Some Studies on Renewable Energy-Based Scheduling Algorithms for Geo-Distributed Datacenters

Submitted in partial fulfilment of the requirements of the degree of

Doctor of Philosophy

by

SLOKASHREE PADHI

(Roll No.: 21CSRES01)

Under the supervision of

Prof. R. B. V. Subramanyam

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
WARANGAL - 506004, TELANGANA, INDIA
AUGUST 2024

Dedicated to

My Grandmother

Thank You for Your Love and Support

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
WARANGAL - 506004, TELANGANA, INDIA

THESIS APPROVAL SHEET FOR PH. D.

This dissertation work entitled, **Some Studies on Renewable Energy-Based Scheduling Algorithms for Geo-Distributed Datacenters**, by **Mrs. Slokashree Padhi (Roll No.: 21CSRES01)** is approved for the degree of **Doctor of Philosophy** at the National Institute of Technology Warangal.

Examiners

Supervisor
Prof. R. B. V. Subramanyam
Dept. of Computer Science and Engg.
NIT Warangal, India

Chairman
Prof. P. Radha Krishna
Dept. of Computer Science and Engg.
NIT Warangal, India

Date: 20/08/2024

Place: NIT Warangal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
WARANGAL - 506004, TELANGANA, INDIA

CERTIFICATE

This is to certify that the thesis entitled, **Some Studies on Renewable Energy-Based Scheduling Algorithms for Geo-Distributed Datacenters**, submitted in partial fulfilment of requirements for the award of the degree of **Doctor of Philosophy** to the National Institute of Technology Warangal, is a bonafide research work done by **Mrs. Slokashree Padhi (Roll No.: 21CSRES01)** under my supervision. The contents of the thesis have not been submitted elsewhere for the award of any degree.

Prof. R. B. V. Subramanyam
Professor and Supervisor
Department of Computer Science and Engg.
NIT Warangal, India

Place: NIT Warangal
Date: 20/08/2024

DECLARATION

This is to certify that the work presented in the thesis entitled "**Some Studies on Renewable Energy-Based Scheduling Algorithms for Geo-Distributed Datacenters**" is a bonafide work done by me under the supervision of Prof. R. B. V. Subramanyam and was not submitted elsewhere for the award of any degree.

I declare that this written submission represents my ideas in my own words, and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all academic honesty and integrity principles and have not misrepresented, fabricated, or falsified any idea/date/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Mrs. Slokashree Padhi
Roll No.: 21CSRES01

Date: 20/08/2024

Acknowledgements

First and foremost, I extend my heartfelt thanks to my supervisor, Prof. R. B. V. Subramanyam, whose invaluable insights, encouragement, and unwavering support played a pivotal role in shaping this research. His guidance has been instrumental in every step of the process, and I am genuinely grateful for the knowledge and wisdom he shared. I would like to express my gratitude to the Doctoral Scrutiny Committee (DSC) members, Prof. P. Radha Krishna, Prof. Ch. Sudhakar, Prof. S. Ravichandra and Prof. D. Srinivasacharya, for their constructive comments and scholarly input. I would like to thank my DSC external members, Prof. K. Ramesh and Prof. B. Srinivasa Rao, for their invaluable suggestions during the comprehensive examination.

I am indebted to Prof. Bidyadhar Subudhi, Director; Prof. A. Sarath Babu, Dean (Academic); Prof. Sonawane Shirish Hari, Dean (Research and Development); Prof. D. Srinivasacharya, Dean (Student Welfare) and Prof. N. V. Umamahesh, Registrar In-Charge, for their support, providing me with all necessary facilities during my candidature. I am immensely thankful to Prof. P. Radha Krishna, Prof. S. Ravichandra, and Prof. R. Padmavathy, Heads of the Department of Computer Science and Engineering (CSE) at NIT Warangal, India, for providing adequate facilities during my stay in the department. I would like to thank all the faculty members and staff of the Department of CSE for their help and support.

I would like to thank the anonymous examiners (i.e., Indian and foreign) for their valuable comments, future research directions and timely submission of the report.

Special thanks go to my family for their unwavering support and understanding. I thank my sister, Ms. Shyamashree Padhi, who has always motivated and encouraged me to achieve good things. I thank my parents, who have given me all kinds of support throughout this Ph. D. period. I thank my husband for backing me in every endeavour. Their love and encouragement sustained me during the challenging phases of this academic endeavour. I am grateful for the sacrifices you made to ensure my success.

I am indebted to all my seniors, friends, and juniors, Dr. Sohan Kumar Pande, Dr.

Sanjib Kumar Nayak, Mr. Sanjib Kumar Raul, Ms. Rachana Behera, Dr. Thanedar Md. Asif, Ms. U. Shivani Sri Varshini, Mrs. P. Navya, and Mrs. M. Sarika, who provided valuable insights and encouragement, making the academic journey more enjoyable.

Place: NIT Warangal
Date: 20/08/2024

Mrs. Slokashree Padhi
Roll No.: 21CSRES01

Abstract

The cloud marketplace is continuously rising as enterprises desire to streamline their processes. The marketplace can address the potential computation, storage, databases, and bandwidth needs across various sectors, including business, industry, manufacturing, entertainment, government, education, agriculture, banking and smart cities. As adaptability increases, cloud service providers (CSPs) expand their datacenters to handle requests of any size. It increases the fossil fuels consumed in each datacenter, increasing the overall cost. Therefore, many CSPs have adopted renewable energy (RE) sources to increase profitability and reduce carbon emissions. However, RE generation fluctuates with time, location and climate conditions, which introduces uncertainty (UN) in fulfilling user requests (URs). Thus, non-renewable energy (NRE) generation continues to power the datacenters to make stability. Recent works are directed to the use of RE generation followed by NRE generation while assigning the URs to the resources of the datacenters. However, they present the requirements of URs using the processor nodes without considering memory nodes. Moreover, these works aim to maximize the usage of RE or minimize the cost and do not model the UN of RE and NRE resources and the level of UN (UNL). Furthermore, the URs may be redirected from one datacenter to another depending on the presence of RE resources to minimize the cost. However, systematically planning of datacenter migration to move URs is quite challenging.

This thesis mainly addresses the UR-based scheduling problems in geo-distributed datacenters. It presents several algorithms for these problems, considering processor and memory nodes, UN, UNL, and migration. Firstly, we present two algorithms, processor and memory-based future-aware best fit (PM-FABEF) and processor and memory-based highest available renewable first (PM-HAREF), that incorporate both processor and memory nodes for geographical load balancing (GLB). PM-FABEF determines the cost of processor and memory nodes for assigning URs to the datacenters and assigns them to the least cost datacenter. PM-HAREF determines the highest RE resource slots in processor and memory for assigning the URs. Secondly, we present three UR-based scheduling algorithms, namely UN-based future-aware best fit (UN-FABEF), UN-based highest available

renewable first (UN-HAREF) and UN-based round-robin (UN-RR), by managing the UN of RE and NRE. These algorithms consider two types of UN, namely UN of RE (UN-RE) and UN of NRE (UN-NRE) resources, concerning the UR. UN-FABEF matches the UR to all the available datacenters and determines the assignment cost, including UN time duration. Then, it assigns that UR to a datacenter that results in the least assignment cost. UN-HAREF matches the UR to all the available datacenters and determines the number of RE resource slots, including UN time duration. Then, it assigns that UR to a datacenter that results in the highest number of RE resource slots. On the contrary, UN-RR assigns the URs to the datacenters in a circular fashion. Thirdly, we extend the three benchmark algorithms, namely FABEF, HAREF and RR, by incorporating UN and UNL, and we call them UNL-based future-aware best fit (UNL-FABEF), UNL-based highest available renewable first (UNL-HAREF) and UNL-based round-robin (UNL-RR), respectively. The goal of UNL-FABEF is to minimize the overall cost, whereas UNL-HAREF is to maximize the available RE usage. On the contrary, UNL-RR assigns the URs to the datacenters in a roundabout fashion. Then, we introduce the UNL-based multi-objective scheduling algorithm (UNL-MOSA) to make a trade-off between UNL-FABEF and UNL-HAREF. UNL-MOSA creates a balance between the overall cost and the available RE usage. Lastly, we introduce a novel RE-oriented migration algorithm (REOMA) to minimize the total cost of geo-distributed datacenters through strategic migration between datacenters. REOMA finds the cost based on the time window of the UR in each datacenter. Then, it fits the cost of each datacenter into a polynomial curve based on the time window and determines the slope and intercept. Subsequently, it finds the migration points between the datacenter with the lowest cost and the other datacenters and performs the migration between a pair of datacenters that results in the lowest cost.

We perform rigorous simulations on the proposed algorithms and measure their performance in terms of various performance metrics, namely overall cost (OCO), the total number of used RE resource slots (TNRE), the total number of used NRE resource slots (TNNRE), the UN time (UNT) and the UN cost (UNCO). The proposed algorithms are compared with three benchmark algorithms using fifty instances of ten datasets with 200 to 2000 URs and 20 to 200 datacenters based on their applicability. For comparison purposes,

we also evaluate the simulation results against existing scheduling algorithms and present the findings in various tabular and graphical formats. The comparison results demonstrate that the proposed scheduling algorithms outperform existing ones in terms of the aforementioned performance metrics. Furthermore, we validate these results using analysis of variance (ANOVA) statistical tests based on the applicability.

Keywords: Cloud Computing, Cloud Service Provider, Datacenter, Geo-Distributed Datacenters, Geographical Load Balancing, Non-Renewable Energy, Overall Cost, Renewable Energy, Scheduling, Uncertainty, Uncertainty Level, User Request.

Contents

List of Tables

List of Figures

List of Abbreviations and Notations

List of Abbreviations

Acronym	Abbreviation
1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
ACO	Assignment Cost
Ah	Ampere hours
ANOVA	ANalysis Of VAriance
AT	Arrival Time
AWS	Amazon Web Service
Bi-GRU	Bidirectional - Gated Recurrent Unit
Bi-LSTM	Bidirectional - Long Short-Term Memory
C/R	Checkpoint/Restore
CAGR	Compound Annual Growth Rate
CNN-GRU	Convolutional Neural Networks - Gated Recurrent Unit
CO_2	Carbon diOxide
COVID-19	COronaVIrus Disease of 2019
CPM	Constrained Power Management
CPU	Central Processing Unit
CRES	Cost-aware Renewable Energy Scheduling
CSPs	Cloud Service Providers

D	Duration
DF	Degree of Freedom
EC2	Elastic Compute Cloud
FABEF	Future-Aware BEst Fit
FC	F Critical
FLB	Fuzzy Logic-Based Load Balancing
g/kWh	grams per kiloWatt Hour
GB	GigaByte
GCC	Green Cloud Computing
GCP	Google Cloud Platform
GHz	GigaHertz
GLB	Geographical Load Balancing
GRU	Gated Recurrent Unit
HAREF	Highest Available REnewable First
HP	Hewlett-Packard
HPC	High-Performance Computing
IaaS	Infrastructure as a Service
IBM	International Business Machines
ID	IDentification number
IEA	International Energy Agency
IEEE	Institute of Electrical and Electronics Engineers
ILP	Integer Linear Programming
IoT	Internet of Things
kW	kiloWatt
kWh	kiloWatt Hour
LSTM	Long Short-Term Memory
MATLAB	MATrix LABoratory
MB	MinBrown
MCC	Mobile Cloud Computing
MinUtil	Minimum Utilization

MIPS	Million Instructions Per Second
MORE	Multi-Objective Renewable Energy
MS	Mean Square
N	Number of nodes
NM	Number of required Memory nodes
NP	Number of required Processor nodes
NRE	Non-Renewable Energy
OCO	Overall COst
PaaS	Platform as a Service
PM	Processor and Memory-based
PM-FABEF	Processor and Memory-based Future-Aware BEst Fit
PM-HAREF	Processor and Memory-based Highest Available REnewable First
PUE	Power Usage Effectiveness
PV	Photovoltaic
QoS	Quality of Service
RAM	Random-Access Memory
RE	Renewable Energy
REOMA	Renewable Energy-Oriented Migration Algorithm
RR	Round Robin
SaaS	Software as a Service
SCA	Static Cost-Aware ordering
SLA	Service-Level Agreement
SLO	Service-Level Objectives
SS	Sum of Square
ST	Start Time
SV	Sources of Variations
SVR	Support Vector Regression
TNNRE	Total Number of used Non-Renewable Energy resource slots
TNRE	Total Number of used Renewable Energy resource slots
UN	UNcertainty

UN-FABEF	UNcertainty-based Future-Aware BEst Fit
UN-HAREF	UNcertainty-based Highest Available REnewable First
UN-NRE	UNcertainty of Non-Renewable Energy
UN-RE	UNcertainty of Renewable Energy
UN-RR	UNcertainty-based Round Robin
UNCO	UNcertainty COst
UNL	UNcertainty Level
UNL-FABEF	UNcertainty Level-based Future-Aware BEst Fit
UNL-HAREF	UNcertainty Level-based Highest Available REnewable First
UNL-MOSA	UNcertainty Level-based Multi-Objective Scheduling Algorithm
UNL-RR	UNcertainty Level-based Round Robin
UNT	UNcertainty Time
UPM	Unconstrained Power Management
URQ	User Request Queue
URs	User Requests
US	United States
vCPU	virtual Central Processing Unit
VMs	Virtual Machines
WF	Worst Fit

List of Notations

Notation	**Description**
τ_1	Threshold of level 1 uncertainty
τ_2	Threshold of level 2 uncertainty
τ_3	Threshold of level 3 uncertainty
ACO	Assignment cost of renewable energy and non-renewable energy resources
$ACOM$	Assignment cost of memory nodes

$ACOP$	Assignment cost of processor nodes
$ANRE$	Available non-renewable energy resources
ARE	Available renewable energy resources
CO	A cost series of renewable energy and non-renewable energy resources over time
$COST$	Cumulative cost
d	Maximum duration
$D[k]$	Duration of k^{th} user request or user request identification number k
DC_i	i^{th} datacenter or Datacenter identification number i
DCE	Available renewable energy slots in datacenter
DCP	Cost of non-renewable energy slot in datacenter
$FMIN\_DC$	Final minimum cost datacenter
i	A loop variable to keep the datacenter identification number
i'	A index to keep the best datacenter
k	A loop variable to keep the user request identification number
l	A loop variable to check the resources between start time and the sum of start time and duration - 1
m	Number of datacenters
M	A two-dimensional matrix to know the availability of memory nodes
max	A pre-defined function/procedure to find the greatest value
$MCOST\_DC$	Minimum cost datacenter
$MCOST$	Minimum cost
mem_cost_re	A cost series of renewable energy resources over time for memory
mem_cost_nre	A cost series of non-renewable energy resources over time for memory
mem_flag	Memory flag
mem_slots	Number of memory slots
mem_slots_re	Number of memory slots that are using renewable energy
mem_slots_nre	Number of memory slots that are using non-renewable energy
MIG_CHK	Migration check
min	A pre-defined function/procedure to find the least value

MIN_VAL	Minimum value
MP	Migration point
n	Number of user requests
$NM[k]$	Number of requried memory nodes for k^{th} user request or user request identification number k
$NP[k]$	Number of requried processor nodes for k^{th} user request or user request identification number k
nre	Number of non-renewable energy resources
o	Number of resources per datacenter
P	A two-dimensional matrix to know the availability of processor nodes
$POLYFIT$	A function to determine the slope and intercept
$proc\_cost\_re$	A cost series of renewable energy resources over time for processor
$proc\_cost\_nre$	A cost series of non-renewable energy resources over time for processor
$proc\_flag$	Processor flag
$proc\_slots$	Number of processor slots
$proc\_slots\_re$	Number of processor slots that are using renewable energy
$proc\_slots\_nre$	Number of processor slots that are using non-renewable energy
R	A two-dimensional matrix to check whether the resource is assigned or unassigned
re	Number of renewable energy resources
SEL_DC	Selected datacenter
$slots\_nre$	Number of non-renewable energy resource slots
$slots\_re$	Number of renewable energy resource slots
$ST[k]$	Start time of k^{th} user request or user request identification number k
$TCOST$	Temporary cost
$UN\text{-}COST[i]$	Uncertainty cost of datacenter i
$UN\text{-}NRE$	UN of non-renewable energy resources
$UN\text{-}RE$	UN of renewable energy resources
$UN\text{-}TIME[i]$	Uncertainty time of datacenter i
$UNL\text{-}R$	Uncertainty level of resource

$UNL\text{-}RE$	Uncertainty level of renewable energy resource required for user request
UR_k	k^{th} user request or User request identification number k
$XVAL$	X value
$YVAL$	Y value

Chapter 1

Introduction

The global cloud computing market has a tremendous upward trajectory and has transformed our lives and livelihoods [1–4]. The market provides a platform for enterprises to buy services from cloud service providers (CSPs) over the Internet [5–7]. It reforms how to conduct business, develop applications, and configure infrastructure [8–11]. It enables enterprises to compete in the market without concern about infrastructure cost, application deployment, computation, storage, databases, security and recovery [12–15]. As per Cloudwards [16], 94% of enterprises rely on various cloud services. The enterprises also streamline their processes by adopting various services, namely infrastructure (IaaS), platform (PaaS) and software (SaaS) [1, 17–19]. As per the survey conducted by Centrify and CensusWide, 43% of organizations were reluctant to migrate to the cloud before the COronaVIrus Disease of 2019 (COVID-19) pandemic in March 2020 [20]. The rationality behind this reluctance is security, compliance, migration plan or approach, availability, cost and stability. However, most organizations have streamlined their migration plan and digitized their process through cloud technology after the pandemic. Therefore, the cloud computing market's compound annual growth rate (CAGR) is expected to reach 17.5% and be worth around $832.1 billion by the end of 2025 [16]. More specifically, the CAGR of the cloud computing storage market from 2021 to 2028 is expected to reach 26.2% and be worth around $390 billion by the end of 2028 [16]. Cloud infrastructure provides creative solutions to any scale enterprise, from small to large, and become one of the game-changing technologies in today's era. In general, the solutions are provided by deploying

applications in virtual machines (VMs), which take the resources from the servers of the datacenters [21–28]. CSPs deliver on-demand services to fulfil enterprise requirements and charge them as per the pay-as-you-go model [2]. As the demand increases daily, CSPs are constantly looking to escalate hardware and software infrastructures and their efficiency to accommodate any scale requirements without hindrance. For instance, as per Gartner Incorporated, a research and consulting firm in the United States (US), user spending on cloud services is estimated to reach $678.8 billion in 2024, which was $563.6 billion in 2023, with a CAGR of 20.4% [29].

The overwhelming growth of the cloud market opens up new challenges to CSPs. The challenges include infrastructure management, energy consumption, cost, security, customization, vendor lock-in, expanded service availability, and skilled engineers [30–41]. Power management is one of the biggest challenges, as it directly impacts the cost, environment, and outages. For instance, the average power usage effectiveness (PUE) of large datacenters is approximately 1.58 as per the survey conducted in 2023 [42]. Most of the datacenters hosted by CSPs primarily depend on the electric grid. The uninterrupted supply of electricity will ensure the timely delivery of cloud services. Still, the CSPs maintain backup power systems to run their datacenters without interruption. The primary source of electricity and backup power systems is fossil fuels (i.e., coal and its products, gas and its variants (natural and derived), petroleum, etc.) [43, 44]. However, these sources are unsustainable and harmful to our physical environments, such as climate change, water pollution, and air pollution. They are formed by decomposing animals and plants. As the cost of these fuels increases daily and their availability is delimited, many CSPs have been planning to minimize their carbon footprints, water usage, electricity usage, and so on. The tiny changes and progress can save significant power and costs [45]. One of the possible solutions is to adopt renewable energy (RE) sources, namely sunlight/solar, water/hydropower, wind, biomass, geothermal and tidal [46–51]. For instance, the average cost of hydroelectric power, solar, wind, and biomass is $0.05 per kilowatt hour (kWh), $0.10/kWh, $0.13/kWh and $0.10/kWh, respectively, as per Forbes, a business magazine in the US [52]. On the other hand, Google claims that it is the largest buyer of RE and has attained 100% RE in its datacenters since 2017 [53]. Microsoft has also powered its

datacenter using 100% RE since 2014 [53]. Amazon has announced that it has achieved 85% RE and is committed to attaining 100% by 2025 [53]. These sources are sustainable and not harmful to the environment. However, CSPs continue to use fossil fuels, as RE sources are affected by weather conditions, locations and storage. Therefore, recent literature (especially future-aware best fit (FABEF) [46], static cost-aware ordering (SCA) [54], round-robin (RR) [46] and highest available renewable first (HAREF) [46], MinBrown (MB) [55]) focuses on RE sources for managing the resources of datacenters without detaching the traditional way of powering datacenters using non-renewable energy (NRE) sources. However, these studies focus on minimizing the cost or maximizing the usage of RE without considering both processors and memory nodes while assigning the user requests (URs) and assume that the URs execute smoothly without interruption. Note that Amazon's popular service, called elastic compute cloud (EC2), considers both processor and memory while deploying an instance. Moreover, the studies have not emphasized on the uncertainty (UN). Here, UN refers to the delay in executing URs due to internal and external factors [56]. Some examples of these factors include network failure, VM interruption, checkpointing, overbooking, economics, load structure, human resources, power supply, air conditioning, and many more [57]. CSPs need to adopt several precautions to deal with these UN factors. Otherwise, the UN will violate the service-level agreement (SLA) and cause customer dissatisfaction. On the other hand, the URs can be migrated from one datacenter to another to fulfil the requirements.

The rest of this chapter is organized as follows. In Section 1.1, the motivation behind the proposed works and four objectives are discussed. In Section 1.2, the overview of the four contributions of the thesis is highlighted. In Section 1.3, the organization of the thesis with seven chapters is presented.

1.1 Motivation and Objectives

A datacenter is a centralized facility for building, processing and storing applications, services and data [1, 58, 59]. They are owned by CSPs, wholesale operators and retail firms [60]. These owners seek ways to improve energy efficiency and reduce the data-

center operating costs [31]. Moreover, they make a power purchase agreement with the electricity providers [61]. On the other hand, the commercial sector is the largest electricity consumer globally, with Alphabet (Google), Microsoft, and Facebook at the top of the list [62]. Therefore, energy is shifted from non-renewable to renewable sources to power the datacenters. This shift not only helps to safeguard our environment but also decarbonizes datacenters. RE is generated from natural sources: biomass, geothermal, hydro, ocean, sunlight, tidal and wind [33,46]. They are being replenished significantly more than their use. The carbon dioxide (CO_2) emissions of RE generation are much less than those of NRE generation. Therefore, RE is a changer in the climate crisis [63, 64]. According to the International energy agency (IEA), 35% of global power will be generated using RE by 2025 [65]. Even though RE has numerous advantages, it results in low efficiency, high initial cost and substantial space requirements. Moreover, its generation depends on time and location. For instance, CSPs can use solar energy in those locations (like Norway, Iceland, Finland, etc.) where sunlight is available almost around the clock [66]. Similarly, they can use wind energy in windy areas like New Zealand, France, Canada, etc. Still, RE generation is inconsistent and unpredictable in its current state [43, 67, 68]. As a result, CSPs power their datacenters using RE and NRE generation. When RE generation cannot fulfil the power requirements of datacenters, CSPs switch to NRE generation to meet such requirements [67]. However, no datacenter has an unlimited power supply, processor and memory nodes. Therefore, each datacenter encounters some unavoidable circumstances concerning its resources. These circumstances can be considered UN while assigning the URs to such resources. As the RE generation is more unstable than the NRE generation, the UN of the RE generation is relatively higher than the NRE generation. Recent literature focuses on both RE and NRE generation and makes their proper usage while assigning the URs to the datacenters without looking into the UN of resources and UN level (UNL) [46, 57, 63, 64, 69–71]. This fact motivates us to develop RE-based scheduling algorithms by incorporating processor and memory nodes, UN and UNL, and migration between datacenters, which are not well-investigated in the literature.

The following objectives are formulated concerning the above-mentioned motivation.

Objective 1: To minimize the cost of assigning URs to the resources of datacenters by

incorporating processor and memory nodes of datacenters.

Objective 2: To maximize the usage of RE resources of datacenters by managing the UN of RE and NRE resources while assigning the URs.

Objective 3: To minimize the overall cost and maximize the usage of RE resources of datacenters by incorporating UN of RE and NRE resources and their level while assigning the URs.

Objective 4: To find the migration points of the URs to facilitate their relocation among datacenters and minimize overall costs.

In this thesis, we present RE-based scheduling algorithms for geo-distributed datacenters. First, we present RE-based algorithms by incorporating processor and memory nodes of datacenters to minimize the cost of assigning URs. Then, we consider the UN of RE and NRE resources and present RE-based algorithms to maximize the usage of RE resources for assigning URs. Subsequently, we model the UN from the user and CSP perspectives and propose RE-based algorithms to minimize the cost and maximize the usage of RE resources for assigning URs. Finally, we present a RE-oriented migration algorithm to migrate the URs among the datacenters to minimize the overall costs. Throughout the thesis, we interchangeably use brown (or green) energy and non-renewable (renewable) energy. Similarly, we use nodes and slots interchangeably.

1.2 Overview of the Contributions of the Thesis

In this section, an overview of the chapter-wise contributions of the thesis is presented.

1.2.1 Efficient Renewable Energy-Based Geographical Load Balancing Algorithms

As the adaptability of the cloud increases, CSPs expand their datacenters to handle any UR size. It increases the fossil fuels consumed in each datacenter, increasing the overall cost (OCO). Therefore, CSPs are looking for economical ways to reduce fossil fuels. Consequently, three benchmark algorithms were developed in the literature for geographical load

balancing (GLB) using RE sources. However, they present the UR using the processor nodes without considering memory nodes. This work presents two algorithms, processor and memory-based future-aware best fit (PM-FABEF) and processor and memory-based highest available renewable first (PM-HAREF), for green cloud computing (GCC) that incorporates processor and memory nodes. PM-FABEF determines the cost of processor and memory nodes for assigning URs to the datacenters and assigns them to the least cost datacenter. PM-HAREF determines the highest RE resource slots in processor and memory for assigning the URs. The proposed algorithms are compared with three algorithms using ten datasets to show their superiority in three performance metrics, namely the OCO, the total number of used RE resource slots (TNRE) and the total number of used NRE resource slots (TNNRE). The primary novelties of this work are as follows.

1. We develop two GLB algorithms by incorporating processor and memory nodes at a glance.

2. We consider the varying costs with respect to processor and memory nodes in each datacenter to make the proposed algorithms more realistic.

3. The proposed algorithms are compared with three benchmark algorithms to show their efficacy in three performance metrics.

1.2.2 User Request-Based Scheduling Algorithms by Managing Uncertainty of Renewable Energy

Many CSPs have adopted RE sources to increase profitability and reduce carbon emissions. Recent literature focuses on managing cloud infrastructure with RE sources in addition to traditional NRE sources whenever required. However, these works aim to maximize the usage of RE or minimize the cost without considering the UN. This work presents three UR-based scheduling algorithms, namely UN-based FABEF (UN-FABEF), UN-based HAREF (UN-HAREF), and UN-based RR (UNRR), by managing the UN of RE and NRE. These algorithms consider two types of UN, namely UN of RE (UN-RE) and UN of NRE (UN-NRE), concerning the UR. The proposed algorithms undergo a rigorous simulation process

with 200 to 2000 URs and 20 to 200 datacenters. They are compared using OCO, TNRE, UN time (UNT), and UN cost (UNCO). The significant contributions of this work are listed as follows.

1. Development of three UR-based scheduling algorithms by managing the UN-RE and UN-NRE sources.

2. The UN of UR is modelled in terms of the percentage of UN-RE and UN-NRE and considered as a variable for RE resources and fixed for NRE resources as per their sustainability.

3. The UNT is determined using the number of RE and NRE resource slots assigned to the UR. This time can be considered as a reserved time to handle the UN.

4. Simulation of three proposed algorithms in ten different datasets and comparison of results in four performance metrics.

1.2.3 Uncertainty Level-Based Algorithms by Managing Renewable Energy for Geo-Distributed Datacenters

RE generation fluctuates with time and location, introducing the UN in fulfilling URs. Thus, NRE generation continues to power the datacenters to achieve stability. Recent works are directed to the use of RE generation followed by NRE generation while assigning the URs to the resources of the datacenters. These works do not model the UN of RE and NRE resources and the UNL. This work extends the three benchmark algorithms, FABEF, HAREF, and RR, by incorporating UNL, which we call UNL-based FABEF (UNL-FABEF), UNL-based HAREF (UNL-HAREF), and UNL-based RR (UNL-RR), respectively. The goal of UNL-FABEF is to minimize the OCO, whereas UNL-HAREF is to maximize the TNRE. On the contrary, UNL-RR assigns the URs to the datacenters in a roundabout fashion. This work also introduces the UNL-based multi-objective scheduling algorithm (UNL-MOSA) to make a trade-off between UNL-FABEF and UNL-HAREF. UNL-MOSA creates a balance between the OCO and the TNRE. All four algorithms consider three UNLs of URs, namely low, medium and high, for RE resources. These al-

gorithms are tested using fifty instances of ten datasets with 200 to 2000 URs and 20 to 200 datacenters and compared using five performance metrics: the OCO, TNRE, TNNRE, UNT and UNCO. The performance of four algorithms in these performance metrics is extensively examined to know their applicability. The uniqueness of this work is listed as follows.

1. We develop four scheduling algorithms for geo-distributed datacenters in which UN is modelled from the user and CSP perspectives. The user's perspective categorizes the UNLs into low, medium, and high, whereas the CSP's perspective presents the uncertainty between 1% to 100%.

2. The UN-RE generation is modelled as dynamic, whereas UN-NRE generation is fixed as per their relevance in real-life scenarios.

3. UNL-MOSA balances the performance of UNL-FABEF and UNL-HAREF using their objectives' linear combination.

4. The proposed algorithms are examined using ten datasets and five performance metrics to show their supremacy and applicability.

1.2.4 A Renewable Energy-Oriented Migration Algorithm for Minimizing Cost in Geo-Distributed Cloud Datacenters

A datacenter migration systematically plans to move URs/infrastructure/assets/ applications from one datacenter to another. It reduces costs and complexities and increases agility, flexibility, scalability, security and performance. As the digital revolution progresses swiftly, datacenter companies have embarked on initiatives to integrate RE sources and detach their total dependence on NRE sources. However, the accessibility of RE sources varies based on geographical location and climate conditions. As a result, URs may be redirected from one datacenter to another depending on the presence of RE resources to minimize the OCO. The benchmark algorithm, FABEF, matches the UR with the available datacenters and prioritizes assignment to the datacenter with the lowest cost.

However, this algorithm fails to utilize the availability of RE resources within the datacenters. This observation motivates us to incorporate RE resources into FABEF by facilitating the migration of URs from one datacenter to another. We introduce a novel RE-oriented migration algorithm (REOMA) to minimize the OCO of geographically distributed datacenters through strategic migration between datacenters. REOMA finds the cost based on the time window of the UR in each datacenter. Then, it fits the cost of each datacenter into a polynomial curve based on the time window and determines the slope and intercept. Subsequently, it finds the migration points between the datacenter with the lowest cost and the other datacenters and performs the migration between a pair of datacenters that results in the lowest cost. We compare the performance of the proposed algorithm with three benchmark algorithms in three metrics: the OCO, TNRE, and TNNRE, using 200 to 2000 URs and 20 to 200 datacenters and demonstrate significant improvements. In summary, the work offers the following contributions to the research community.

1. We develop a novel migration algorithm that integrates RE and NRE sources. This algorithm facilitates the efficient relocation of URs from one datacenter to another within a geo-distributed cloud datacenter environment, aiming to minimize OCO.

2. The proposed migration algorithm fits the costs of datacenters over time intervals for a UR into a polynomial curve and calculates both the slope and intercept.

3. The proposed migration algorithm identifies the intersection point between the least cost datacenter and other datacenters as the migration point.

4. The proposed migration algorithm is evaluated against three benchmark algorithms, FABEF, HAREF and RR, to demonstrate its enhancements in OCO reduction and optimization of TNRE and TNNRE across ten datasets.

1.3 Organization of the Thesis

The main focus of this thesis is to design RE-based scheduling algorithms for geo-distributed datacenters. The thesis comprises seven chapters, namely an introduction, a literature re-

view, four contributions, and a conclusion and future scope. The content of these chapters is briefly described as follows.

Chapter 1: This chapter presents the importance of RE in geo-distributed datacenters, motivation, thesis objectives, significant contributions and thesis outline.

Chapter 2: This chapter presents the literature on RE-based scheduling algorithms based on the thesis objectives, especially cost, RE and NRE generations, UN and migration.

Chapter 3: This chapter presents the PM-FABEF and PM-HAREF algorithms by incorporating processors and memory nodes of datacenters to minimize the OCO of assigning the URs.

Chapter 4: This chapter introduces the UN-FABEF, UN-HAREF and UN-RR algorithms by managing the UN-RE and UN-NRE resources to maximize the TNRE.

Chapter 5: This chapter develops the UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA algorithms by incorporating the UN-RE and UN-NRE resources and UNL to minimize the OCO and maximize the TNRE of datacenters.

Chapter 6: This chapter presents the REOMA algorithm to find the migration points of the URs to facilitate their relocation among datacenters and minimize the OCO.

Chapter 7: This chapter summarizes the contributions of the thesis and discusses further extensions of the proposed works.

Chapter 2

Literature Review

The research on RE for sustainable development is rapidly increasing and is the need of the hour [72–74]. The CSPs are willing to run their datacenters using RE sources to provide low-cost services and attract more enterprises to increase profitability. For instance, Microsoft Azure targets to achieve carbon neutrality by 2030. Similarly, Amazon Web Service (AWS) targets to achieve carbon-free by the year 2025 [75–78]. Still, they face many challenges in supplying power to the datacenters. Therefore, researchers [46, 57, 79–85] suggest using RE and NRE sources to handle future circumstances, such that the OCO is minimized and the TNRE is maximized. Moreover, many GLB algorithms have been developed to balance the assignment of URs or VMs among datacenters [43, 46, 54, 55, 57, 58, 63, 64, 67–69, 80–122]. These algorithms consider the OCO, availability of RE, change in energy, carbon footprint, UN and many more. However, most of these algorithms assume the requirement of UR in terms of the processor nodes without memory nodes. Here, minimizing the cost of the processor node does not necessarily reduce the cost of the memory node and vice-versa. Moreover, the UN of these sources needs to be taken care of to handle the URs without violating their requirements. Alternatively, the SLA between the user and the CSP is violated, leading to service credits. On the other hand, the URs can be migrated between datacenters to meet their requirements. CSPs are generally looking for intelligent algorithms in their platforms to schedule the URs to the datacenters to optimize one or more objectives, namely OCO, time, energy, carbon footprint, operations, overhead, efficiency, and many more. Therefore, we segregate the

existing RE-based scheduling algorithms by considering cost, RE and NRE generations, UN and migration objectives.

2.1 Cost-Based Scheduling Algorithms

Majid and Kumar [80] have discussed RE development in various states/countries/ continents and presented predictions, employments, and obstacles. They have shown the projections from 1990 to 2040 by targeting some regions, namely Africa, Brazil, China, the European Union, India, the Middle East, Russia, and the US. They have reported that the lack of regulations and policies is one of the significant barriers to adopting RE. In the cloud context, the standardization of SLA is needed to avoid this barrier. Toosi and Buyya [46] have introduced a fuzzy logic-based load balancing (FLB) algorithm without considering future RE generation. They have also presented three benchmark algorithms: FABEF, HAREF and RR. FABEF routes the UR to a minimum cost datacenter, whereas HAREF routes the UR to maximum available RE datacenters. On the contrary, RR does not consider the cost and available RE. It simply routes the UR circularly without any further information. They have shown that FABEF saves 14% normalized cost, whereas HAREF saves 7.5% normalized cost compared to worst performing RR.

Nayak, Panda and Das [69] have presented an algorithm, called unconstrained power management (UPM), using three power sources, namely photovoltaic (PV), battery and grid. The objective is to provide a continuous power supply to the datacenters with the help of green and brown energy sources. However, they have not considered the minimum power generation of the three supplies that may lead to power failure. To avoid power failure, Nayak, Panda and Das [64] have again presented an algorithm, called constrained power management (CPM), using four power supplies, namely wind, PV, battery and grid. The algorithm applies a minimum threshold on each RE source to run the datacenter smoothly and avoid power failure. However, they have taken only one datacenter and UR duration without resource slots. Furthermore, Nayak et al. [63] have obtained the advantage of both FABEF and HAREF. They have introduced a multi-objective RE (MORE) algorithm to consider the OCO and the RE generation at a glance. Their algorithm makes a trade-off

between FABEF and HAREF. However, the cost of the memory nodes in datacenters is not taken into consideration. Nayak et al. [100] have developed a minimum utilization (MinUtil) algorithm to allocate the URs to the geo-distributed datacenters. Their algorithm is a task consolidation algorithm in which they consider UR utilization. The datacenters' resources are categorized into RE and NRE resources. They restricted the utilization of NRE resources to a pre-defined threshold, i.e., 70%, whereas the utilization of RE resources is not restricted to take full advantage. They suggest investigating this threshold further.

Al-Jumaili et al. [101] have comprehensively reviewed hybrid RE systems. They have focused on two areas, namely battery management systems and power management systems. They have identified three functions of battery management systems, namely battery protection, power management and battery monitoring. They have controlled power consumption monitoring as follows. If the RE sources generate enough energy and the battery charge is below 20%, then the excess energy is used to charge the battery. However, if the RE sources generate insufficient energy, then the battery is used to supply power. In this case, the battery charge is between 20% to 95%. But, if the battery charge is more than 95%, it is only charged once the level reaches 20%. For this, sensors are used to stop charging and charging upon reaching the mentioned levels. They have suggested lower and upper-range charging as per the institute of electrical and electronics engineers (IEEE) standard. On the contrary, if the RE sources generate insufficient energy and the battery charge is also insufficient, then the electric generator provides an uninterrupted power supply. In the worst case, if the RE sources, battery charge and electric generator do not provide sufficient energy, then the power grid is used to provide the supply. In summary, they have aimed to minimize costs by developing battery charging as a service. Rajagopalan et al. [102] have shown the impact of power systems by combining two emerging technologies, namely cloud computing and the Internet of Things (IoT). More specifically, they aim to improve power distribution efficiency. Cloud computing can help store and analyze data, whereas IoT can help reduce cost, minimize user interaction, and improve services and maintenance. However, seamless integration of multiple technologies is always challenging, especially when applied in real-life scenarios, such as a grid to vehicle technology, electric vehicles and many more.

Hassan et al. [103] have introduced a task-scheduling algorithm to deal with mobility and dealy-sensitive applications. Their objective is to minimize the cost of the applications that apply to mobile cloud computing (MCC) and improve resource utilization, boot time and arrival time (AT). Their proposed approach is divided into three layers: mobile users, task scheduling, and mobile cloud. Here, scheduling is performed by matching resources, sequencing, and offloading tasks. In sequencing tasks, they use three well-known algorithms: shortest job first, shortest size first and first come, first served. They have shown the results by taking tasks for various applications, namely augmented reality, e-transport, three-dimensional (3D) games, and healthcare. However, their algorithm does not explicitly consider RE, which can increase the OCO. Tervydis et al. [104] have developed a tool called cloud-edge-asset-optimizer to provide estimates in order to ensure decision-making and resource allocation. They have mentioned the requirements as cost, battery capacity, and delay. Their tool considers data processing requests (e.g., arrival rate), load balancing, edge parameters (i.e., number of devices, processing time, battery performance and cost), cloud parameters (i.e., number of servers, processing time and cost) and critical parameters (i.e., waiting time and working time). They have listed other related tools, iFogSim, MyiFogSim, EdgeCloudSim, FogNetSim++, EdgeFogCloud, FogTorch, EmuFog and FogBus. The basic difference between their tool and other tools is that if the user enters the wrong parameters, their tool helps correct and explain such parameters. They have suggested further improving this tool to support geographical distribution and load distribution scenarios.

Chakraborty, Toosi and Kopp [88] have introduced a framework, named elastic power utilization, to match the RE generation and energy demand. Their algorithm addresses both the overbooking of resources and VM consolidation. However, they focus only on central processing unit (CPU) utilization and consider only homogenous servers in the datacenters. Regaieg et al. [89] have solved the VM placement problem by forming integer linear programming (ILP). They consider the problem to have multiple objectives, namely minimizing the count of active servers, reducing the wastage of resources, and reducing the number of VMs to lessen power consumption. They model the VM requirements in terms of CPU, memory and storage. They consider both homogeneous and heterogeneous

datacenters in their solution. However, they do not examine RE sources in their work. Chen et al. [55] have presented a scheduling algorithm, MB, to minimize the usage of NRE resources by managing the load among the datacenters. The algorithm is based on the availability of RE resources, energy consumption, deadline, workload structure, fine-grained scheduling, and cooling power. However, they have not considered the cost of memory nodes.

2.2 Renewable and Non-Renewable Generation-Based Scheduling Algorithms

Carvalho et al. [84] have listed the factors influencing the energy regulatory process on the energy clouds. Specifically, they have considered twenty-nine factors grouped into seven types: availability, economic, ideology, information, institutional and market, personal and regulatory infrastructure. They have suggested establishing the mathematical model by analyzing the relationships between these factors. This model can determine the crucial factors that need immediate attention. This thesis highlights some of these types by developing RE-based scheduling algorithms. Toosi and Buyya [46] have discussed three benchmark algorithms: FABEF, HAREF, and RR. HAREF matches a UR to all the available datacenters and assigns it to the highest available RE resource datacenter. They have observed that HAREF utilizes maximum RE resources compared to FABEF and FLB. HAREF utilizes RE generation and is the best performing over all the algorithms regarding RE usage. However, RE usage does not necessarily reduce the OCO, as seen from HAREF and FABEF results. Mishra and Panda [87] have proposed a cost-aware RE scheduling (CRES) algorithm by considering a variety of RE sources and their costs. They assume two types of URs, called non-critical and critical. Non-critical UR can execute using any resources, whereas critical UR can execute only in NRE resources. However, their primary focus is to minimize the OCO without emphasizing much on the usage of RE resources. Le et al. [54] have identified that datacenters' cost and ambient temperature vary based on location. As a result, they have introduced a load distribution policy to save the cost to a greater extent.

They have also shown three baseline policies: RR, worst fit (WF), and SCA. Here, the WF assigns the URs to the highest available resources datacenter. SCA assigns the URs to the least cost datacenter.

Leng and Zhang [105] have suggested evaluating the different plans of RE sources before further exploiting such resources. They have introduced a new evaluation method using the grey cloud model and rough set theory. Their evaluation model is divided into three phases: building a matrix, determining the index weights and finally ranking. They apply rough set theory in the second phase, whereas the grey cloud model in the last phase. They have compared the CO_2 emissions of two RE sources, wind and coal and reported that the effect of coal is greater in the production, manufacturing, operation, maintenance and overall life cycle. In contrast, the effect of wind is more on transport, construction, recycling, and disposal of life cycles. They have shown the life cycle of various electricity generation systems with respect to CO_2 emissions in which coal dominates all the systems with 666 to 888 g/kWh and wind is the least among all with 3.404 to 40.996 g/kWh. Alternatively, the order of the systems is coal, diesel, gas, PV, hydropower and wind from the highest to the lowest. However, their study is limited to RE production of two countries, China and Turkey.

He et al. [106] have ordered the RE sources as hydro, wind, electrochemical, biological and solar using the prospect theory. They have reported that determining the optimal RE source is challenging for sustainable growth. Moreover, they have listed some critical issues as follows. 1) A set of criteria can determine the optimal RE source. However, such criteria need to be more well-defined. 2) The evaluation of RE sources needs to include the UN. However, it has not been taken into consideration. 3) The evaluation of RE ignores the rational choices of the decision-makers. They have solved these issues and shown the use cases by considering China country. They have shown the energy consumption of coal, oil, gas and RE from 2000 to 2020. It is noteworthy to mention that the percentage of energy consumption is dominated by coal from 2000 to 2020, i.e., 68.5% to 56.8%. However, it is slowly reducing year by year. Next to coal, the percentage of energy consumption of oil was 22% in 2000 and reduced to 18.9% in 2020. Next to oil, the percentage of energy consumption of RE was 7.3% in 2000 and increased to 15.9% in 2020. Next to RE, the

percentage of energy consumption of gas was 2.2% in 2000 and increased to 8.4% in 2020. These statistics clearly show the adoption of RE in energy sectors.

Zhao and Zhou [107] have stated that the grid and RE source power the geographically distributed datacenters. Their study also stated that energy consumption is not necessarily related to carbon emissions. As a result, optimizing energy consumption does not optimize carbon emissions. Therefore, both energy consumption and carbon emissions need to be tackled mutually. They have proposed an energy-aware and carbon-aware algorithm to place the VMs on the two cloud servers, namely Hewlett-Packard (HP) ProLiant G4 and HP ProLiant G5. The power consumption of the HP ProLiant G4 server is 86 watts at idle load, whereas it is increased to 117 watts at full load. Similarly, the power consumption of the HP ProLiant G5 server is 93.7 watts at idle load, whereas it is increased to 135 watts at full load. The CPU capacity of these servers is 1860 and 2660 million instructions per second (MIPS), respectively, whereas the number of cores in these servers is 2. They have shown the comparison with the first fit, best fit, energy-aware and carbon-footprint aware with predictive and non-predictive RE source, energy-aware with predictive and non-predictive RE source and carbon footprint-aware with predictive RE source algorithms. However, their algorithm is limited to VM placement without scheduling the URs to the VMs or cloud servers.

Jeong et al. [108] have investigated the public opinion on RE and other energy sources using social media data. They have used correlation and sentiment analyses. They have collected data from Reddit and performed data preprocessing. They have ranked the top five keywords as energy, electricity, RE, oil, nuclear power, and fossil fuel. For instance, the five keywords related to energy are energy, solar, power, oil and nuclear in the order of their rank. Similarly, the five keywords related to electricity are electricity, power, energy, voltage and current. The five keywords related to fossil fuels are coal, gas, energy, solar and power. They found that the relationship between RE and nuclear energy is complementary and positive. On the contrary, the substitute relationship is negative. However, their study does not consider the opinion of decision-makers. Panda and Jana [109] have presented an energy-efficient algorithm for a heterogeneous cloud environment. The algorithm comprises three steps: estimation, normalization and selection, and execution. Although energy

is considerably reduced using this algorithm, they have not taken RE sources into consideration. Table 2.1 summarizes cost-based, renewable and non-renewable generation-based scheduling algorithms.

2.3 Uncertainty-Based Scheduling Algorithms

Kabir et al. [57] have explored various uncertainties and their possible impact on the cloud components. They have identified five parameters affecting the UN: availability, OCO, security, traffic, and workload. They have shown the impact of common influencing factors, such as user location, AT, checkpointing, execution time, VM interruption, network failure, and closest datacenters, on the five parameters. This thesis focuses on two parameters, namely availability and OCO, and considers some of the influencing factors: AT, execution time, and network failure. They have suggested the researchers to design cloud management techniques by incorporating the UN. Koutsoyiannis [83] has suggested that UN modelling is essential in managing the RE. This modelling needs both structural measures and optimization techniques. This thesis intends to focus on optimizing the resources and their OCO.

Panda and Jana [56] have considered UN for the low quality of service (QoS) applications. They categorize the URs into high and low QoSs. They further categorize low QoS into high UN and low UN, respectively. Here, high UN and low UN are determined based on the deviation from the expected completion time. However, their study is limited to task scheduling without considering RE resources. Methenitis et al. [81] have adopted SLA by incorporating cost, reliability, and quantity. The rationality behind this adoption is that electricity production using RE sources is uncertain, and the delivery of the same cannot be promised. However, their study focuses on the smart grid and does not consider the UN of NRE sources. Zhao, Wang and Mo [58] have aimed to manage the workload and energy of datacenters and presented an optimal power generation scheduling algorithm. Their algorithm uses local RE to reduce operating costs and energy consumption. They have suggested combining datacenters and green energy to reduce the environmental impact. In our work, we consider the green energy of datacenters in order to dispatch the URs

Table 2.1: Summary of the cost-based, renewable and non-renewable generation-based scheduling algorithms

Article	Algorithm	Objective		UR Properties				Remark
		Minimize the OCO	Maximize the RE Usage	Start Time	Duration	Node		
						Processor	Memory	
Toosi and Buyya (2015)	FABEF	✓	×	✓	✓	✓	×	The RE resource slots are not explicitly considered.
Toosi and Buyya (2015)	HAREF	×	✓	✓	✓	✓	×	The usage of RE resource slots does not necessarily reduce the OCO.
Toosi and Buyya (2015)	RR	✓	✓	✓	✓	✓	×	The algorithm does not consider RE availability and OCO.
Toosi and Buyya (2015)	FLB	✓	✓	✓	✓	✓	×	The duration is not considered in terms of memory nodes.
Nayak et al. (2022)	MORE	✓	✓	✓	✓	✓	×	The duration is only considered in terms of the processor nodes.
Nayak, Panda and Das (2022)	UPM	✓	×	✓	×	×	×	The least power generation for each source is not considered, which leads to power supply failure.
Nayak, Panda and Das (2022)	CPM	✓	×	✓	×	×	×	The algorithm considers only one datacenter without resource slots.
Mishra and Panda (2022)	CRES	✓	×	✓	✓	✓	×	URs are divided into critical and non-critical based on their importance.

to an appropriate datacenter.

Hasan et al. [110] have focused on green SLA, which is challenging for CSPs and network providers. They have shown both explicit and implicit integration of RE in the datacenters. The availability of the UN of RE is addressed through the virtualization of RE. They have compared total energy demand and RE demand to calculate the surplus and degraded energy to define virtualized RE. In their cloud architecture, they have shown the end-user as a consumer of the SaaS provider, which is further a consumer of the IaaS provider and further a consumer of the energy-as-a-service provider. In each provider, the service-level objectives (SLOs) are well-defined. For instance, SLOs between end-user and SaaS are availability and response time. Similarly, SLOs between SaaS and IaaS are the availability of physical and green resources. On the contrary, SLOs between IaaS and energy-as-a-service are the availability of RE and NRE. They have used cloud SLA language to enable green SLA. Pabitha et al. [111] have addressed the uncertain UR demands and shown its impact on the performance of task scheduling. They have mentioned some uncertain factors, such as bandwidth, processing speed in millions of instructions per second, cost, makespan and task completion time. Subsequently, they have presented an optimization algorithm called a chameleon and remora search for task scheduling. However, they have considered only two datacenters and 100 to 500 tasks for implementation.

Saini, Al-Sumaiti and Kumar [113] have addressed the UN in the context of power supply, storage systems and microgrids. They have proposed a UN quantification fusion mechanism that integrates with RE. Their fusion mechanism is developed using three models: support vector regression (SVR), long short-term memory (LSTM), and convolutional neural networks gated recurrent unit (CNN-GRU), which can forecast a one-day load ahead of time. Specifically, input data is given to the LSTM and SVR models simultaneously, and the outcome is given to CNN-GRU. The performance is compared using mean squared error, mean absolute error, mean absolute percentage error, and root mean squared error, in which the fusion model outperforms other models. However, their framework does not incorporate spatio-temporal aspects. Further, Saini et al. [112] have shown the necessity of cloud energy storage systems, which enable the interaction between utilities and consumers. They have used artificial ecosystem optimization and UN quantization using ma-

chine learning to calculate the ideal battery capacity. Moreover, they consider the UN with respect to price. Their methodology comprises input elements, process data, evaluation process and test process. In the input elements, they collect load data from the individual household. In the process data, they perform preprocessing and split the data into training, validation and testing. In the evaluation process, they use various models, such as LSTM, GRU, bidirectional LSTM (Bi-LSTM) and bidirectional GRU (Bi-GRU), to optimize the loss function. In the test process, they use mean absolute error, mean squared error, and root mean squared error as performance metrics. They found that Bi-LSTM, GRU and LSTM outperform other models in load, PVs and price forecasting, respectively. They have claimed that their approach not only minimizes the cost of users but also maximizes the profit of the provider incorporating UN.

Tchernykh et al. [114] have stated that the study of the UN is limited in the field of cloud computing systems and addressed the UN in the context of availability, confidentiality and integrity. The source of UN is cost, replication, data, energy consumption, migration, fault tolerance, virtualization, elastic provisioning, infrastructure, provisioning time, consolidation, communication, elasticity, availability, etc. They have provided various scheduling objectives, namely expected makespan, expected mean turnaround time, waiting time and bounded slowdown, and expected total weighted completion time and tardiness, to determine the quality of solutions. However, the risk cannot be overcome in future large-scale systems. Xu et al. [67] have presented a job scheduling algorithm to address the cost minimization problem in geographically distributed datacenters. They have integrated reinforcement learning and neural networks. However, they have not considered the UN of jobs in datacenters.

Ahmed [82] has reported that the traditional SLA does not cover the eco-friendly, green, and information technology ethics issues. As a circumstance, they have suggested a green SLA framework by incorporating these issues and adding a new layer to their framework. However, the UN is not considered in the green SLA framework. In summary, most of the algorithms do not consider the UN and assume that the URs finish their execution without interruption. Padhi and Subramanyam et al. [115] have considered the UN of renewable and non-renewable resources and modelled them in percentages between 1% and 100%.

They have also extended the three benchmark algorithms and called them UN-FABEF, UN-HAREF and UN-RR. However, the UN's resources are considered static, and there is no level of uncertainty. The above-discussed scheduling algorithms are compared in terms of the UNL of UR, UN consideration of resources, objectives (i.e., minimizing cost and maximizing available RE usage), scheduling decision (i.e., static and dynamic) and ease of implementation as shown in Table 2.2.

2.4 Migration-Based Scheduling Algorithms

Xu and Buyya [79] have addressed the problem of huge energy consumption in the datacenters. They have suggested managing the loads among various clouds deployed in different time zones to minimize the usage of NRE and maximize the usage of RE resources. However, their model considers RE, power consumption, and carbon emissions without looking into UN in load shifting. Khosravi, Toosi and Buyya [33] have developed an optimal offline cost algorithm. Here, they have assumed the awareness of future RE generation. However, it is impractical to know such a generation apriori. Therefore, they have proposed two online algorithms, optimal online deterministic and future-aware dynamic provisioning, by considering no or limited knowledge of RE generation. However, their main goal is migrating VMs from the datacenter with no RE to the datacenter with more RE.

Silva et al. [117] have conducted energy and resource migrations within datacenters, focusing on transferring these resources between green compute nodes equipped with energy storage and transport capabilities. Nevertheless, their analysis did not encompass NRE sources, especially when RE may be insufficient. Rajeev and Ashok [116] have analyzed the consumers' load-shifting patterns to utilize the RE sources properly. They have considered both temporal and locational characteristics, namely metering and monitoring of RE sources for forecasting. They categorize the energy consumer into low, medium and high-end to provide the solar power in kW. For instance, a high-end consumer requires two, whereas a low-end consumer requires one. On the contrary, a medium-end consumer requires 1.5. Similarly, the number of 12 V, 200 Ampere hours (Ah) batteries required for a high-end consumer is five, whereas low-end and medium-end consumers require three.

Table 2.2: Comparison of UN-based scheduling algorithms

Scheduling Algorithm	UNL	UN	Minimizing OCO	Maximizing Available RE Usage	Scheduling Decision	Simplicity
FABEF [46]	×	×	✓	×	Dynamic	×
UN-FABEF [115]	×	✓	✓	×	Dynamic	×
HAREF [46]	×	×	×	✓	Dynamic	×
UN-HAREF [115]	×	✓	×	✓	Dynamic	×
RR [46]	×	×	×	×	Static	✓
UN-RR [115]	×	✓	×	×	Static	✓
SCA [54]	×	×	✓	×	Dynamic	×
MB [55]	×	×	×	✓	Dynamic	×
WF [54]	×	×	×	×	Dynamic	×
MORE [63]	×	×	✓	✓	Dynamic	×

They categorize the events into temperature and time-shiftable, in which television, microwave, dish wash and washing machine are time-shiftable events. In contrast, freezers, air conditioners and water heaters are temperature-shiftable events. The priority of these events is in the order of washing machine, water heater, dish wash, microwave, television, air conditioner and freezer from high to low. However, the average load is in the order of microwave, water heater, air conditioner, washing machine, dish wash, television and freezer from high to low. They have claimed 8% annual bill savings of households using a load-shifting algorithm.

Guitart [118] has implemented various container migrations, namely iterative and diskless, for specific high-performance computing (HPC) applications. Note that a container is an executable software unit that packs codes and their dependencies and libraries. The author has integrated checkpoint/restore (C/R) techniques of HPC with containerization in order to reduce the freeze time and enable live migrations. Here, the C/R technique enables the storage of the computation state so that computation can be resumed from that point without losing the preceding things. On the other hand, live migration enables an executing process to move from one physical host to another physical host with minimal downtime. It leads to improved load-balancing capabilities. The various forms of live migrations are storage, memory (diskless and iterative) and networking. Further, diskless migration can be considered disk and diskless, without and with page servers. On the contrary, naive/cold migration pauses the executing process in one host, dumps its state, transfers it via the network and resumes its execution in another host. As a result, the downtime is quite high. The iterative migration is similar to the live migration and is presented in two forms, pre-copy and post-copy. In the first form, all the states are copied from source to destination without interrupting the execution at the source. In the later form, the migration is initiated by pausing the execution and transferring the minimal part of the execution state (e.g., registers and CPU state) to the destination host to resume the execution. The remaining part of the execution state is later transferred from the source host to the destination host. However, resource management and scheduling algorithms have not been taken into consideration. Seddiki et al. [119] have performed VM migrations among datacenters and implemented them using Cloudsim. They have considered the availability of RE dynam-

ically in the datacenters. They have enabled two functionalities, namely conceptual and simulation entities, in Cloudsim. The conceptual entities include a renewable cloud datacenter, renewable power host and meta-scheduler, whereas the simulation entities include cluster information, renewable cluster, renewable helper and renewable constants. In their implementation, they use two servers, namely HP ProLiant ML110 G4 and HP ProLiant ML110 G5. Note that the latter is more powerful than the former in terms of processing speed. They have considered three scenarios with the number of hosts as 265, 530 and 800 and VMs as 350, 695 and 1052, respectively. On the other hand, the number of cloudlets in these scenarios ranges from 500 to 3000, 1000 to 5000 and 1500 to 10000, respectively. They have claimed a maximum of 64% RE usage using their proposed systems. However, UR scheduling is not explicitly considered in their systems.

Xu, Toosi and Buyya [122] have targeted minimizing the carbon footprint by utilizing RE and avoiding brown energy. They have presented a self-adaptive algorithm for managing the resources to handle both batch and interactive workloads. Specifically, they have proposed two algorithms, namely the brownout-based algorithm and deferring algorithm, for interactive and batch workloads, respectively. The microservices of interactive and batch workloads are made optional and mandatory, respectively. In their perspective model, the controller monitors, analyses, plans, and executes workloads. They have used nine machines for the performance evaluation, three of which were IBM X3500 M4, four of which were IBM X3200 M3, and two of which were Dell OptiPlex 990. The CPU sizes of these three types of machines are 2, 2.8 and 3.4 gigahertz (GHz). The number of cores is 12, 4 and 4, respectively. Moreover, the memory sizes are 64 Gigabyte (GB), 16 GB and 8GB, respectively, whereas the storage sizes are 2.9 TB, 199 GB and 399 GB, respectively. On the other hand, the idle power of these three types of machines is 153 watts, 60 watts and 26 watts, respectively, whereas the peak power is 230 watts, 150 watts and 106 watts, respectively. Their algorithms minimize brown energy by 21% and increase RE usage by 10%. However, they have not considered multiple clouds available in different zones to migrate the workloads. Yang et al. [120] have examined RE source patterns and introduced a task migration algorithm to relocate workloads to areas with abundant RE generation to combat carbon pollution. Their algorithm distinguishes between two task

types: delay-tolerant and delay-sensitive. However, their model does not account for each task's required number of nodes.

Benblidia et al. [121] have focused on power grid-cloud architecture and modelled the power issues as a non-cooperative game. They have proved the existence of Nash equilibrium. Their proposed scheme is aware of price, gas, power and renewable. In their implementation, they have considered only three datacenters, the peak and off-peak times as 6 AM to 10 PM and 10 PM to 6 AM, respectively, and PUE is between 1.1 and 2, respectively. They have compared with the RE-based scheme and price-based approach in which they achieve a 31.2% improvement regarding power load rate. Grange et al. [123] have taken a batch of jobs with individual due date constraints and proposed an algorithm to minimize the utilization of NRE, consequently lowering costs. However, their algorithm is constrained to a single datacenter, lacking virtualization and full awareness of RE generation. Table 2.3 summarizes migration-based scheduling algorithms.

2.5 Summary

This chapter discusses the importance of RE for sustainable development. Then, it segregates the existing RE-based scheduling algorithms into various objectives, such as cost, RE and NRE generations, UN and migration and explains each in different sections. This chapter also compares various cost-based, RE and NRE-based scheduling algorithms using certain characteristics and remarks in tabular form. Subsequently, it compares various UN-based scheduling algorithms using their consideration, decision and implementations. Finally, it compares various migration-based scheduling algorithms and their applicability in different characteristics.

Table 2.3: Comparison of migration-based scheduling algorithms

Algorithm	Focus on Cost	Focus on RE	Migration	Geo-Distributed Datacenters	UR Properties	Usage of RE	Usage of NRE
					Processor Nodes		
FABEF [46]	✓	×	×	✓	✓	✓	✓
HAREF [46]	×	✓	×	✓	✓	✓	✓
RR [46]	×	×	×	✓	✓	✓	✓
Spatio-Temporal Task Migration [120]	×	✓	✓	✓	×	✓	✓
ABBSH [123]	×	✓	✓	×	×	✓	✓
Resource Allocation [117]	×	✓	✓	✓	×	✓	×

Chapter 3

Efficient Renewable Energy-Based Geographical Load Balancing Algorithms

This chapter targets the problem of assigning n URs with their requirements to m datacenters with o servers each, powered by RE and NRE generations, for a GCC environment. The goal is to reduce the OCO and increase the usage of RE resources. Here, we introduce two algorithms, PM-FABEF and PM-HAREF, toward the solution to the problem. PM-FABEF takes the URs based on their start time (ST) and determines the cost of processor and memory nodes in each datacenter. Then, it calculates the sum of the cost of processor and memory nodes. Subsequently, it schedules the URs to the minimum cost datacenter. On the other hand, PM-HAREF considers the URs based on their ST and calculates the number of RE resource slots in the processor and memory for each datacenter. Then, it calculates the sum of the processor and memory resource slots. Subsequently, it assigns the URs to the datacenter with the maximum RE resource slots. It is noteworthy to mention that both algorithms make decisions based on processor and memory nodes of datacenters in contrast to only processor nodes considered in the literature. We show that PM-FABEF achieves good performance in the OCO, and PM-HAREF achieves better in RE usage. We compare two proposed and three existing benchmark algorithms using ten datasets with 200 to 2000 URs and 20 to 200 datacenters. We also compared all of them using three

performance metrics, called the OCO, TNRE and TNNRE. The comparison results depict the supremacy of the PM-FABEF and PM-HAREF algorithms as per their applicability. The primary novelties of this work are as follows.

1. We develop two GLB algorithms by incorporating processor and memory nodes at a glance.

2. We consider the varying costs with respect to processor and memory nodes in each datacenter to make the proposed algorithms more realistic.

3. The proposed algorithms are compared with three benchmark algorithms to show their efficacy in three performance metrics.

The upcoming sections are outlined as follows. Section 3.1 introduces the cloud model and problem to be investigated. Section 3.2 introduces two proposed algorithms, PM-FABEF and PM-HAREF, with their illustration. Section 3.3 presents the simulation runs' results and discussion. Section 3.4 summarizes the chapter.

3.1 Cloud Model and Problem Statement

This section outlines the system model and defines the problem, including its objectives and constraints.

3.1.1 Cloud Model

We visualize the proposed model in the IaaS cloud, where datacenters are geographically distributed worldwide. For instance, AWS spans over 31 regions, 99 availability zones and more than 450 points of presence [76]. Moreover, Amazon announces another five regions and 15 availability zones to expand its global infrastructure. Therefore, CSPs (e.g., Amazon, Google, International business machines (IBM) and others) rely on RE sources to create a sustainable business for the enterprises/users and the environment. Amazon is currently the leading purchaser of RE and expects to use 100% RE by 2025. It has generated more than 20 Gigawatts of power using RE sources by January 2023. Still, CSPs

rely on NRE sources to avoid any hindrance to RE sources. On the other hand, CSPs receive the URs from the users and accommodate their requests in the resources/servers of the datacenters. They attempt to run the servers using RE sources and switch to NRE sources in case of insufficiency. As a result, CSPs save a considerable amount of cost by minimizing the usage of NRE. As a result, the cost of using green energy resources is low-priced than brown energy resources for the users.

3.1.2 Problem Statement

We consider n URs, UR = $\{UR_1, UR_2, UR_3, \ldots, UR_n\}$ and m datacenters, DC = $\{DC_1, DC_2, DC_3, \ldots, DC_m\}$. A UR, UR_k, $1 \leq k \leq n$, is presented using five requirements, namely UR identification (ID) number, ST, duration (D), number of required processor nodes (NP) and number of required memory nodes (NM). Note that NP and NM can be visualized as virtual CPU (vCPU) and memory in Amazon EC2 instance (i.e., t2.medium, t2.small, t2.micro and so on), respectively. A datacenter, DC_i, $1 \leq i \leq m$, is presented using ID, number of resources/servers, a series of green energy and brown energy resources over time and a cost series of green energy and brown energy resources over time for processor and memory nodes, respectively. It should be noted that the number of resources is the sum of green energy and brown energy resources.

The problem is to determine a mapping (i.e., matching and scheduling) between the URs and resources of the datacenters without prior knowledge of available RE beyond a time window so that the below goals are achieved.

1. Minimize the OCO of datacenters
2. Maximize the proper usage of RE resources
3. Minimize the usage of NRE resources

The following restrictions bound the above problem.

1. The execution order of URs, i.e., $UR_1, UR_2, UR_3, \ldots, UR_n$, remains unchanged.
2. A UR can take a mixture of RE and NRE resources without any hindrance.

3. A UR cannot migrate from one datacenter to another datacenter at any cost.

4. A UR cannot be postponed due to the future availability of RE, i.e., in the next time window.

3.2 Renewable Energy-Based GLB Algorithms

In this section, we introduce two algorithms, PM-FABEF and PM-HAREF, for the GCC environment. They are RE-based scheduling algorithms for GLB. The objective of PM-FABEF and PM-HAREF is to reduce the OCO by improving green energy usage. PM-FABEF determines a UR's cost for processor and memory nodes in each datacenter and calculates the OCO by adding both the cost of processor and memory nodes. It then chooses the datacenter that results in the least OCO. On the contrary, PM-HAREF determines the available RE resource slots for processor and memory in each datacenter as per the UR requirements. Then, it aggregates the total available green energy resource slots and selects the datacenter with the maximum available green energy resource slots. The pseudo-code of the proposed algorithms, PM-FABEF and PM-HAREF, is shown in Algorithms 3.1 and 3.2, respectively.

PM-FABEF takes a set of n URs with their properties, namely, ST, D, NP and NM, a set of m datacenters with a number of resources/servers, a series of green energy and brown energy resources over time and a cost series of green energy and brown energy resources over time for the processor ($proc\_cost\_re$ and $proc\_cost\_nre$) and memory (mem_cost_re and mem_cost_nre). We represent each datacenter as two-dimensional (2D) matrices, P and M, in which columns indicate the time series and rows indicate the processors/memory/ resources/servers. PM-FABEF determines the OCO, TNRE and TNNRE.

PM-FABEF creates a UR queue (URQ) to hold the URs as they arrive in the system. At a time instance, it checks whether the queue is empty (Line 1 of Algorithm 3.1). If the queue is not empty, then it picks a UR from the URQ and finds a suitable datacenter (Line 3 and Line 4). Note that the values of k or UR_k are present in a one-dimensional (1D) array. Then it determines the free resource slots for that UR between the ST and the D from the ST (Line 6 and Line 8). In each time slot, PM-FABEF checks whether the

Algorithm 3.1 PM-FABEF

Input: URQ, n, m, ST, D, o, P, M, $proc\_cost\_re$, $proc\_cost\_nre$, mem_cost_re, mem_cost_nre, NP and NM
Output: OCO, $TNRE$ and $TNNRE$

while URQ is not empty **do**
  Set $OCO \leftarrow 0$, $TNRE \leftarrow 0$ and $TNNRE \leftarrow 0$
  for $k \leftarrow 1$ to n **do**
    for $i \leftarrow 1$ to m **do**
      Set $ACOP[i] \leftarrow 0$, $ACOM[i] \leftarrow 0$, $proc\_slots\_re \leftarrow 0$, $proc\_slots\_nre \leftarrow 0$, $mem\_slots\_re \leftarrow 0$ and $mem\_slots\_nre \leftarrow 0$
      for $l \leftarrow ST[k]$ to $ST[k]$ + $D[k]$ - 1 **do**
        Set $proc\_slots \leftarrow 0$, $mem\_slots \leftarrow 0$, $proc\_flag \leftarrow 0$ and $mem\_flag \leftarrow 0$
        for $j \leftarrow 1$ to o **do**
          if $P[l, j]$ is free and $proc\_flag$ = 0 **then**
            Set $proc\_slots$ += 1
            if $P[l, j]$ is supplied by the RE sources **then**
              Set $proc\_slots\_re$ += 1 and $ACOP[i]$ += $proc\_cost\_re[l]$
            else
              Set $proc\_slots\_nre$ += 1 and $ACOP[i]$ += $proc\_cost\_nre[l]$
            end if
          end if
          if $M[l, j]$ is free and mem_flag = 0 **then**
            Set mem_slots += 1
            if $M[l, j]$ is supplied by the RE sources **then**
              Set mem_slots_re += 1 and $ACOM[i]$ += $mem\_cost\_nre[l]$
            else
              Set mem_slots_nre += 1 and $ACOM[i]$ += $mem\_cost\_nre[l]$
            end if
          end if
          if $proc\_slots$ = $NP[k]$ **then**
            Set $proc\_flag \leftarrow 1$
          end if
          if mem_slots = $NM[k]$ **then**
            Set $mem\_flag \leftarrow 1$
          end if
          if $proc\_flag \leftarrow 1$ and $mem\_flag \leftarrow 1$ **then**
            break
          end if
        end for
      end for
    end for
    for $i \leftarrow 1$ to m **do**
      Set $ACO[i] \leftarrow ACOP[i]$ + $ACOM[i]$
    end for
    Determine $min(ACO)$ and the datacenter i' that keeps the least value
    Schedule the UR k to the selected datacenter i'
    Set OCO += $ACO[i']$, $TNRE$ += $proc\_slots\_re$ + mem_slots_re and $TNNRE$ += $proc\_slots\_nre$ + mem_slots_nre
  end for
end while

Algorithm 3.2 PM-HAREF

Input: URQ, n, m, ST, D, o, P, M, $proc\_cost\_re$, $proc\_cost\_nre$, mem_cost_re, mem_cost_nre, NP and NM
Output: OCO, $TNRE$ and $TNNRE$

while $URQ \neq$ empty **do**
  Set $OCO \leftarrow 0$
  for $k \leftarrow 1$ to n **do**
    for $i \leftarrow 1$ to m **do**
      Set $ACOP[i] \leftarrow 0$, $ACOM[i] \leftarrow 0$, $proc\_slots\_re[i] \leftarrow 0$, $mem\_slots\_re[i] \leftarrow 0$, $proc\_slots\_nre[i] \leftarrow 0$ and $mem\_slots\_nre[i] \leftarrow 0$
      for $l \leftarrow ST[k]$ to $ST[k]$ + $D[k]$ - 1 **do**
        Set $proc\_slots \leftarrow 0$, $mem\_slots \leftarrow 0$, $proc\_flag \leftarrow 0$ and $mem\_flag \leftarrow 0$
        for $j \leftarrow 1$ to o **do**
          if $P[l, j]$ is free and $proc\_flag$ = 0 **then**
            Set $proc\_slots$ += 1
            if $P[l, j]$ is supplied by the RE sources **then**
              Set $proc\_slots\_re[i]$ += 1 and $ACOP[i]$ += $proc\_cost\_re[l]$
            else
              Set $proc\_slots\_nre[i]$ += 1 and $ACOP[i]$ += $proc\_cost\_nre[l]$
            end if
          end if
          if $M[l, j]$ is free and mem_flag = 0 **then**
            Set mem_slots += 1
            if $M[l, j]$ is supplied by the RE sources **then**
              Set $mem\_slots\_re[i]$ += 1 and $ACOM[i]$ += $mem\_cost\_nre[l]$
            else
              Set $mem\_slots\_nre[i]$ += 1 and $ACOM[i]$ += $mem\_cost\_nre[l]$
            end if
          end if
          if $proc\_slots$ = $NP[k]$ **then**
            Set $proc\_flag \leftarrow 1$
          end if
          if $mem\_slots \leftarrow NM[k]$ **then**
            Set $mem\_flag \leftarrow 1$
          end if
          if $proc\_flag$ = 1 and mem_flag = 1 **then**
            break
          end if
        end for
      end for
    end for
    for $i \leftarrow 1$ to m **do**
      Set $ACO[i] \leftarrow ACOP[i]$ + $ACOM[i]$, $TNRE[i] \leftarrow proc\_slots\_re[i]$ + $mem\_slots\_re[i]$ and $TNNRE[i] \leftarrow proc\_slots\_nre[i]$ + $mem\_slots\_nre[i]$
    end for
    Determine $max(TNRE)$ and the datacenter i' that keeps the least value
    Schedule the UR k to the selected datacenter i'
    Set OCO += $ACO[i']$ and determine the $TNRE$ and $TNNRE$
  end for
end while

processor resource slot is free and the flag is unset (Line 9). Note that $P[l, j]$ represents a 2D array (as there are two subscripts). If so, it subsequently finds whether the resource slot is powered by RE (Line 11). If so, it increases the number of used RE resource slots by one and calculates the processor cost (Line 12). Otherwise, it increases the number of used NRE resource slots by one and calculates the processor cost (Line 13 and Line 14).

In a similar manner, PM-FABEF checks whether the memory resource slot is free and the flag is unset (Line 17). Note that $M[l, j]$ represents a 2D array. If so, it subsequently finds whether the resource slot is powered by RE (Line 19). If so, it increases the number of used RE resource slots by one and calculates the memory cost (Line 20). Otherwise, it increases the number of used NRE resource slots by one and calculates the memory cost (Line 21 and Line 22). If the number of required processor slots matches the UR requirement, then PM-FABEF sets the flag (Line 25 to Line 27). Similarly, if the number of required memory slots matches the UR requirement, then PM-FABEF sets the flag (Line 28 to Line 30). If both flags are set, then it exits from the innermost loop (Line 31 to Line 32) and continues with the next time slot (Line 6). Note that the flags are zero in line 7 of Algorithm 3.1 and one in lines 26, 29 and 31. The reason behind this is that these flags count the number of processor and memory slots, respectively. When the count reaches the required slots, they are set as one. However, it is reset to zero for each time slot. The above process is iterated for each datacenter for a specific duration as per the UR requirement (Line 4 to Line 36). Now, PM-FABEF calculates the assignment cost (ACO) by adding the processor and memory costs (Line 37 to Line 39). Then it finds the minimum assignment cost and the datacenter that keeps the least value (Line 40). Finally, it assigns that UR to the corresponding datacenter (Line 41) and updates the OCO, TNRE and TNNRE (Line 42). The above process is iterated for all the URs until the URQ is empty (Line 1 to Line 44).

PM-HAREF works in a similar way to PM-FABEF, but with a different objective. It takes the same inputs and determines the OCO, TNRE and TNNRE. It also maintains URQ to keep the URs (Line 1 of Algorithm 3.2) and processes them in order of arrival (Line 3). It finds a suitable datacenter with maximum RE resource slots, including processor and memory (Line 4). For this, it iterates from the ST of a UR to the ST + D - 1 (Line 6). Each

time slot determines the available RE resource slots for the processor and memory (Line 8 to Line 34) and exits upon identifying the resource slots (Line 32). Once PM-HAREF iterates over all the datacenters, it calculates the ACO to update the OCO (Line 37 to Line 39). It also determines the TNRE by adding processor and memory slots of each datacenter. It selects the datacenter with maximum TNRE and assigns the UR to that datacenter (Line 40 and Line 41). The above process is iterated for all the URs until the URQ is $NULL$ (Line 1 to Line 44).

3.2.1 Time Complexity Analysis

In PM-FABEF, Step 2 requires constant time or $O(1)$. Step 3 loop requires $O(n)$ time. Step 4 loop needs $O(m)$ time. Step 5 requires constant time. Step 6 takes $O(d)$ time by assuming d is the maximum duration. Step 7 needs constant time. In the worst case, the Step 8 loop takes $O(o)$ time. Step 9 to Step 33 take constant time. Step 37 loop needs $O(m)$ time. Step 40 takes $O(m)$ time. Step 41 and Step 42 take constant time. Therefore, the whole time complexity of PM-FABEF (Step 1 to Step 44) is $O(nmdo)$ for assigning n URs to m datacenters. In a similar way, we can calculate that the complexity of PM-HAREF is $O(nmdo)$ time.

3.2.2 Illustration

We demonstrate the proposed algorithms, PM-FABEF and PM-HAREF, using nine URs (i.e., UR_1 to UR_9) and two datacenters (i.e., DC_1 and DC_2). Each datacenter contains seven processors/resources/servers for processor and memory. The UR properties appear in Table 3.1, and the primary setups of datacenters are demonstrated in Fig. 3.1. In Fig. 3.1, RE and NRE resource slots are shown in green and white colours, respectively. Each datacenter's first row of processor and memory represents the cost of brown resource slots. However, the cost of green resource slots is assumed to be 0.1 unit, irrespective of datacenters. The last row of processor and memory represents the time slot. It is initially shown from t = 1 to t = 9.

In the proposed algorithm, PM-FABEF, the first UR, UR_1, requires one unit of proces-

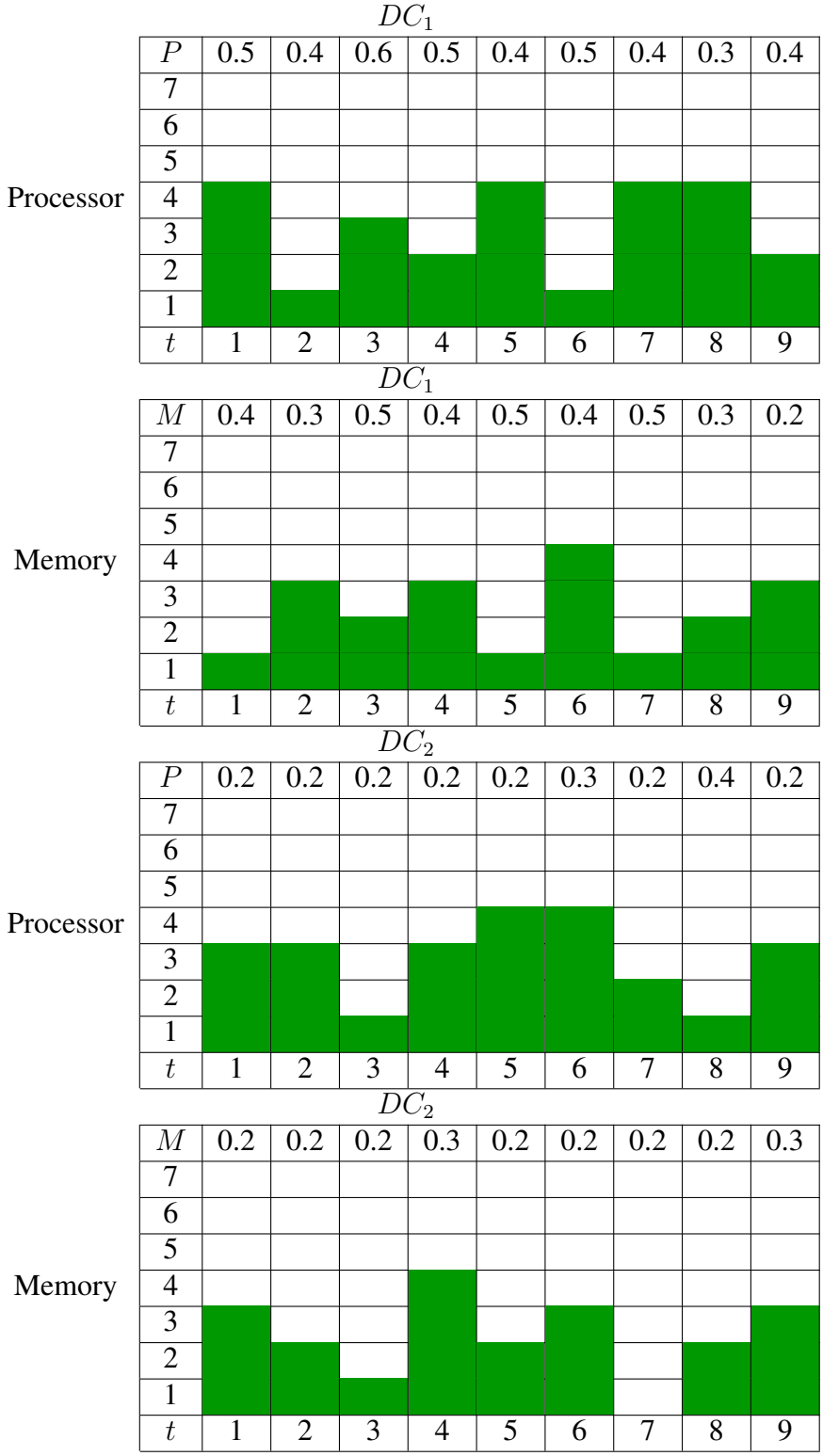

Figure 3.1: A view of datacenter with RE and NRE resource slots.

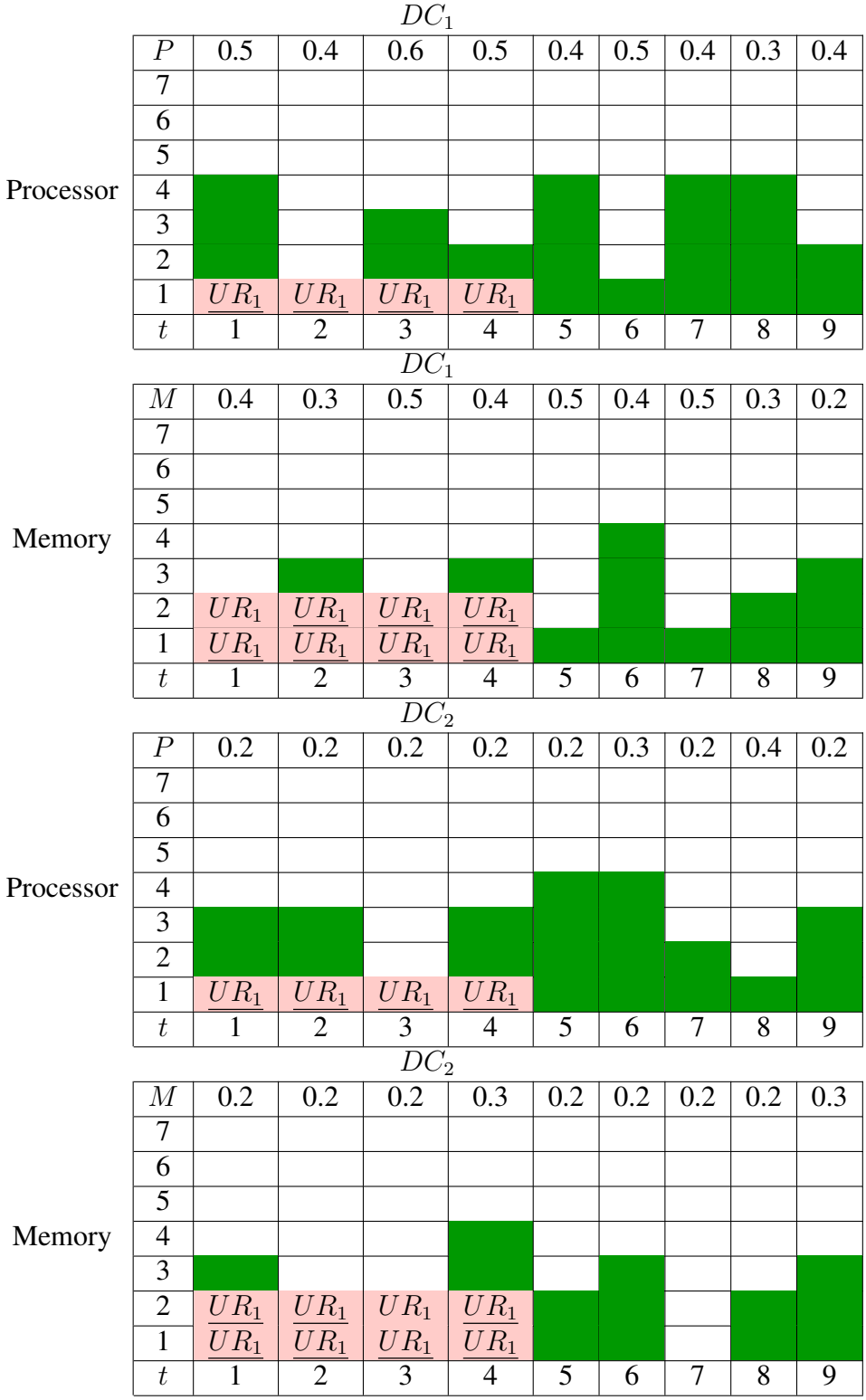

Figure 3.2: Matching of UR_1 to DC_1 and DC_2 in PM-FABEF.

Table 3.1: A set of nine URs with their ST, D, NP and NM

UR ID	ST	D	NP	NM
UR_1	$t = 1$	4	1	2
UR_2	$t = 1$	1	1	2
UR_3	$t = 1$	4	1	2
UR_4	$t = 3$	5	2	1
UR_5	$t = 4$	3	1	2
UR_6	$t = 5$	2	1	2
UR_7	$t = 5$	3	1	2
UR_8	$t = 7$	2	2	3
UR_9	$t = 8$	2	3	4

sor node and two units of memory nodes for the time instance $t = 1$ to $t = 4$, and it can be assigned to one of the two datacenters, i.e., DC_1 and DC_2. The UR_1 is first checked with DC_1 as demonstrated in Fig. 3.2 in which occupied RE resource slots are underlined (__) as the colour of the UR and the colour of the resource slots are overlaps with each other. The processor cost for assigning the UR_1 is 0.4 in DC_1, as the resource slots are renewable. The memory cost for assigning the UR_1 is 1.1, as there are seven RE resource slots and one NRE resource slot. Therefore, the ACO for assigning UR_1 to DC_1 is 1.5. Similarly, the processor and memory cost for assigning UR_1 to DC_2 is 0.4 and 0.9, respectively. Therefore, the ACO is 1.3. As DC_2 results in the minimum cost, UR_1 is assigned to DC_2. The TNRE and TNNRE of DC_2 are 11 and 1, respectively. Next, UR_2 needs one unit of processor node and two units of memory nodes for the time interval $t = 1$. The processor and memory costs for assigning the UR_2 is 0.1 and 0.5 in DC_1. Therefore, the ACO for assigning UR_2 to DC_1 is 0.6. Similarly, the processor and memory costs for assigning UR_2 to DC_2 is 0.1 and 0.3, respectively. Therefore, the ACO is 0.4. As DC_2 results in the minimum cost, UR_2 is assigned to DC_2. The TNRE and TNNRE of DC_2 are 2 and 1, respectively.

Next, UR_3 needs one unit of processor node and two units of memory nodes for the time instance $t = 1$ to $t = 4$. The processor and memory costs for assigning the UR_3 is 0.4 and 1.1 in DC_1. Therefore, the ACO for assigning UR_3 to DC_1 is 1.5. Similarly, the processor and memory costs for assigning UR_3 to DC_2 is 0.5 and 1.4, respectively.

Therefore, the ACO is 1.9. As DC_1 results in the minimum cost, UR_3 is assigned to DC_1. The TNRE and TNNRE of DC_1 are 11 and 1, respectively. Next, UR_4 needs two units of processor nodes and one unit of memory node for the time interval t = 3 to t = 7. The processor and memory costs for assigning the UR_4 is 1.8 and 0.9 in DC_1. Therefore, the ACO for assigning UR_4 to DC_1 is 2.7. Similarly, the processor and memory costs for assigning UR_4 to DC_2 is 1.2 and 0.7, respectively. Therefore, the ACO is 1.9. As DC_2 results in the minimum cost, UR_4 is assigned to DC_2. The TNRE and TNNRE of DC_2 are 11 and 4, respectively. Similarly, UR_5 to UR_9 are assigned to DC_2, DC_1, DC_2, DC_2 and DC_1, respectively. The ACOs are 1.9, 1.3, 1.0, 1.6, 1.9 and 2.2, respectively. However, the OCO is 13.1 as the OCO of DC_1 and DC_2 is 4.7 and 8.4, respectively. The TNRE and TNNRE are 61 and 29, respectively. The Gantt chart of PM-FABEF is depicted in Fig. 3.3.

In the proposed algorithm, PM-HAREF, the first UR, UR_1, requires one unit of processor node and two units of memory nodes for the time instance t = 1 to t = 4. The UR_1 is first checked with DC_1. The TNRE for the processor and memory is four and seven in order to assign the UR_1 to DC_1. Therefore, the TNRE in DC_1 is 11. Similarly, the TNRE for the processor and memory is four and seven in order to assign the UR_1 to DC_2. Therefore, the TNRE in DC_2 is 11. As both datacenters contain the same TNRE, UR_1 is assigned to DC_1 to break the ties. The ACO of assigning UR_1 to DC_1 is 1.5. Next, the UR_2 is checked with DC_1. The TNRE for the processor and memory is one and zero in order to assign the UR_2 to DC_1. Therefore, the TNRE in DC_1 is one. Similarly, the TNRE for the processor and memory is one and two in order to assign the UR_2 to DC_2. Therefore, the TNRE in DC_2 is 3. As DC_2 holds the maximum TNRE, UR_2 is assigned to DC_2. The ACO of assigning UR_2 to DC_2 is 0.3.

Next, the UR_3 is checked with DC_1. The TNRE for the processor and memory is three and two in order to assign the UR_3 to DC_1. Therefore, the TNRE in DC_1 is five. Similarly, the TNRE for the processor and memory is four and six in order to assign the UR_3 to DC_2. Therefore, the TNRE in DC_2 is 10. As DC_2 holds the maximum TNRE, UR_3 is assigned to DC_2. The ACO of assigning UR_3 to DC_2 is 1.4. Next, the UR_4 is checked with DC_1. The TNRE for the processor and memory is eight and four in order to assign the UR_4

DC_1

	P	0.5	0.4	0.6	0.5	0.4	0.5	0.4	0.3	0.4
	7									
	6									
	5									
Processor	4									
	3								UR_9	UR_9
	2								UR_9	UR_9
	1	UR_3	UR_3	UR_3	UR_3	UR_6	UR_6		UR_9	UR_9
	t	1	2	3	4	5	6	7	8	9

DC_1

	M	0.4	0.3	0.5	0.4	0.5	0.4	0.5	0.3	0.2
	7									
	6									
	5									
Memory	4								UR_9	UR_9
	3								UR_9	UR_9
	2	UR_3	UR_3	UR_3	UR_3	UR_6	UR_6		UR_9	UR_9
	1	UR_3	UR_3	UR_3	UR_3	UR_6	UR_6		UR_9	UR_9
	t	1	2	3	4	5	6	7	8	9

DC_2

	R	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.4	0.2
	7									
	6									
	5							UR_8		
Processor	4				UR_5	UR_7	UR_7	UR_8		
	3			UR_4	UR_4	UR_5	UR_5	UR_7		
	2	UR_2		UR_4	UR_4	UR_4	UR_4	UR_4	UR_8	
	1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_8	
	t	1	2	3	4	5	6	7	8	9

DC_2

	M	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.3
	7									
	6							UR_8		
	5				UR_5	UR_7	UR_7	UR_8		
Memory	4	UR_2			UR_5	UR_7	UR_7	UR_8		
	3	UR_2		UR_4	UR_4	UR_5	UR_5	UR_7	UR_8	
	2	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_7	UR_8	
	1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_8	
	t	1	2	3	4	5	6	7	8	9

Figure 3.3: Final Gantt chart of PM-FABEF.

DC_1

Processor

P	0.5	0.4	0.6	0.5	0.4	0.5	0.4	0.3	0.4
7									
6									
5									
4							UR_8		
3			UR_4	UR_4	UR_6	UR_6	UR_8		
2			UR_4	UR_4	UR_4	UR_4	UR_4	UR_8	
1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_8	
t	1	2	3	4	5	6	7	8	9

DC_1

Memory

M	0.4	0.3	0.5	0.4	0.5	0.4	0.5	0.3	0.2
7									
6									
5									
4							UR_8		
3			UR_4	UR_4	UR_6	UR_6	UR_8	UR_8	
2	UR_1	UR_1	UR_1	UR_1	UR_6	UR_6	UR_8	UR_8	
1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_8	
t	1	2	3	4	5	6	7	8	9

DC_2

Processor

P	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.4	0.2
7									
6									
5									
4									
3								UR_9	UR_9
2	UR_3			UR_5	UR_7	UR_7		UR_9	UR_9
1	UR_2	UR_3	UR_3	UR_3	UR_5	UR_5	UR_7	UR_9	UR_9
t	1	2	3	4	5	6	7	8	9

DC_2

Memory

M	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.3
7									
6									
5									
4	UR_3			UR_5	UR_7	UR_7		UR_9	UR_9
3	UR_3			UR_5	UR_7	UR_7		UR_9	UR_9
2	UR_2	UR_3	UR_3	UR_3	UR_5	UR_5	UR_7	UR_9	UR_9
1	UR_2	UR_3	UR_3	UR_3	UR_5	UR_5	UR_7	UR_9	UR_9
t	1	2	3	4	5	6	7	8	9

Figure 3.4: Final Gantt chart of PM-HAREF.

Table 3.2: Comparison of illustration results for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms

Algorithm	OCO	TNRE	TNNRE
PM-FABEF	13.1	61	29
FABEF [46]	14.3	62	28
PM-HAREF	14.8	67	23
HAREF [46]	14.9	63	27
RR [46]	16.5	60	30

to DC_1. Therefore, the TNRE in DC_1 is 12. Similarly, the TNRE for the processor and memory is eight and three in order to assign the UR_4 to DC_2. Therefore, the TNRE in DC_2 is 11. As DC_1 holds the maximum TNRE, UR_4 is assigned to DC_1. The ACO of assigning UR_4 to DC_1 is 2.7. Similarly, UR_5 to UR_9 are assigned to DC_2, DC_1, DC_2, DC_1 and DC_2, respectively. The TNREs for UR_5 to UR_9 are 12, 9, 3, 4, 6 and 9, respectively. The TNRE and TNNRE is 67 and 23, respectively. However, the ACO is 14.8 as the ACO of DC_1 and DC_2 is 8.4 and 6.4, respectively. The Gantt chart of PM-HAREF is depicted in Fig. 3.4. We compare the illustration results of PM-FABEF and PM-HAREF algorithms with FABEF, HAREF and RR algorithms as shown in Table 3.2. The Gantt charts are also shown in Figs. 3.5-3.7. As seen from this table, PM-FABEF outperforms FABEF regarding the OCO, and PM-HAREF beats HAREF regarding the TNRE.

3.3 Simulation Results

This section explains the three performance metrics that are used to carry out the simulation results, system configuration and datasets, and compares the simulation results.

3.3.1 Performance Metrics

We consider three performance metrics, called OCO, TNRE and TNNRE. OCO is the cost of assigning all the URs to the datacenters by considering both processor and memory

DC_1

Processor

P	0.5	0.4	0.6	0.5	0.4	0.5	0.4	0.3	0.4
7									
6									
5								UR_9	
4								UR_9	
3								UR_9	UR_9
2	UR_2			UR_5			UR_8	UR_8	UR_9
1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_8	UR_8	UR_9
t	1	2	3	4	5	6	7	8	9

DC_1

Memory

M	0.4	0.3	0.5	0.4	0.5	0.4	0.5	0.3	0.2
7								UR_9	
6								UR_9	
5								UR_9	
4	UR_2			UR_5				UR_9	UR_9
3	UR_2			UR_5			UR_8	UR_8	UR_9
2	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_8	UR_8	UR_9
1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_8	UR_8	UR_9
t	1	2	3	4	5	6	7	8	9

DC_2

Processor

P	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.4	0.2
7									
6									
5									
4					UR_7	UR_7			
3			UR_4	UR_4	UR_6	UR_6	UR_7		
2			UR_4	UR_4	UR_4	UR_4	UR_4		
1	UR_3	UR_3	UR_3	UR_3	UR_4	UR_4	UR_4		
t	1	2	3	4	5	6	7	8	9

DC_2

Memory

M	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.3
7									
6									
5					UR_7	UR_7			
4					UR_7	UR_7			
3				UR_4	UR_6	UR_6	UR_7		
2	UR_3	UR_3	UR_3	UR_3	UR_6	UR_6	UR_7		
1	UR_3	UR_3	UR_3	UR_3	UR_4	UR_4	UR_4		
t	1	2	3	4	5	6	7	8	9

Figure 3.5: Final Gantt chart of FABEF.

DC_1

Processor									
P	0.5	0.4	0.6	0.5	0.4	0.5	0.4	0.3	0.4
7									
6									
5								UR_9	
4							UR_8	UR_9	
3			UR_4	UR_4			UR_8	UR_9	UR_9
2	UR_2		UR_4	UR_4	UR_4	UR_4	UR_4	UR_8	UR_9
1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_8	UR_9
t	1	2	3	4	5	6	7	8	9

DC_1

Memory									
M	0.4	0.3	0.5	0.4	0.5	0.4	0.5	0.3	0.2
7								UR_9	
6								UR_9	
5								UR_9	
4	UR_2						UR_8	UR_9	UR_9
3	UR_2		UR_4	UR_4			UR_8	UR_8	UR_9
2	UR_1	UR_1	UR_1	UR_1			UR_8	UR_8	UR_9
1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_8	UR_9
t	1	2	3	4	5	6	7	8	9

DC_2

Processor									
P	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.4	0.2
7									
6									
5									
4									
3					UR_7	UR_7			
2				UR_5	UR_6	UR_6			
1	UR_3	UR_3	UR_3	UR_3	UR_5	UR_5	UR_7		
t	1	2	3	4	5	6	7	8	9

DC_2

Memory									
M	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.3
7									
6					UR_7	UR_7			
5					UR_7	UR_7			
4				UR_5	UR_6	UR_6			
3				UR_5	UR_6	UR_6			
2	UR_3	UR_3	UR_3	UR_3	UR_5	UR_5	UR_7		
1	UR_3	UR_3	UR_3	UR_3	UR_5	UR_5	UR_7		
t	1	2	3	4	5	6	7	8	9

Figure 3.6: Final Gantt chart of HAREF.

DC_1

Processor

P	0.5	0.4	0.6	0.5	0.4	0.5	0.4	0.3	0.4
7									
6									
5									
4									
3				UR_5				UR_9	UR_9
2	UR_3	UR_3	UR_3	UR_3	UR_7	UR_7		UR_9	UR_9
1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_7	UR_9	UR_9
t	1	2	3	4	5	6	7	8	9

DC_1

Memory

M	0.4	0.3	0.5	0.4	0.5	0.4	0.5	0.3	0.2
7									
6				UR_5					
5				UR_5					
4	UR_3	UR_3	UR_3	UR_3	UR_7	UR_7		UR_9	UR_9
3	UR_3	UR_3	UR_3	UR_3	UR_7	UR_7		UR_9	UR_9
2	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_7	UR_9	UR_9
1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_5	UR_7	UR_9	UR_9
t	1	2	3	4	5	6	7	8	9

DC_2

Processor

P	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.4	0.2
7									
6									
5									
4							UR_8		
3					UR_6	UR_6	UR_8		
2			UR_4	UR_4	UR_4	UR_4	UR_4	UR_8	
1	UR_2		UR_4	UR_4	UR_4	UR_4	UR_4	UR_8	
t	1	2	3	4	5	6	7	8	9

DC_2

Memory

M	0.2	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.3
7									
6									
5									
4							UR_8		
3					UR_6	UR_6	UR_8	UR_8	
2	UR_2				UR_6	UR_6	UR_8	UR_8	
1	UR_2		UR_4	UR_4	UR_4	UR_4	UR_4	UR_8	
t	1	2	3	4	5	6	7	8	9

Figure 3.7: Final Gantt chart of RR.

nodes. Mathematically,

$$OCO = \sum_{k=1}^{n} \sum_{i=1}^{m} (ACOP[k,i] + ACOM[k,i]) \times X[k,i] \tag{3.1}$$

where

$$X[k,i] = \begin{cases} 1 & \text{if } UR_k \text{ is scheduled to } DC_i \\ 0 & \text{if } UR_k \text{ is scheduled to } DC_{i'}, i \neq i' \end{cases} \tag{3.2}$$

$TNRE$ is the total number of used green energy resource slots by all the URs. Mathematically,

$$TNRE = \sum_{k=1}^{n} TNRE[k] \tag{3.3}$$

$TNNRE$ is the total number of used brown energy resource slots by all the URs. Mathematically,

$$TNNRE = \sum_{k=1}^{n} TNNRE[k] \tag{3.4}$$

3.3.2 System Configuration and Datasets

We carry out the simulation runs with the help of a workstation with the following configuration. (1) Processor name: Intel Xeon Gold Processor 6226R CPU @ 2.90 GHz 2.89 GHz and x64-based processor (2) Random-access memory (RAM) size: 64.0 GB (3) Make and model: HP Incorporated Z6 G4 workstation (4) Operating system: Windows 10 and 64-bit. Each simulation run is independent of the other without any parallel events and is not limited to the above system configuration. However, each simulation contains parallel events, as seen in the proposed algorithms. We generate ten datasets by considering 200 to 2000 URs with a gap of 200 each and 20 to 200 datacenters with a gap of 20 each. As a result, the URs and datacenters are discrete and not continuous. Each dataset is shown in terms of URs $\times$ datacenters. For example, 1000 $\times$ 100 indicates that 1000 URs are to be assigned to 100 datacenters. The datasets are categorized into low and medium datasets. In the low dataset, the number of URs is 200 to 1000, and datacenters is 20 to 100. On the other hand, in the high dataset, the number of URs is 1200 to 2000, and datacenters is 120 to 200. Each dataset comprises five independent instances. The outcomes of the

Table 3.3: UR/datacenter properties and their range

UR/Datacenter Properties	Range
ST	[1 $\sim$ 100]
D	[10 $\sim$ 25]
NP	[10 $\sim$ 100]
NM	[10 $\sim$ 100]
o	[10 $\sim$ 400] per datacenter
$proc\_cost\_nre$	[1 $\sim$ 100]
mem_cost_nre	[1 $\sim$ 100]
$proc\_cost\_re$ and mem_cost_re	1

five instances are summed and divided by the number of instances, i.e., five, to show the result of the dataset. These instances are created using the predefined random function of MATLAB R2022b, version 9.13.0.2105380. Note that the random function follows the uniform discrete distribution and sampling with replacement. The range of UR and datacenter properties is shown in Table 3.3. It should be noted that the number of resource slots is 10 to 400 per datacenters. As a result, the total number of resource slots ranges from 200 to 80000 across all the datacenters.

3.3.3 Results, Comparison and Discussion

The simulation results are conducted by running the proposed and existing algorithms in all 50 instances of 10 datasets. The comparison of the OCO for these algorithms in low and high datasets is shown graphically in Fig. 3.8 and Fig. 3.9, respectively. Here, the x-axis (horizontal) shows the low and high datasets, and the y-axis (vertical) shows the OCO. It can be seen from these figures that the proposed algorithm, PM-FABEF, outperforms all other algorithms in terms of OCO. On the contrary, the HAREF algorithm is the worst-performing algorithm than others. The reason behind the performance of PM-FABEF is that it considers the cost of the processor and memory nodes to schedule the UR to the selected datacenter.

Next, we compare the proposed and existing algorithms regarding the TNRE in Fig. 3.10 (low dataset) and Fig. 3.11 (high dataset). It is clear from the figures that PM-HAREF performs better than other algorithms as it dispatches the URs based on the RE in the

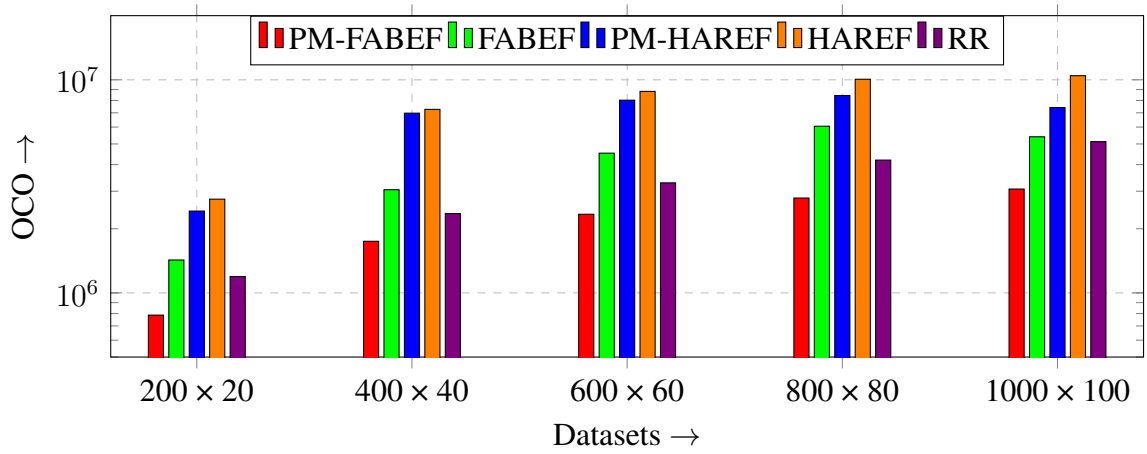

Figure 3.8: Comparison of the OCO for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms in low datasets.

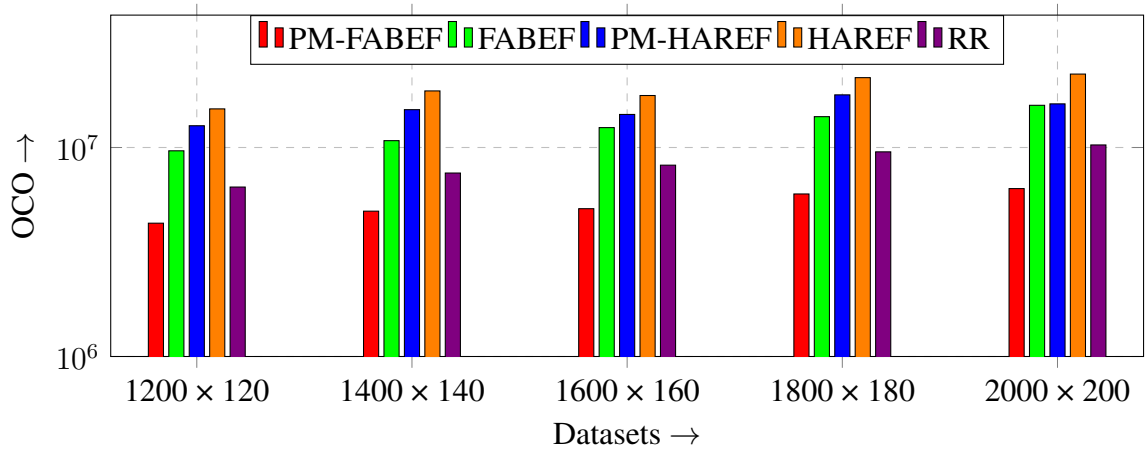

Figure 3.9: Comparison of the OCO for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms in high datasets.

datacenters. Notably, using RE resource slots does not necessarily reduce the OCO, as seen in Fig. 3.8 to Fig. 3.11 and mentioned in [46]. We also compare the TNNRE for the proposed and existing algorithms in low and high datasets and show the results in Fig. 3.12 and Fig. 3.13, respectively. These figures are converse to Fig. 3.10 and Fig. 3.11, respectively, in the sense that the usage of RE resource slots reduces the usage of NRE resource slots. For instance, PM-HAREF uses more RE resource slots. Therefore, as seen in the figures, it reduces the use of NRE resource slots than other algorithms. On the contrary, RR is the worst-performing algorithm compared to other algorithms.

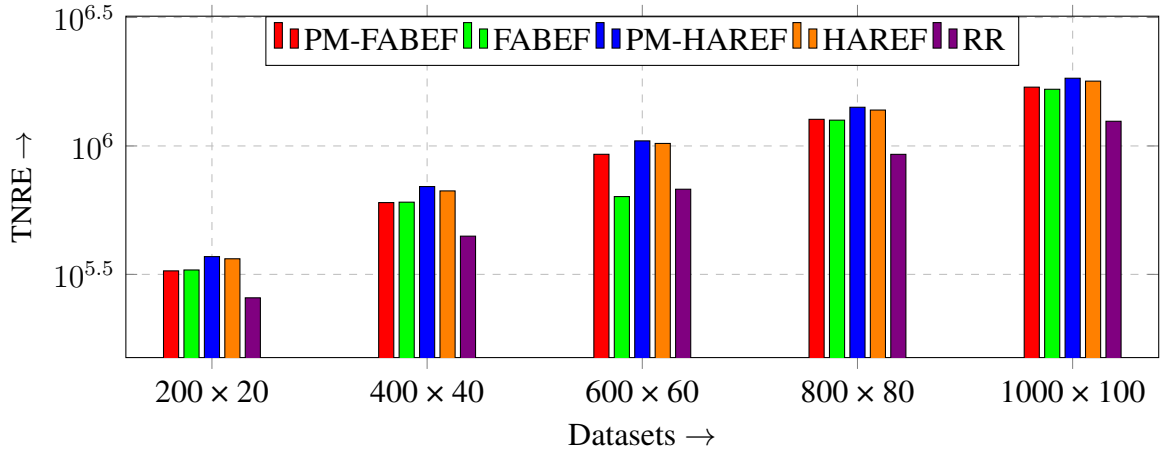

Figure 3.10: Comparison of the TNRE for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms in low datasets.

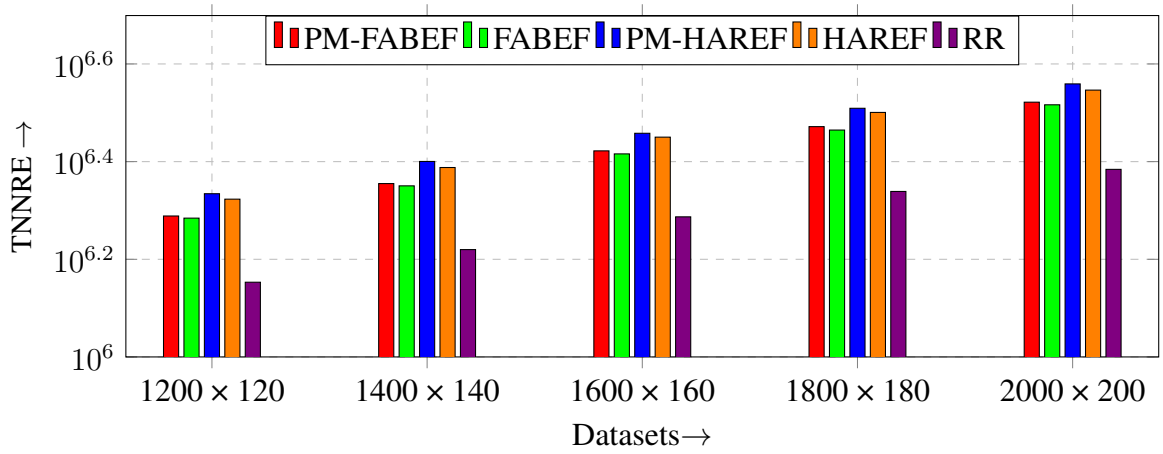

Figure 3.11: Comparison of the TNNRE for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms in high datasets.

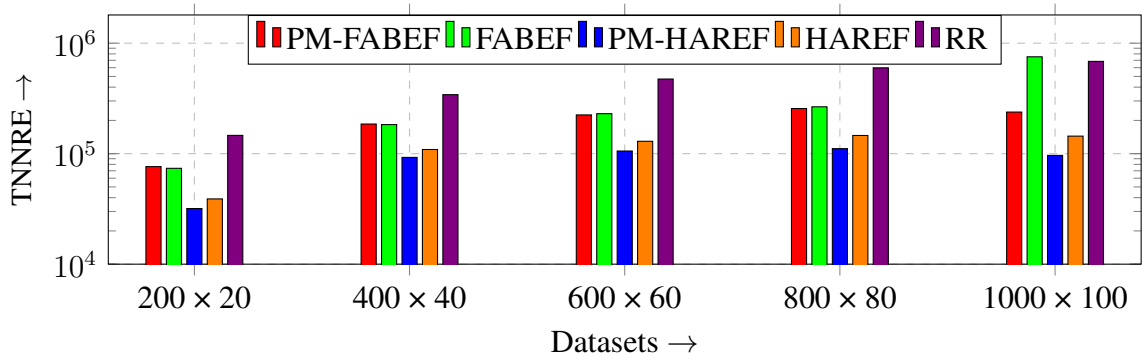

Figure 3.12: Comparison of the TNNRE for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms in low datasets.

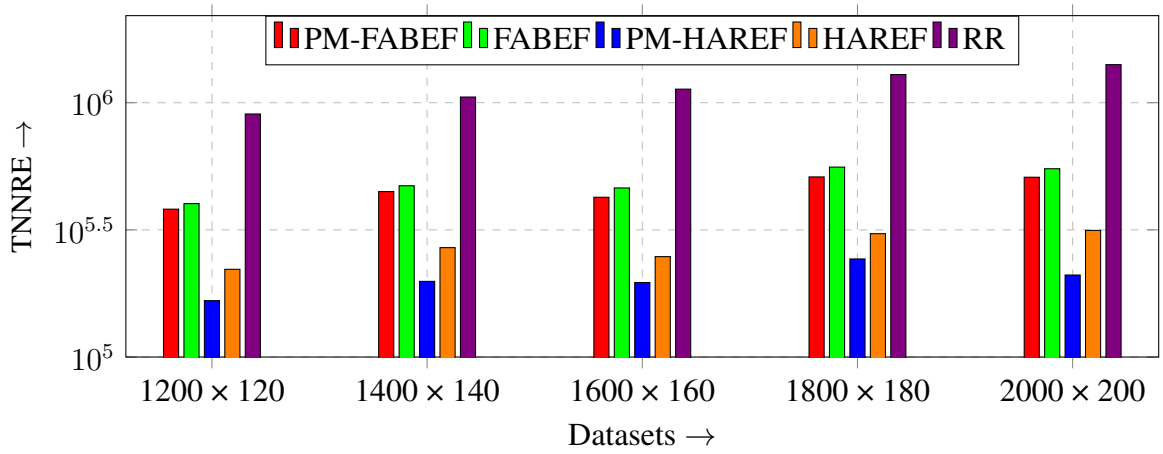

Figure 3.13: Comparison of the TNNRE for PM-FABEF, FABEF, PM-HAREF, HAREF and RR algorithms in high datasets.

3.4 Summary

In this chapter, we have introduced two algorithms, PM-FABEF and PM-HAREF, for GCC. They target to reduce the OCO and TNNRE and maximize the TNRE. PM-FABEF minimizes the OCO, whereas PM-HAREF maximizes the TNRE. Both algorithms appear to require $O(nmdo)$ time for n URs, m datacenters, d maximum duration and o maximum resources. We have presented the simulation results of the proposed algorithms using 50 instances of 10 datasets and compared them with three benchmark algorithms, namely FABEF, HAREF and RR, using three performance metrics. The results and comparison show the superiority of PM-FABEF in the OCO and PM-HAREF in the TNRE and TNNRE. We found a non-linear relationship between the OCO and TNRE. However, the proposed algorithms do not consider the UN of the RE and NRE resources. As a result, the D of UR may deviate due to unforeseen circumstances. Therefore, we aim to explore the the UN of the RE and NRE resources and develop RE-based scheduling algorithms by managing the UN in the next chapter.

Chapter 4

User Request-Based Scheduling Algorithms by Managing Uncertainty of Renewable Energy

This chapter addresses the problem of matching and scheduling URs to the resources of datacenters by managing the UN of RE and NRE resources. It presents three RE-based scheduling algorithms for cloud computing, namely UN-FABEF, UN-HAREF, and UN-RR. These algorithms considers two types of UN, namely UN-RE and UN-NRE, and model the UN in terms of the percentage of UN-RE and UN-NRE. The UN-RE is variable, and the UN-NRE is fixed as RE and NRE resources are unreliable and reliable, respectively. For instance, consider a UR (i.e., UR_1) that requires one node for four time slots in one of the two available datacenters. The UN of this request is pre-determined as 50% for RE and 10% for NRE. The request is matched with datacenters DC_1 and DC_2. In both datacenters, four RE resource slots are available. Therefore, there is no need to assign the NRE resources. The UN is calculated by multiplying four RE resource slots and 50%, which is 2. This means that the duration of UR_1 can be extended from 4 to 6 (i.e., 4 + 2) by incorporating the UN. Note that this is essential for fulfilling the requirements as per the SLA.

UN-FABEF matches the UR to all the available datacenters and determines the OCO, including UN time duration. Then it assigns that UR to a datacenter that results in the

least OCO. UN-HAREF matches the UR to all the available datacenters and determines the TNRE, including UN time duration. Then it assigns that UR to a datacenter that results in the highest TNRE. On the contrary, UN-RR assigns the URs to the datacenters in a circular fashion. The performance of the proposed algorithms is assessed through a rigorous simulation process by incorporating UN and tested with 200 to 2000 URs and 20 to 200 datacenters. The results are obtained in terms of OCO, TNRE, UNT and UNCO. Note that there is no dependency between the OCO and TNRE. To the best of our knowledge, existing literature does not consider the UN of URs in terms of the percentage of UN-RE and UN-NRE. Therefore, the proposed algorithms are compared among themselves to show their applicability.

The significant contributions of this chapter are listed as follows.

1. Development of three RE-based scheduling algorithms by managing the UN-RE and UN-NRE sources.

2. The UN of UR is modelled in terms of the percentage of UN-RE and UN-NRE and considered as a variable for RE resources and fixed for NRE resources as per their sustainability.

3. The UN time is determined using the TNRE and TNNRE that are assigned to the UR. This time can be considered as a reserved time to handle the UN.

4. Simulation of three proposed algorithms in ten different datasets and comparison of results in four different performance metrics.

The remaining sections of this chapter are arranged as follows. Section 4.1 presents the RE-based system model with UN and the problem statement with objectives and constraints. Section 4.2 focuses on three proposed algorithms, their pseudocode, illustrations, and complexity analysis. Section 4.3 discusses the performance metrics, the simulation configuration, the generation of datasets, the simulation results, and the performance analysis. Section 4.4 summarizes this chapter.

4.1 Renewable Energy-Based System Model and Problem Formulation

This section discusses the system model and presents the problem with objectives and constraints.

4.1.1 System Model

Consider a CSP that hosts a set of datacenters to assign resources to the URs as per their SLA requirements. These datacenters are powered by both RE and NRE sources. However, the CSP tries to balance the power requirement of the resources of the datacenters by RE sources as it is sustainable and environment-friendly. In case of further power requirements, it can be provided using NRE sources. In the view of CSP, the cost of using RE resources is negligible, whereas the cost of using NRE resources varies with respect to time as it relies on electricity generation costs. On the contrary, in the view of the user, the cost of using RE resources is cheaper than NRE resources. However, the UN of using RE resources is more compared to the NRE resources. In the proposed model, the UR is matched with all the available datacenters to determine the TNRE and TNNRE as per its requirements. Subsequently, the UN can be determined using these resource slots. Finally, the OCO of the UR is determined in each datacenter, including UN time duration. In the case of UN-FABEF, the UR is assigned to the datacenter with the least OCO. On the contrary, the UN-HAREF assigns the UR to the datacenter with the highest TNRE. However, UN-RR assigns the UR to the datacenter without considering the OCO and TNRE. The above model is an extension of the model considered in [46, 63].

4.1.2 Problem Formulation

Consider a set of n URs, UR = $\{UR_1, UR_2, UR_3, \ldots, UR_n\}$ and a set of m datacenters, DC = $\{DC_1, DC_2, DC_3, \ldots, DC_m\}$. Note that $n >> m$. Each UR, UR_k, $1 \leq k \leq n$, is represented using a 6-tuple, i.e., UR_k = $<UR\ ID, ST, D, N, UN\text{-}RE, UN\text{-}NRE>$. Here, $UR\ ID$ represents the unique ID of UR_k, ST represents start time of UR_k, D rep-

resents duration of UR_k, N represents the number of nodes required for UR_k, $UN\text{-}RE$ represents the UN of RE resources available for UR_k and $UN\text{-}NRE$ represents the UN of NRE resources available for UR_k. $UN\text{-}RE$ and $UN\text{-}NRE$ can be any value between 1% and 100%. Mathematically, $UN\text{-}RE$ = [$1\% \sim 100\%$] and $UN\text{-}NRE$ = [$1\% \sim 100\%$]. In the best-case scenario, the value of $UN\text{-}RE$ and $UN\text{-}NRE$ is 0%. However, in the worst-case scenario, the value of $UN\text{-}RE$ and $UN\text{-}NRE$ is 100%.

Each datacenter, DC_i, $1 \leq i \leq m$, is represented by using a 5-tuple, i.e., DC_i = $<DC\ ID$, o, ARE, $ANRE$, $CO>$. Here, $DC\ ID$ represents the unique ID of DC_i, o represents a number of resources in DC_i, ARE represents a series of available RE resources in DC_i over a period of time, $ANRE$ represents a series of available NRE resources in DC_i over a period of time and CO represents the cost of RE and NRE resources of DC_i, respectively. The sum of RE (i.e., re) and NRE (i.e., nre) resources is o. Mathematically, o = re + nre. The problem is to match each UR, one by one, with all the available datacenters and schedule the UR to a suitable datacenter, such that the following objectives are fulfilled.

1. The OCO is minimized.
2. The TNRE is maximized.
3. The UNT and UNCO are minimized.

This problem is formulated with the following constraints.

1. The order of URs remains intact.
2. The number of nodes can be provided from either RE or NRE resources or both, and it cannot exceed the maximum available resources at any particular time.
3. The duration of UR can be adjusted by incorporating the UN.

4.2 Proposed Scheduling Algorithms

This section presents three proposed algorithms: UN-FABEF, UN-HAREF, and UN-RR. These algorithms aim to minimize the OCO, UNT and UNCO, and maximize the TNRE.

The pseudo-codes for UN-FABEF, UN-HAREF and UN-RR are shown in Algorithm 4.1 to Algorithm 4.3, respectively.

4.2.1 UN-FABEF

UN-FABEF picks a UR (say, UR_k) from the URQ (Line 1 and Line 3 of Algorithm 4.1) and matches the UR with all the available datacenters (Line 4). In the process of matching, it finds the resource slots that are available between ST of UR_k, and the sum of ST and D of UR_k minus one, as per the requirement of UR_k (Line 6). Then it determines the number of RE (i.e., $slots\_re$) and NRE (i.e., $slots\_nre$) resource slots that can be given to that UR and the ACO without UN (Line 8 to Line 21).

If the number of resource slots is equal to the number of resource slots required for that UR at a time instance (Line 18), then it breaks the loop (Line 19) and continues with another time instance (Line 6). Otherwise, it will be an infinite loop. Next, UN-FABEF calls the Procedure 1 ($DETERMINE\text{-}UN$) to determine the UN (Line 23). UN can be determined as follows (Line 1 of Procedure 1).

$$UN[k] = \lceil (re \times UN\text{-}RE[k] + nre \times UN\text{-}NRE[k]) \rceil, 1 \leq k \leq n \tag{4.1}$$

where $UN\text{-}RE[k]$ and $UN\text{-}NRE[k]$ are UN of RE and NRE for UR_k. The range of $UN\text{-}RE$ and $UN\text{-}NRE$ is between 1% and 100%. As the number of resource slots is an integer, we use the ceil function ($\lceil\ \rceil$) to round the value to the next integer.

Next, the D of UR_k is extended from $ST[k]$ + $D[k]$ - 1 to $ST[k]$ + $D[k]$ + $\lceil \frac{UN[k]}{N[k]} \rceil$ by incorporating the UN (Line 2). Then it adds the number of RE and NRE resource slots that can be required to fulfil the UN and calculates the updated ACO by considering UN (Line 4 to Line 17). This procedure is called for each UR to determine the updated ACO.

Now, the least ACO is calculated, and the corresponding datacenter is identified (Line 25 of Algorithm 4.1). Finally, the UR is assigned to the least ACO datacenter, and the OCO is updated (Line 26 and Line 27).

Algorithm 4.1 UN-FABEF

Input: URQ, ST, D, N, R, CO, n, m, o
Output: OCO and $TNRE$

1: **while** URQ is not $NULL$ **do**
2: Set $OCO \leftarrow 0$
3: **for** $k \leftarrow 1, 2, 3, \ldots, n$ **do**
4: **for** $i \leftarrow 1, 2, 3, \ldots, m$ **do**
5: Set $ACO[i] \leftarrow 0$, $slots\_re \leftarrow 0$ and $slots\_nre \leftarrow 0$
6: **for** $l \leftarrow ST[k], ST[k] + 1, ST[k] + 2, \ldots, ST[k] + D[k] - 1$ **do**
7: Set $slots \leftarrow 0$
8: **for** $j \leftarrow 1, 2, 3, \ldots, o$ **do**
9: **if** $R[l, j]$ is not assigned to any UR **then**
10: $slots \mathrel{+}= 1$
11: **if** $R[l, j]$ is powered by the RE sources **then**
12: $slots\_re \mathrel{+}= 1$
13: **else**
14: $slots\_nre \mathrel{+}= 1$
15: $ACO[i] \mathrel{+}= CO[l]$
16: **end if**
17: **end if**
18: **if** $slots = N[k]$ **then**
19: break
20: **end if**
21: **end for**
22: **end for**
23: Call $DETERMINE\text{-}UN(k, UN\text{-}RE, UN\text{-}NRE, ST, D, N, R, o, re, nre, ACO, CO)$
24: **end for**
25: Find $min(ACO)$ and determine the best datacenter i' that holds the minimum value
26: Assign the UR k to the datacenter i'
27: $OCO \mathrel{+}= ACO[i']$ and update the $TNRE$
28: **end for**
29: **end while**

Procedure 1 $DETERMINE\text{-}UN$ (k, $UN\text{-}RE$, $UN\text{-}NRE$, ST, D, N, R, o, re, nre, ACO, CO)

1: $UN[k] \leftarrow \lceil re \times UN\text{-}RE[k] + nre \times UN\text{-}NRE[k] \rceil$
2: **for** $l \leftarrow ST[k]$ + $D[k]$, $ST[k]$ + $D[k]$ + 1,. . ., $ST[k]$ + $D[k]$ + $\lceil \frac{UN[k]}{N[k]} \rceil$ **do**
3: Set $slots \leftarrow 0$
4: **for** $j \leftarrow 1, 2, 3, \ldots, o$ **do**
5: **if** $R[l, j]$ is not assigned to any UR **then**
6: $slots \mathrel{+}= 1$
7: **if** $R[l, j]$ is powered by the RE sources **then**
8: $slots\_re \mathrel{+}= 1$
9: **else**
10: $slots\_nre \mathrel{+}= 1$
11: $ACO[i] \mathrel{+}= CO[l]$
12: **end if**
13: **end if**
14: **if** $slots = N[k]$ **then**
15: break
16: **end if**
17: **end for**
18: **end for**

4.2.2 UN-HAREF

UN-HAREF picks a UR (say, UR_k) from the URQ (Line 1 and Line 3 of Algorithm 4.2) and matches the UR with all the available datacenters (Line 4). In the process of matching, it finds the resource slots that are available between ST of UR_k, and the sum of ST and D of UR_k minus one (Line 6). Then, it determines the $slots\_re$ and $slots\_nre$ that can be given to that UR and the ACO without UN (Line 8 to Line 21). We exit from the inner for loop (line 8) using the break statement in line 19. This is performed when the node requirements of a UR are fulfilled by a datacenter in a particular time slot.

Next, UN-HAREF calls the Procedure 2 ($DETERMINE\text{-}UN\text{-}HAREF$) to determine the UN (Line 23). Like UN-FABEF, the D of UR_k is extended from $ST[k]$ + $D[k]$ - 1 to $ST[k]$ + $D[k]$ + $\lceil \frac{UN[k]}{N[k]} \rceil$ by incorporating the UN (Line 2). Then, it adds the $slots\_re$ and $slots\_nre$ that can be required to fulfil the UN and calculates the updated ACO by considering UN (Line 4 to Line 17).

Now, the datacenter with the highest $slots\_re$ is identified (Line 25 of Algorithm 4.2). Finally, the UR is assigned to the datacenter with the highest $slots\_re$, and the OCO is

updated (Line 26 and Line 27).

Algorithm 4.2 UN-HAREF

Input: URQ, ST, D, N, R, CO, n, m, o
Output: OCO and $TNRE$

1: **while** $URQ \neq NULL$ **do**
2: Set $OCO \leftarrow 0$
3: **for** $k \leftarrow 1, 2, 3, \ldots, n$ **do**
4: **for** $i \leftarrow 1, 2, 3, \ldots, m$ **do**
5: Set $ACO[i] \leftarrow 0$, $slots\_re[i] \leftarrow 0$ and $slots\_nre[i] \leftarrow 0$
6: **for** $l \leftarrow ST[k], ST[k] + 1, ST[k] + 2, \ldots, ST[k] + D[k] - 1$ **do**
7: Set $slots \leftarrow 0$
8: **for** $j \leftarrow 1, 2, 3, \ldots, o$ **do**
9: **if** $R[l, j]$ is not assigned to any UR **then**
10: $slots \mathrel{+}= 1$
11: **if** $R[l, j]$ is powered by the RE sources **then**
12: $slots\_re[i] \mathrel{+}= 1$
13: **else**
14: $slots\_nre[i] \mathrel{+}= 1$
15: $ACO[i] \mathrel{+}= CO[l]$
16: **end if**
17: **end if**
18: **if** $slots = N[k]$ **then**
19: break
20: **end if**
21: **end for**
22: **end for**
23: Call $DETERMINE\text{-}UN\text{-}HAREF(k, UN\text{-}RE, UN\text{-}NRE, ST, D, N, R, o, re, nre, ACO, CO)$
24: **end for**
25: Find $max(slots\_re)$ and determine the best DC i' that holds the maximum value
26: Assign the UR k to the DC i'
27: $OCO \mathrel{+}= ACO[i']$ and update the $TNRE$
28: **end for**
29: **end while**

4.2.3 UN-RR

UN-RR picks a UR (say, UR_k) from the URQ (Line 1 and Line 3 of Algorithm 4.3). Then it determines the datacenter by finding whether k is a multiple of m (Line 4). In the process

Procedure 2 $DETERMINE\text{-}UNCERTAINTY\text{-}HAREF(k, UN\text{-}RE, UN\text{-}NRE, ST, D, N, R, o, re, nre, ACO, CO)$

1: $UN[k] \leftarrow \lceil re[i] \times UN\text{-}RE[k] + nre[i] \times UN\text{-}NRE[k] \rceil$
2: **for** $l \leftarrow ST[k]$ + $D[k]$, $ST[k]$ + $D[k]$ + 1,. . ., $ST[k]$ + $D[k]$ + $\lceil \frac{UN[k]}{N[k]} \rceil$ **do**
3: Set $slots \leftarrow 0$
4: **for** $j \leftarrow 1, 2, 3, . . ., o$ **do**
5: **if** $R[l, j]$ is not assigned to any UR **then**
6: $slots \mathrel{+}= 1$
7: **if** $R[l, j]$ is powered by the RE sources **then**
8: $slots\_re[i] \mathrel{+}= 1$
9: **else**
10: $slots\_nre[i] \mathrel{+}= 1$
11: $ACO[i] \mathrel{+}= CO[l]$
12: **end if**
13: **end if**
14: **if** $slots$ = $N[k]$ **then**
15: break
16: **end if**
17: **end for**
18: **end for**

of matching, it finds the resource slots that are available between ST of UR_k, and the sum of ST and D of UR_k minus one (Line 6). Then, it determines the $slots\_re$ and $slots\_nre$ that can be given to that UR and the OCO (Line 8 to Line 21).

We use the break statement in line 19 to exit from the line 8 inner for loop. This is done when a datacenter in a particular time slot fulfils the node requirements of a UR. Next, UN-RR calls the Procedure 1 ($DETERMINE\text{-}UN$) to determine the UN (Line 23). Note that the procedure is the same as the UN-FABEF. Hence, it is not explicitly shown here. However, the symbol ACO needs to be replaced with OCO in line 11 of Procedure 1.

4.2.4 Time Complexity Analysis

Let n be the number of UR, m be the number of datacenters, d be the maximum duration of all the URs, and o be the number of resources. In the UN-FABEF, the process of matching for each request takes $O(mdo)$ time. Procedure 1 takes $O(do)$ time for each UR. The process of scheduling for each request takes $O(m)$ time. Therefore, the overall time complexity of UN-FABEF is $O(nmdo)$ time for executing all the URs.

Like UN-FABEF, the overall time complexity of UN-HAREF is $O(nmdo)$ time for executing all the URs. In the UN-RR, the process of matching for each request takes $O(do)$ time. Procedure 1 takes $O(do)$ time for each UR. The process of scheduling for each request takes $O(1)$ time. Therefore, the overall time complexity of UN-RR is $O(ndo)$ time for executing all the URs. Note that the selection of the datacenter in UN-RR takes $O(1)$ time.

4.2.5 Illustration

Let us consider that there are nine URs (i.e., UR_1 to UR_9) with their tuple (i.e., ST, D, N, UN-RE and UN-NRE) as shown in Table 4.1 and these URs need to be matched to two datacenters (i.e., DC_1 and DC_2) in order to assign to one of the datacenters. Note that we use different colours for URs to uniquely identify each pictorially during the matching process and in the Gantt chart.

An initial setup of two datacenters and their resource is shown in Table 4.2. Here, we show seven resources in each datacenter for simplicity of illustration, and it is shown in the second column of the same table. The first row and last row of each datacenter show the cost of the NRE resources and time slots. For instance, the cost of the NRE resources in DC_1 is 0.3 at time slot t = 3, whereas the cost of the NRE resources in DC_2 is 0.5 at the same time slot. The green colour in each datacenter represents the RE resources, and the white colour represents the NRE resources, respectively. As seen in this table, the RE resources of each datacenter fluctuate over the time slots. The cost of RE resources is lower than that of NRE resources. Therefore, it is assumed to be negligible for easy illustration.

4.2.5.1 Illustration of UN-FABEF

In the UN-FABEF, the first UR (i.e., UR_1) requires one node for the time slots t = 1 to t = 4 (i.e., duration of 4 time slots) without UN. The UN of RE for this UR is pre-determined as 50%, and NRE is pre-determined as 10%. The UR_1 is first matched with DC_1 as shown in Table 4.3 in which we underline (_) the RE resource slots occupied by that UR. Here, DC_1 can provide four RE resource slots. Therefore, the finish time of UR_1 in DC_1 is determined

Algorithm 4.3 UN-RR

Input: URQ, ST, D, N, R, CO, n, m, o
Output: OCO and number of used RE resources

1: **while** $URQ \neq NULL$ **do**
2: Set $OCO \leftarrow 0$
3: **for** $k \leftarrow 1, 2, 3, \ldots, n$ **do**
4: (k is a multiple of m) ? $i \leftarrow m$: $i \leftarrow k \; mod \; m$
5: $ACO[i] \leftarrow 0$, $re \leftarrow 0$ and $nre \leftarrow 0$
6: **for** $l \leftarrow ST[k], ST[k] + 1, ST[k] + 2, \ldots, ST[k] + D[k] - 1$ **do**
7: Set $slots \leftarrow 0$
8: **for** $j \leftarrow 1, 2, 3, \ldots, o$ **do**
9: **if** $R[l, j]$ is not assigned to any UR **then**
10: $slots \mathrel{+}= 1$
11: **if** $R[l, j]$ is powered by the RE sources **then**
12: $slots\_re \mathrel{+}= 1$
13: **else**
14: $slots\_nre \mathrel{+}= 1$
15: $OCO[i] \mathrel{+}= CO[l]$
16: **end if**
17: **end if**
18: **if** $slots = N[k]$ **then**
19: break
20: **end if**
21: **end for**
22: **end for**
23: Call $DETERMINE\text{-}UN(k, UN\text{-}RE, UN\text{-}NRE, ST, D, N, R, o, re, nre, ACO, CO)$
24: **end for**
25: Update the $TNRE$
26: **end while**

Table 4.1: A set of nine URs with their tuple

$UR\ ID$	ST	D	N	$UN\text{-}RE$	$UN\text{-}NRE$
UR_1	1	4	1	50%	10%
UR_2	1	1	1	100%	10%
UR_3	1	4	1	25%	10%
UR_4	3	5	2	60%	10%
UR_5	4	3	1	33%	10%
UR_6	5	2	1	50%	10%
UR_7	5	3	1	33%	10%
UR_8	7	2	2	100%	10%
UR_9	8	2	3	50%	10%

Table 4.2: An initial setup of datacenters and their resource, and cost of the NRE resources

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6										
	5										
	4										
	3										
	2										
	1										
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5										
	4										
	3										
	2										
	1										
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

as 4 + $\lceil(4 \times 50\%)\rceil$ = 6 with UN, and the ACO is calculated as negligible (i.e., 0). Then, UR_1 is matched DC_2 as shown in Table 4.3. Like DC_1, DC_2 can provide four RE resource slots. Therefore, the finish time of UR_1 in DC_2 is determined as 4 + $\lceil(4 \times 50\%)\rceil$ = 6. However, the ACO is calculated as 0.3. This cost is due to the NRE resource slot at t = 6. As DC_1 takes least ACO, UR_1 is assigned to it.

Next, UR_2 requires one node at the time slot t = 1 without UN. The UN of RE for this UR is pre-determined as 100%, and NRE is pre-determined as 10%. The UR_2 is first matched with DC_1 as shown in Table 4.4 in which we underline (_) the RE resource slots occupied by that UR. Here, DC_1 can provide one RE resource slot. Therefore, the finish time of UR_2 in DC_1 is determined as 1 + $\lceil(1 \times 100\%)\rceil$ = 2 with UN, and the ACO is calculated as 0.1. This cost is due to the NRE resource slot at t = 2. Then, UR_2 is matched DC_2 as shown in Table 4.4. Like DC_1, DC_2 can provide one RE resource slot. Therefore, the finish time of UR_2 in DC_2 is determined as 1 + $\lceil(1 \times 100\%)\rceil$ = 2. However, the ACO is calculated as 0. As DC_2 takes least ACO, UR_2 is assigned to it.

Next, UR_3 requires one node for the time slots t = 1 to t = 4 without UN. The UN of

Table 4.3: Matching of UR_1 to DC_1 and DC_2 to determine its ACO

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6										
	5										
	4										
	3										
	2										
	1	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$				
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5										
	4										
	3										
	2										
	1	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$				
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

Table 4.4: Matching of UR_2 to DC_1 and DC_2 to determine its ACO

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6										
	5										
	4										
	3										
	2	$\underline{UR_2}$	$\underline{UR_2}$								
	1	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$				
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5										
	4										
	3										
	2										
	1	$\underline{UR_2}$	$\underline{UR_2}$								
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

Table 4.5: Matching of UR_3 to DC_1 and DC_2 to determine its ACO

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6										
	5										
	4										
	3										
	2	$\underline{UR_3}$	UR_3	$\underline{UR_3}$	$\underline{UR_3}$	$\underline{UR_3}$					
	1	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$	$\underline{UR_1}$				
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5										
	4										
	3										
	2	UR_3	UR_3								
	1	$\underline{UR_2}$	$\underline{UR_2}$	$\underline{UR_3}$	$\underline{UR_3}$	$\underline{UR_3}$					
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

RE for this UR is pre-determined as 25%, and NRE is pre-determined as 10%. The UR_3 is first matched with DC_1 as shown in Table 4.5. Here, DC_1 can provide three RE resource slots and one NRE slot. Therefore, the finish time of UR_3 in DC_1 is determined as 4 + $\lceil(3 \times 25\%) + (1 \times 10\%)\rceil$ = 5 with UN and the ACO is calculated as 0.1. This cost is due to the NRE resource slot at t = 2. Then, UR_3 is matched DC_2 as shown in Table 4.5. Like DC_1, DC_2 can provide two RE resource slots and two NRE resource slots. Therefore, the finish time of UR_3 in DC_2 is determined as 4 + $\lceil(2 \times 25\%) + (2 \times 10\%)\rceil$ = 5. However, the ACO is calculated as 0.2. This cost is due to the NRE resource slots at t = 1 and t = 2. As DC_1 takes least ACO, UR_3 is assigned to it.

Next, UR_4 requires 2 nodes for the time slots t = 3 to t = 7 without UN. The UN of RE for this UR is pre-determined as 60%, and NRE is pre-determined as 10%. The UR_4 is first matched with DC_1. Here, DC_1 can provide seven RE resource slots and three NRE resource slots. The UN time is $\lceil(7 \times 60\%) + (3 \times 10\%)\rceil$ = 5 for 2 nodes. Therefore, the finish time of UR_4 in DC_1 is determined as 5 + $\lceil\frac{5}{2}\rceil$ = 8 with UN and the ACO is calculated as 2.3. Then UR_4 is matched DC_2. Here, DC_2 can provide eight RE resource

Table 4.6: Final Gantt chart of assigning the URs to the DCs using UN-FABEF

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6								UR_9		
	5					UR_7		UR_8	UR_9	UR_9	
	4					UR_6	UR_7	UR_8	UR_9	UR_9	
	3				UR_5	UR_5	UR_6	UR_7	UR_8	UR_9	UR_9
	2	UR_3	UR_3	UR_3	UR_3	UR_3	UR_5	UR_6	UR_8	UR_8	UR_9
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_7	UR_8	UR_9
		$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$	$t = 9$	$t = 10$

		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5										
	4										
	3										
	2			UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4
	1	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4
		$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$	$t = 9$	$t = 10$

slots and two NRE resource slots. The UN time is $\lceil(8 \times 60\%) + (2 \times 10\%)\rceil$ = 5 for 2 nodes. Therefore, the finish time of UR_4 in DC_2 is determined as 5 + $\lceil\frac{5}{2}\rceil$ = 8. However, the ACO is calculated as 0.8. As DC_2 takes least ACO, UR_4 is assigned to it. In the similar process, UR_5 to UR_9 are assigned to DC_1, DC_1, DC_1, DC_1 and DC_1, respectively. The $ACOs$ are 0.6, 0.6, 0.6, 0.6 and 0.9, respectively. The OCO of DC_1 and DC_2 is 3.4 and 0.8, respectively. Note that the UNT and UNCO are 12 and 0.5, respectively. The TNRE is 33. The final Gantt chart for UN-FABEF is shown in Table 4.6.

4.2.5.2 Illustration of UN-HAREF

In the UN-HAREF, the first UR (i.e., UR_1) requires one node for the time slots t = 1 to t = 4 (i.e., duration of 4 time slots) without UN. The UN of RE for this UR is pre-determined as 50%, and NRE is pre-determined as 10%. The UR_1 is first matched with DC_1. Here, DC_1 can provide four RE resource slots. Therefore, the finish time of UR_1 in DC_1 is determined as 4 + (4 $\times$ 50%) = 6 with UN in which all the resource slots are RE. Here, the ACO is calculated as 0. Then, UR_1 is matched DC_2. Like DC_1, DC_2 can provide four RE

resource slots. Therefore, the finish time of UR_1 in DC_2 is determined as 4 + (4 $\times$ 50%) = 6, in which five resource slots are RE and one resource slot is NRE. Here, the ACO is calculated as 0.3. As DC_1 gives highest TNRE, UR_1 is assigned to it.

Next, UR_2 requires one node at the time slot t = 1 without UN. The UN of RE for this UR is pre-determined as 100%, and NRE is pre-determined as 10%. The UR_2 is first matched with DC_1. Here, DC_1 can provide one RE resource slot. Therefore, the finish time of UR_2 in DC_1 is determined as 1 + (1 $\times$ 100%) = 2 with UN in which one resource slot is RE, and another resource slot is NRE. Here, the ACO is calculated as 0.1. Then, UR_2 is matched DC_2. Like DC_1, DC_2 can provide one RE resource slot. Therefore, the finish time of UR_2 in DC_2 is determined as 1 + (1 $\times$ 100%) = 2, in which two resource slots are RE. Here, the ACO is calculated as 0. As DC_2 gives highest TNRE, UR_2 is assigned to it.

Next, UR_3 requires one node for the time slots t = 1 to t = 4 without UN. The UN of RE for this UR is pre-determined as 25%, and NRE is pre-determined as 10%. The UR_3 is first matched with DC_1. Here, DC_1 can provide three RE resource slots and one NRE slot. Therefore, the finish time of UR_3 in DC_1 is determined as 4 + (3 $\times$ 25%) + (1 $\times$ 10%) = 5 with UN in which four resource slots are RE, and one resource slot is NRE. Here, the ACO is calculated as 0.1. Then, UR_3 is matched DC_2. Like DC_1, DC_2 can provide two RE resource slots and two NRE resource slots. Therefore, the finish time of UR_3 in DC_2 is determined as 4 + (2 $\times$ 25%) + (2 $\times$ 10%) = 5, in which three resource slots are RE and two resource slots are NRE. Here, the ACO is calculated as 0.2. As DC_1 gives highest TNRE, UR_3 is assigned to it.

Next, UR_4 requires 2 nodes for the time slots t = 3 to t = 7 without UN. The UN of RE for this UR is pre-determined as 60%, and NRE is pre-determined as 10%. The UR_4 is first matched with DC_1. Here, DC_1 can provide seven RE resource slots and three NRE resource slots. The UN time is $\lceil(7 \times 60\%) + (3 \times 10\%)\rceil$ = 5 for 2 nodes. Therefore, the finish time of UR_4 in DC_1 is determined as 5 + $\lceil\frac{5}{2}\rceil$ = 8 with UN in which twelve resource slots are RE, and four resource slots are NRE. Here, the ACO is calculated as 2.3. Then, UR_4 is matched DC_2. Here, DC_2 can provide eight RE resource slots and two NRE resource slots. The UN time is $\lceil(8 \times 60\%) + (2 \times 10\%)\rceil$ = 5 for 2 nodes. Therefore, the finish time of UