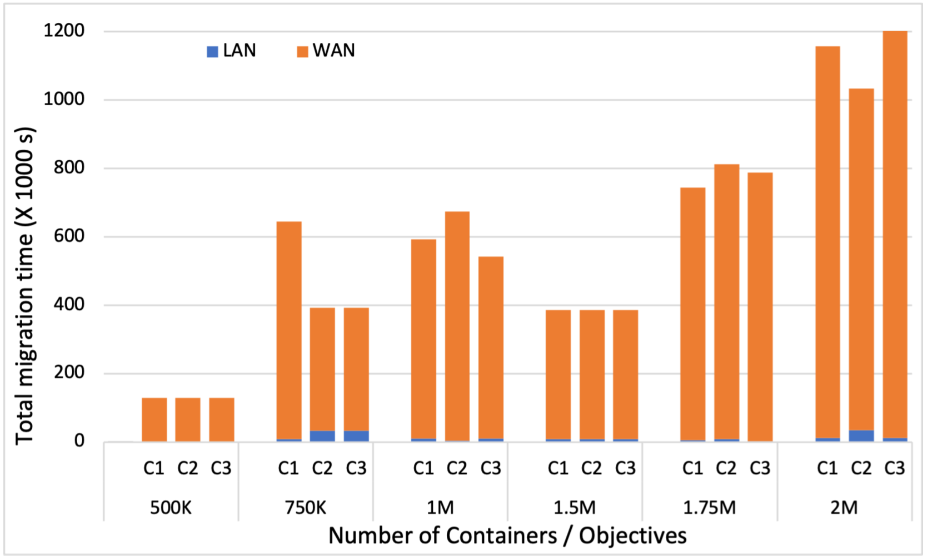


Figure 5.13: Total migration times for different case scenarios (C1, C2, and C3) and numbers of containers.

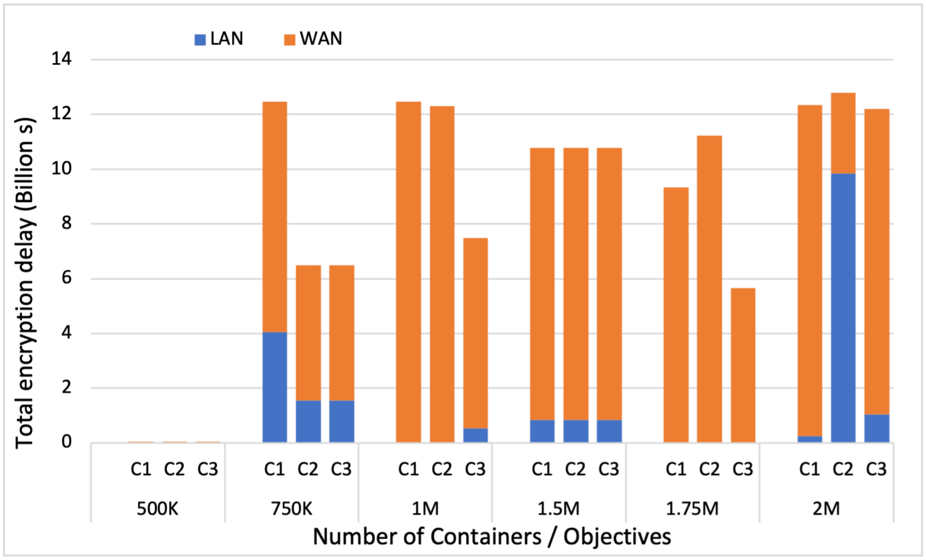


Figure 5.14: Total encryption delays for different case scenarios (C1, C2, and C3) and numbers of containers.

with subfigures (S1) illustrating the memory and CPU utilization of the five data centers after the initial stage (S1). Additionally, subfigures (C1), (C2), and (C3) provide detailed explanations of the memory and CPU utilization subsequent to stage 2, each aligned with the constraints of distinct objective scenarios denoted as C1, C2, and C3, respectively. For a comprehensive elucidation of the proposed scheduling optimization solution, Table 5.4 offers detailed explanations for the objectives scenarios S1, C1, C2, and C3.

The subfigures (C3) in Figures 5.15 and 5.16 depict the overall memory and CPU utilization by running the aforementioned case scenario (C3) of our optimization scheduler. Our scheduler has a remarkable achievement as it takes into account all objectives and constraints to achieve a balanced tradeoff between power consumption and load balancing. The load balance constraint has been instrumental in redistributing the load from the overloaded data centers to the underloaded ones, as illustrated in the subfigures (C3). Although the values may appear small, every 1% represents a significant amount of memory and CPU utilization due to the large capacities of data center resources. Load balancing is essential, and our results show its tangible significance.

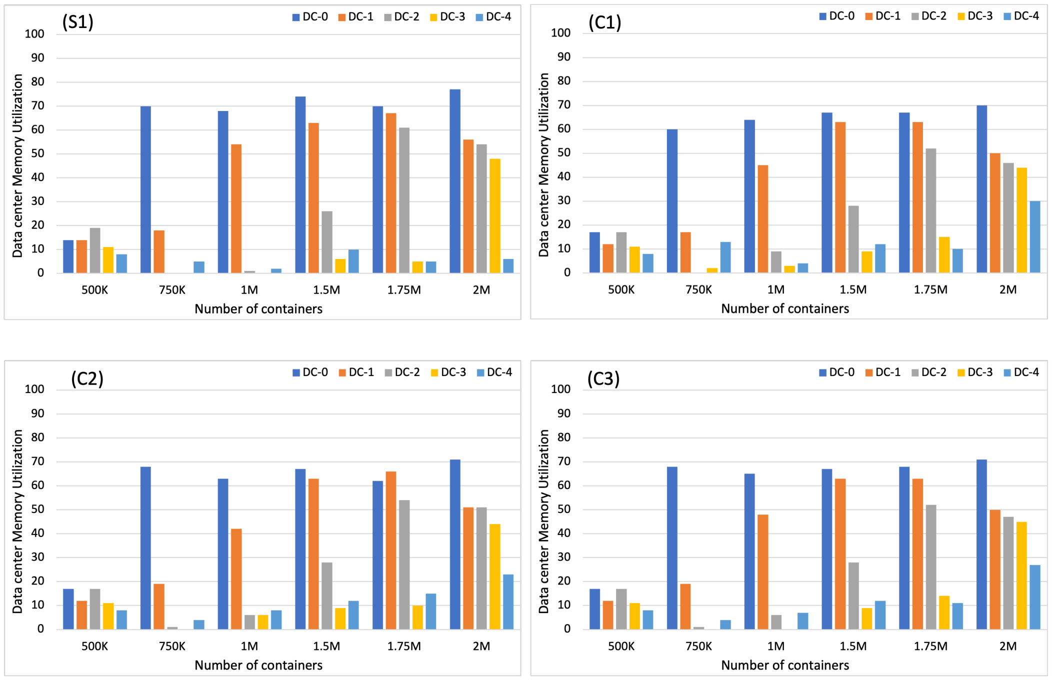


Figure 5.15: Data centers memory utilization for different objectives applied case scenarios (S1, C1, C2, and C3).

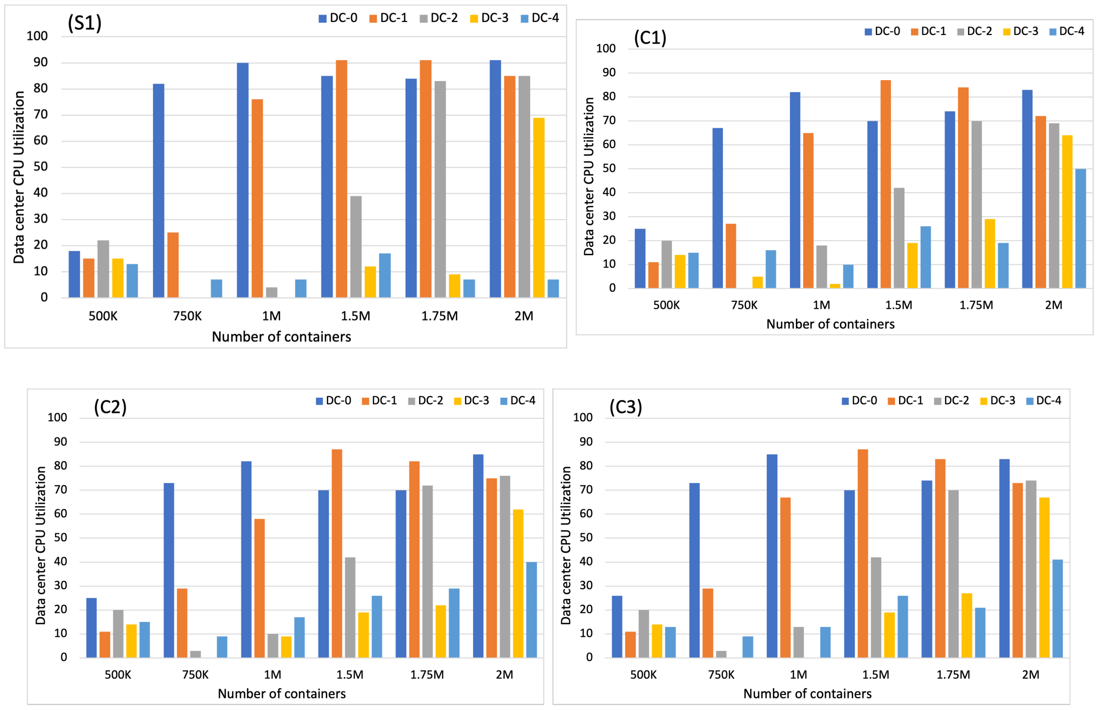


Figure 5.16: Data centers CPU utilization for different objectives applied case scenarios (S1, C1, C2, and C3).

5.2 Chapter Summary

Containers are recognized for their lightweight and virtualization efficiency, making them a vital element in modern application orchestration. In this context, the scheduler is crucial in strategically distributing containers across diverse computing nodes. This chapter presented a novel two-stage container scheduling solution that addresses node imbalances and efficiently deploys containers. The proposed solution formulates the scheduling process as an optimization problem, integrating various objective functions and constraints to enhance server consolidation and minimize energy consumption. The confidentiality of migrated containers is ensured through encryption, and the associated costs are incorporated into the optimization constraints. This approach ensures security in container scheduling, considering container attributes as input features in our proposed attributes-based encryption model. By carefully selecting containers and destination nodes, this work seeks to establish balance within cloud-based clusters. This contributes to the improvement of container orchestration systems and their effectiveness in real-world scenarios. The proposed solution's efficacy is demonstrated in its ability to efficiently deploy containers in multi-data center cloud environments and seamlessly migrate them between hosts within the same data center or across different data centers. The results show optimal consolidation with a reduction in the number of running hosts, ranging from 4% to over 18%. Additionally, the solution promotes minimal total power consumption with savings ranging from 3.5 to 16.25 megawatts, while also ensuring balanced server loads, highlighting the effectiveness of the proposed container scheduling approach.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have provided the mathematical models for the classical live VM migration methods: pre-copy and post-copy, and hybrid-copy methods of these two. Mathematical models are important to study cloud systems and different cloud policies in simulation environments. A new extension of the famous cloud simulator: CloudSim, is provided with support for different live VM migration methods for LAN and WAN migration.

We introduced a blockchain-based distributed cloud resources management model that employs the BigchainDB framework as a blockchain registry. This registry allows for secure and efficient information sharing about data centers, facilitating quick and dependable scheduling decisions for various cloud data centers and edge nodes. Additionally, we implemented machine learning as a VM migration service that selects a migration method based on specific VM characteristics and SLA policies to comply with. Compared to other solutions, our proposed distributed model approach requires less management overhead and delay. Our blockchain-based model reduces the total messages exchanged for the VM migration with percentages ranging from 0.5% to 22% and the total communication delay by 8% to 72% compared to another REST-based models. The proposed blockchain-based distributed model has also shown a remarkable decrease of 97.9% in exchanged communication messages, as well as a significant reduction in total delay ranging from 49.4% to

91.7%, when compared to a VPN-based centralized model. Furthermore, our machine learning algorithms for the VM migration service demonstrate acceptable training time and SLA compliance rate, and significantly more accurate prediction than other methods presented in the literature. The SLA compliance rate of the proposed solution ranges from 18% to 94.9% for different machine learning algorithms and SLA policies during VM migrations across WAN-connected data centers. The proposed solution reduces the total migration time by 14% to 79% and the downtime by 64% to 99%. These results demonstrate the effectiveness of the proposed model and its potential to improve the efficiency of cloud systems management.

Finally, we introduced a two-stage container scheduling solution that efficiently deploys containers and addresses imbalances across nodes through migration. The solution formulates the scheduling process as an optimization problem and incorporates various objective functions and constraints to improve server consolidation and reduce energy consumption. Encryption is used to ensure the confidentiality of containers during migration. The cost of encryption and decryption has been included in the optimization constraints of the proposed solution. The solution has been shown to be effective in multi-data center cloud environments, contributing to the improvement of container orchestration systems in real-world scenarios. Our results show optimal consolidation with a reduction in the number of running hosts, ranging from 4% to over 18%. Additionally, the solution promotes minimal total power consumption with savings ranging from 3.5 to 16.25 megawatts, while also ensuring balanced server loads.

6.2 Future Work

Our future work involves implementing our blockchain-based proposed solution in a real production cloud management setup to address other cloud management components that could impact the operation of our solution and impose overhead implications on its performance. Furthermore, we aim to improve our system architecture so that it can seamlessly operate within any cloud management system.

Our upcoming research will be dedicated to expanding our current understanding of container and VM security. Specifically, we will be focusing on three crucial levels of security: data at rest, data in use, and data in transit. Our goal is to explore each of these areas in greater depth and develop comprehensive strategies that can effectively safeguard sensitive information.

One promising approach that we plan to investigate is the use of homomorphic encryption. This technique allows computations to be performed on encrypted data without the need to decrypt it first, which could significantly enhance security for data storage and processing within containers and VMs. By incorporating homomorphic encryption into our security framework, we hope to provide an extra layer of protection for data at rest and in use, ensuring confidentiality and integrity while maintaining seamless operations within virtualized environments.

In addition, we will be examining the potential benefits of artificial intelligence (AI) algorithms in the scheduling of VMs and containers. Effective resource allocation and workload distribution are critical for optimizing performance and minimizing vulnerabilities in virtualized environments. By leveraging AI-driven scheduling algorithms, we can dynamically allocate resources based on workload demands, prioritize critical tasks, and mitigate potential security risks such as resource contention and denial-of-service attacks. Our aim is to develop intelligent scheduling mechanisms that enhance the overall security of containerized and virtualized infrastructures while maximizing resource utilization and minimizing operational overhead.

Through these research endeavors, we hope to contribute to the advancement of container and VM security practices. Our insights and solutions will address the evolving challenges and complexities inherent in modern computing environments, empowering organizations to embrace the benefits of containerization and virtualization technologies while ensuring the confidentiality, integrity, and availability of their data and resources.

Bibliography

[1] RedHat, “What is a virtual machine (vm)?” 2022, last accessed 20 October 2022. [Online]. Available: https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine

[2] Docker, “What is a container? - docker,” 2023, last accessed 15 September 2023. [Online]. Available: https://www.docker.com/resources/what-container/

[3] O. Oleghe, “Container placement and migration in edge computing: Concept and scheduling models,” *IEEE Access*, vol. 9, pp. 68 028–68 043, 2021.

[4] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, “A container-based edge cloud paas architecture based on raspberry pi clusters,” in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW).* IEEE, 2016, pp. 117–124.

[5] Weaveworks, “Docker vs virtual machines (vms) : A practical guide to docker containers and vms,” 2020, last accessed 15 September 2023. [Online]. Available: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation*, ser. NSDI’05, vol. 2. USA: USENIX Association, 2005, p. 273–286.

[7] M. R. Hines, U. Deshpande, and K. Gopalan, “Post-copy live migration of virtual machines,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, p. 14–26, jul 2009. [Online]. Available: https://doi.org/10.1145/1618525.1618528

[8] A. Araldo, A. D. Stefano, and A. D. Stefano, "Resource allocation for edge computing with multiple tenant configurations," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1190–1199. [Online]. Available: https://doi.org/10.1145/3341105.3374026

[9] R. W. Ahmad, A. Gani, S. H. Ab. Hamid, M. Shiraz, F. Xia, and S. A. Madani, "Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues," *The Journal of Supercomputing*, vol. 71, pp. 2473–2515, 2015.

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[11] T. C. Computing and U. o. M. Distributed Systems (CLOUDS) Laboratory, "Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services," 2023, last accessed 10 October 2023. [Online]. Available: http://www.cloudbus.org/cloudsim/

[12] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Comput. Surv.*, vol. 51, no. 6, feb 2019. [Online]. Available: https://doi.org/10.1145/3281010

[13] C. Pham, D. T. Nguyen, Y. Njah, N. H. Tran, K. K. Nguyen, and M. Cheriet, "Share-to-run iot services in edge cloud computing," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 497–509, 2021.

[14] T. Hirofuchi, H. Nakada, H. Ogawa, S. Itoh, and S. Sekiguchi, "A live storage migration mechanism over wan and its performance evaluation," in *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing*, ser. VTDC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 67–74. [Online]. Available: https://doi.org/10.1145/1555336.1555348

[15] U. Mandal, P. Chowdhury, M. Tornatore, C. U. Martel, and B. Mukherjee, "Bandwidth provisioning for virtual machine migration in cloud: Strategy and application," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 967–976, 2018.

[16] M. A. Altahat, A. Agarwal, N. Goel, and M. Zaman, "Analysis and comparison of live virtual machine migration methods," in *2018 IEEE 6th international Conference on Future internet of Things and Cloud (FiCloud)*. IEEE, 2018, pp. 251–258.

[17] M. A. Altahat, A. Agarwal, N. Goel, and J. Kozlowski, "Dynamic hybrid-copy live virtual machine migration: Analysis and comparison," *Procedia Computer Science*, vol. 171, pp. 1459–1468, 2020, third International Conference on Computing and Network Communications (CoCoNet'19). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920311352

[18] M. Sharma and R. Garg, "An artificial neural network based approach for energy efficient task scheduling in cloud data centers," *Sustainable Computing: Informatics and Systems*, vol. 26, p. 100373, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537918302798

[19] S. Ouhame, Y. Hadi, and A. Ullah, "An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model," *Neural Computing and Applications*, vol. 33, pp. 10 043–10 055, 2021.

[20] S. Kumar T, S. D. S. Mustapha, P. Gupta, and R. P. Tripathi, "Hybrid approach for resource allocation in cloud infrastructure using random forest and genetic algorithm," *Scientific Programming*, vol. 2021, pp. 1–10, 2021.

[21] G. Singh and P. Gupta, "A review on migration techniques and challenges in live virtual machine migration," in *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2016, pp. 542–546.

[22] M. Soni, "The cloudsim framework: Modelling and simulating the cloud environment," 2014, last accessed 14 August 2021. [Online]. Available: https://www.opensourceforu.com/2014/03/cloudsim-framework-modelling-simulating-cloud-environment/

[23] I. C. Education, “What are neural networks?” 2020, last accessed 14 October 2021. [Online]. Available: https://www.ibm.com/topics/neural-networks

[24] ——, “What are convolutional neural networks?” 2020, last accessed 14 October 2021. [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks

[25] ——, “What are recurrent neural networks?” 2020, last accessed 14 October 2021. [Online]. Available: https://www.ibm.com/topics/recurrent-neural-networks

[26] R. Awadallah, A. Samsudin, J. S. Teh, and M. Almazrooie, “An integrated architecture for maintaining security in cloud computing based on blockchain,” *IEEE Access*, vol. 9, pp. 69 513–69 526, 2021.

[27] C. V. B. Murthy, M. L. Shri, S. Kadry, and S. Lim, “Blockchain based cloud computing: Architecture and research challenges,” *IEEE access*, vol. 8, pp. 205 190–205 205, 2020.

[28] P. Stetsenko and G. Khalimov, “Blockchain-based protocol for ensuring authenticity of data origin in cloud environments.” in *ICST*, 2020, pp. 176–190.

[29] MongoDB, “Mongodb,” 2023, last accessed April 2023. [Online]. Available: https://www.mongodb.com

[30] Tendermint, “Tendermint,” 2023, last accessed April 2023. [Online]. Available: https://tendermint.com/

[31] B. GmbH, “Bigchaindb 2.0: the blockchain database,” 2018.

[32] Kubernetes, “Kubernetes: Production-grade container orchestration,” 2023, last accessed 25 September 2023. [Online]. Available: https://kubernetes.io/

[33] C. Rosen, “Docker swarm vs. kubernetes: A comparison,” IBM Blog, June 2022, last accessed 25 September 2023. [Online]. Available: https://www.ibm.com/blog/docker-swarm-vs-kubernetes-a-comparison/

[34] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, “A survey on virtual machine migration: Challenges, techniques, and open issues,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

[35] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, “Containers checkpointing and live migration,” in *Proceedings of the Linux Symposium*, vol. 2, 2008, pp. 85–90.

[36] J. Pecholt, M. Huber, and S. Wessel, “Live migration of operating system containers in encrypted virtual machines,” in *Proceedings of the 2021 on Cloud Computing Security Workshop*, 2021, pp. 125–137.

[37] F. Brasser, P. Jauernig, F. Pustelnik, A.-R. Sadeghi, and E. Stapf, “Trusted container extensions for container-based confidential computing,” *arXiv preprint arXiv:2205.05747*, 2022.

[38] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, “Enabling live migration of containerized applications across clouds,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2529–2538.

[39] G. Singh, P. Singh, A. Motii, and M. Hedabou, “A secure and lightweight container migration technique in cloud computing,” *Journal of King Saud University-Computer and Information Sciences*, p. 101887, 2023.

[40] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[41] V. Rijmen and J. Daemen, “Advanced encryption standard,” *Proceedings of federal information processing standards publications, national institute of standards and technology*, vol. 19, p. 22, 2001.

[42] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[43] B. Elaine, “Recommendation for key management, part 1: General,” *NIST Special Publication 800-57 Part 1 Revision 5*, 2020.

[44] D. Patel, B. Patel, J. Vasa, and M. Patel, “A comparison of the key size and security level of the ecc and rsa algorithms with a focus on cloud/fog computing,” in *International Conference on Information and Communication Technology for Intelligent Systems*. Springer, 2023, pp. 43–53.

[45] T. Wood, K. Ramakrishnan, P. Shenoy, J. Van der Merwe, J. Hwang, G. Liu, and L. Chaufournier, “Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines,” *IEEE/ACM Transactions On Networking*, vol. 23, no. 5, pp. 1568–1583, 2014.

[46] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty, “{XvMotion}:{Unified} virtual machine migration over long distance,” in *2014 usenix annual technical conference (usenix atc 14)*, 2014, pp. 97–108.

[47] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, “Live wide-area migration of virtual machines including local persistent state,” in *Proceedings of the 3rd international conference on Virtual execution environments*, 2007, pp. 169–179.

[48] S. He, C. Hu, B. Shi, T. Wo, and B. Li, “Optimizing virtual machine live migration without shared storage in hybrid clouds,” in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2016, pp. 921–928.

[49] U. Deshpande and K. Keahey, “Traffic-sensitive live migration of virtual machines,” *Future Generation Computer Systems*, vol. 72, pp. 118–128, 2017.

[50] H. Wang, F. Li, W. Mo, P. Tao, H. Shen, Y. Wu, Y. Zhang, and F. Deng, “Novel cloud-edge collaborative detection technique for detecting defects in pv components, based on transfer learning,” *Energies*, vol. 15, no. 21, p. 7924, 2022.

[51] D. Wang, W. Zhang, X. Han, J. Lin, and Y.-C. Tian, “A multi-objective virtual network migration algorithm based on reinforcement learning,” *IEEE Transactions on Cloud Computing*, 2022.

[52] Z. Yang, R. Gu, H. Li, and Y. Ji, “Approximately lossless model compression-based multilayer virtual network embedding for edge-cloud collaborative services,” *IEEE Internet of Things Journal*, 2023.

[53] E. F. Maleki, L. Mashayekhy, and S. M. Nabavinejad, “Mobility-aware computation offloading in edge computing using machine learning,” *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 328–340, 2021.

[54] J. Wang, J. Hu, G. Min, Q. Ni, and T. El-Ghazawi, “Online service migration in mobile edge with incomplete system information: A deep recurrent actor-critic learning approach,” *IEEE Transactions on Mobile Computing*, vol. 22, no. 11, pp. 6663–6675, 2023.

[55] S. Padhy and J. Chou, “Mirage: A consolidation aware migration avoidance genetic job scheduling algorithm for virtualized data centers,” *Journal of Parallel and Distributed Computing*, vol. 154, pp. 106–118, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731521000575

[56] Z. Tong, J. Wang, Y. Wang, B. Liu, and Q. Li, “Energy and performance-efficient dynamic consolidate vms using deep-q neural network,” *IEEE Transactions on Industrial Informatics*, pp. 1–11, 2023.

[57] M. H. Sayadnavard, A. T. Haghighat, and A. M. Rahmani, “A multi-objective approach for energy-efficient and reliable dynamic vm consolidation in cloud data centers,” *Engineering science and technology, an International Journal*, vol. 26, p. 100995, 2022.

[58] J. Zhu, J. Wang, Y. Zhang, and Y. Jiang, “Virtual machine migration method based on load cognition,” *Soft Computing*, vol. 23, no. 19, pp. 9439–9448, 2019.

[59] H. Zhao, N. Feng, J. Li, G. Zhang, J. Wang, Q. Wang, and B. Wan, “Vm performance-aware virtual machine migration method based on ant colony optimization in cloud environment,” *Journal of Parallel and Distributed Computing*, vol. 176, pp. 17–27, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731523000187

[60] S. E. Motaki, A. Yahyaouy, and H. Gualous, “A prediction-based model for virtual machine live migration monitoring in a cloud datacenter,” *Computing*, vol. 103, no. 11, pp. 2711–2735, 2021.

[61] C. Jo, Y. Cho, and B. Egger, “A machine learning approach to live migration modeling,” in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 351–364.

[62] C. Systems, P. L. D. of Computer Science, and S. N. U. Engineering, “Csap virtual machine live migration dataset,” 2017, last accessed 22 December 2023. [Online]. Available: https://csap.snu.ac.kr/software/lmdataset

[63] M. E. Elsaid, H. M. Abbas, and C. Meinel, “Live migration timing optimization for vmware environments using machine learning techniques.” in *CLOSER*, 2020, pp. 91–102.

[64] M. Duggan, R. Shaw, J. Duggan, E. Howley, and E. Barrett, “A multitime-steps-ahead prediction approach for scheduling live migration in cloud data centers,” *Software: Practice and Experience*, vol. 49, no. 4, pp. 617–639, 2019.

[65] K. Mason, M. Duggan, E. Barrett, J. Duggan, and E. Howley, “Predicting host cpu utilization in the cloud using evolutionary neural networks,” *Future Generation Computer Systems*, vol. 86, pp. 162–173, 2018.

[66] M. Hassan, A. Babiker, M. Amien, and M. Hamad, “Sla management for virtual machine live migration using machine learning with modified kernel and statistical approach,” *Engineering, Technology & Applied Science Research*, vol. 8, no. 1, pp. 2459–2463, 2018.

[67] X. Sui, D. Liu, L. Li, H. Wang, and H. Yang, “Virtual machine scheduling strategy based on machine learning algorithms for load balancing,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, pp. 1–16, 2019.

[68] N. Garg, D. Singh, and M. S. Goraya, “Optimal virtual machine scheduling in virtualized cloud environment using vikor method,” *The Journal of Supercomputing*, pp. 1–29, 2022.

[69] Q. Gao, J. Xiao, Y. Cao, S. Deng, C. Ouyang, and Z. Feng, “Blockchain-based collaborative

edge computing: efficiency, incentive and trust," *Journal of Cloud Computing*, vol. 12, no. 1, p. 72, 2023.

[70] S. Rathod, R. Joshi, S. Gonge, S. Pandya, T. R. Gadekallu, and A. R. Javed, "Blockchain based simulated virtual machine placement hybrid approach for decentralized cloud and edge computing environments," in *Security and Risk Analysis for Intelligent Edge Computing*. Springer, 2023, pp. 223–236.

[71] A. Wilczyński and J. Kołodziej, "Modelling and simulation of security-aware task scheduling in cloud computing based on blockchain technology," *Simulation Modelling Practice and Theory*, vol. 99, p. 102038, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X19301698

[72] S. Lupaiescu, P. Cioata, C. E. Turcu, O. Gherman, C. O. Turcu, and G. Paslaru, "Centralized vs. decentralized: Performance comparison between bigchaindb and amazon qldb," *Applied Sciences*, vol. 13, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/1/499

[73] R. Liu, P. Yang, H. Lv, and W. Li, "Multi-objective multi-factorial evolutionary algorithm for container placement," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1430–1445, 2023.

[74] Y. Mao, W. Yan, Y. Song, Y. Zeng, M. Chen, L. Cheng, and Q. Liu, "Differentiate quality of experience scheduling for deep learning inferences with docker containers in the cloud," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1667–1677, 2023.

[75] L. Deng, Z. Wang, H. Sun, B. Li, and X. Yang, "A deep reinforcement learning-based optimization method for long-running applications container deployment," *International Journal of Computers Communications & Control*, vol. 18, no. 4, 2023.

[76] L. Zhu, K. Huang, K. Fu, Y. Hu, and Y. Wang, "A priority-aware scheduling framework for heterogeneous workloads in container-based cloud," *Applied Intelligence*, vol. 53, no. 12, p. 15222–15245, nov 2022. [Online]. Available: https://doi.org/10.1007/s10489-022-04164-1

[77] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. Leung, “Tailored learning-based scheduling for kubernetes-oriented edge-cloud system,” in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[78] A. Katal, T. Choudhury, and S. Dahiya, “Energy optimized container placement for cloud data centers: a meta-heuristic approach,” *The Journal of Supercomputing*, pp. 1–43, 2023.

[79] A. Bouaouda, K. Afdel, and R. Abounacer, “Forecasting the energy consumption of cloud data centers based on container placement with ant colony optimization and bin packing,” in *2022 5th Conference on Cloud and Internet of Things (CIoT)*, 2022, pp. 150–157.

[80] S. Long, W. Wen, Z. Li, K. Li, R. Yu, and J. Zhu, “A global cost-aware container scheduling strategy in cloud data centers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2752–2766, 2021.

[81] Y. Chen, S. He, X. Jin, Z. Wang, F. Wang, and L. Chen, “Resource utilization and cost optimization oriented container placement for edge computing in industrial internet,” *The Journal of Supercomputing*, vol. 79, no. 4, pp. 3821–3849, 2023.

[82] W. Zhang, L. Chen, J. Luo, and J. Liu, “A two-stage container management in the cloud for optimizing the load balancing and migration cost,” *Future Generation Computer Systems*, vol. 135, pp. 303–314, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22001674

[83] N. Zhou, F. Dufour, V. Bode, P. Zinterhof, N. J. Hammer, and D. Kranzlmüller, “Towards confidential computing: A secure cloud architecture for big data analytics and ai,” *arXiv preprint arXiv:2305.17761*, 2023.

[84] H. Song, J. Li, and H. Li, “A cloud secure storage mechanism based on data dispersion and encryption,” *IEEE Access*, vol. 9, pp. 63 745–63 751, 2021.

[85] J. Gabriel, S. Ankermann, M. Seidel, and F. H. Fitzek, “Transparent storage encryption in kubernetes,” in *European Wireless 2022; 27th European Wireless Conference*. VDE, 2022, pp. 1–6.

[86] Q. Deng, X. Tan, J. Yang, C. Zheng, L. Wang, and Z. Xu, “A secure container placement strategy using deep reinforcement learning in cloud,” in *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2022, pp. 1299–1304.

[87] T. Kong, L. Wang, D. Ma, Z. Xu, Q. Yang, and K. Chen, “A secure container deployment strategy by genetic algorithm to defend against co-resident attacks in cloud computing,” in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 1825–1832.

[88] Z. Lei, E. Sun, S. Chen, J. Wu, and W. Shen, “A novel hybrid-copy algorithm for live migration of virtual machine,” *Future Internet*, vol. 9, no. 3, p. 37, 2017.

[89] A. Beloglazov and R. Buyya, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers,” *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[90] Prometheus, “Prometheus - monitoring system & time series database,” 2024, last accessed 20 Jun 2024. [Online]. Available: https://prometheus.io/

[91] G. Labs, “Grafana: The open observability platform — grafana labs,” 2024, last accessed 20 Jun 2024. [Online]. Available: https://grafana.com/

[92] OpenStack, “Compute api - nova documentation,” 2023, last accessed 20 Jun 2024. [Online]. Available: https://docs.openstack.org/api-ref/compute/

[93] ——, “Open source cloud computing platform software - openstack,” 2024, last accessed 20 Jun 2024. [Online]. Available: https://www.openstack.org/software/

[94] G. Liu, J. Li, and J. Xu, “An improved min-min algorithm in cloud computing,” in *Proceedings of the 2012 International Conference of Modern Computer Science and Applications*. Springer, 2013, pp. 47–52.

[95] D. Gritto and P. Muthulakshmi, "Scheduling cloudlets in a cloud computing environment: A priority-based cloudlet scheduling algorithm (pbcsa)," in *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*. IEEE, 2022, pp. 80–86.

[96] Z. Liu and X. Wang, "A pso-based algorithm for load balancing in virtual machines of cloud computing environment," in *Advances in Swarm Intelligence: Third International Conference, ICSI 2012, Shenzhen, China, June 17-20, 2012 Proceedings, Part I 3*. Springer, 2012, pp. 142–147.

[97] C.-C. Lin, P. Liu, and J.-J. Wu, "Energy-aware virtual machine dynamic provision and scheduling for cloud computing," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 736–737.

[98] V. Polepally and K. Shahu Chatrapati, "Dragonfly optimization and constraint measure-based load balancing in cloud computing," *Cluster Computing*, vol. 22, no. Suppl 1, pp. 1099–1111, 2019.

[99] N. Alaei and F. Safi-Esfahani, "Repro-active: a reactive–proactive scheduling method based on simulation in cloud computing," *The Journal of Supercomputing*, vol. 74, no. 2, pp. 801–829, 2018.

[100] S. Supreeth, K. Patil, S. D. Patil, and S. Rohith, "Comparative approach for vm scheduling using modified particle swarm optimization and genetic algorithm in cloud computing," in *2022 IEEE International Conference on Data Science and Information System (ICDSIS)*. IEEE, 2022, pp. 1–6.

[101] Z. Xiao, X. Liu, and Z. Ming, "A deep reinforcement learning based vm scheduling strategy decreasing data center communication costs," in *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2022, pp. 1173–1179.

[102] S. Kamat, S. Naik, S. Kanamadi, S. Alur, D. Narayan, and S. Patil, "Compute and network

aware vm scheduling using reinforcement learning in cloud," in *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*. IEEE, 2023, pp. 1–7.

[103] I. Bhandary, K. Atul, A. Athani, S. Patil, and D. Narayan, "Energy-efficient vm scheduling in the cloud environment using reinforcement learning," in *2021 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*. IEEE, 2021, pp. 46–51.

[104] X. Ma, H. Xu, H. Gao, M. Bian, and W. Hussain, "Real-time virtual machine scheduling in industry iot network: a reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 2129–2139, 2022.

[105] T. Daradkeh, A. Agarwal, N. Goel, and A. Kozlowski, "Point estimator log tracker for cloud monitoring," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 62–67.

[106] Z. Lee, Y. Wang, and W. Zhou, "A dynamic priority scheduling algorithm on service request scheduling in cloud computing," in *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*, vol. 9, 2011, pp. 4665–4669.

[107] S. Network and M. HPC Group Gina Cody School, Concordia University, "Speed hpc facility," 2023, last accessed July 2023. [Online]. Available: https://github.com/NAG-DevOps/speed-hpc

[108] T. C. S. Ltd., "Wanem the wide area network emulator," 2014, last accessed April 2023. [Online]. Available: https://wanem.sourceforge.net

[109] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers, 6th Edition*. Wiley, 2013.

[110] M. A. Altahat, A. Agarwal, N. Goel, and M. Zaman, "Neural network based regression model for virtual machines migration method selection," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2021, pp. 1–6.

[111] F. Bellard, “Qemu, a fast and portable dynamic translator.” in *USENIX annual technical conference, FREENIX Track*, vol. 41, no. 46. California, USA, 2005, pp. 10–5555.

[112] QEMU, “Qemu a generic and open source machine emulator and virtualizer,” 2023, last accessed 4 April 2024. [Online]. Available: https://www.qemu.org/

[113] J. Tang, T. Qin, Y. Xiang, Z. Zhou, and J. Gu, “Optimization search strategy for task offloading from collaborative edge computing,” *IEEE Transactions on Services Computing*, vol. 16, no. 3, pp. 2044–2058, 2023.

[114] I. Attiya, M. A. Elaziz, L. Abualigah, T. N. Nguyen, and A. A. A. El-Latif, “An improved hybrid swarm intelligence for scheduling iot application tasks in the cloud,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6264–6272, 2022.

[115] A. Satpathy, M. N. Sahoo, A. Mishra, B. Majhi, J. J. Rodrigues, and S. Bakshi, “A service sustainable live migration strategy for multiple virtual machines in cloud data centers,” *Big Data Research*, vol. 25, p. 100213, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214579621000307

[116] J. Tang, M. M. Jalalzai, C. Feng, Z. Xiong, and Y. Zhang, “Latency-aware task scheduling in software-defined edge and cloud computing with erasure-coded storage systems,” *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1575–1590, 2023.

[117] M. Maray, E. Mustafa, J. Shuja, and M. Bilal, “Dependent task offloading with deadline-aware scheduling in mobile edge networks,” *Internet of Things*, vol. 23, p. 100868, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660523001919

[118] J. Bi, K. Zhang, H. Yuan, and J. Zhang, “Energy-efficient computation offloading for static and dynamic applications in hybrid mobile edge cloud system,” *IEEE Transactions on Sustainable Computing*, vol. 8, no. 2, pp. 232–244, 2023.

[119] R. W. Ahmad, A. Gani, S. H. Ab. Hamid, M. Shiraz, F. Xia, and S. A. Madani, “Virtual machine migration in cloud data centers: A review, taxonomy, and open research

issues," *J. Supercomput.*, vol. 71, no. 7, p. 2473–2515, jul 2015. [Online]. Available: https://doi.org/10.1007/s11227-015-1400-5

[120] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões, "Container-based virtualization for real-time industrial systems—a systematic review," *ACM Comput. Surv.*, vol. 56, no. 3, oct 2023. [Online]. Available: https://doi.org/10.1145/3617591

[121] R. Gupta, P. Kanungo, and N. Dagdee, "A survey of state-of-the-art multi-authority attribute based encryption schemes in cloud environment." *KSII Transactions on Internet & Information Systems*, vol. 17, no. 1, 2023.

[122] R. Imam, K. Kumar, S. M. Raza, R. Sadaf, F. Anwer, N. Fatima, M. Nadeem, M. Abbas, and O. Rahman, "A systematic literature review of attribute based encryption in health services," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 6743–6774, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157822002269

[123] P. S. K. Oberko, V.-H. K. S. Obeng, and H. Xiong, "A survey on multi-authority and decentralized attribute-based encryption," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–19, 2022.

[124] M. A. Altahat, T. Daradkeh, and A. Agarwal, "Optimized encryption-integrated strategy for containers scheduling and secure migration in multi-cloud data centers," *IEEE Access*, vol. 12, pp. 51 330–51 345, 2024.

[125] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012, special Section: Energy efficiency in large-scale distributed systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X11000689

[126] J. Luo, Q. Liu, Y. Yang, X. Li, M. rong Chen, and W. Cao, "An artificial bee colony algorithm for multi-objective optimisation," *Applied Soft Computing*, vol. 50, pp. 235–251, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156849461630583X

[127] PyPI, “Pure python rsa implementation,” 2022, last accessed December 2023. [Online]. Available: https://pypi.org/project/rsa/

[128] B. Kaliski, “Rfc2313: Pkcs# 1: Rsa encryption version 1.5,” 1998, last accessed December 2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2313

[129] PyCryptodome, “Aes - pycryptodome 3.210b0 documentation,” 2022, last accessed December 2023. [Online]. Available: https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

[130] P. S. Foundation, “secrets — generate secure random numbers for managing secrets,” 2022, last accessed December 2023. [Online]. Available: https://docs.python.org/3/library/secrets.html

[131] PyPI, “tinyec 0.4.0,” 2021, last accessed December 2023. [Online]. Available: https://pypi.org/project/tinyec/

[132] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, “Borg: the next generation,” in *Proceedings of the fifteenth European conference on computer systems*, 2020, pp. 1–14.

[133] Google, “Borg cluster traces from google,” 2020, last accessed 20 December 2023. [Online]. Available: https://github.com/google/cluster-data/tree/master

[134] G. OR-Tools, “Cp-sat solver,” 2023, last accessed November 2023. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver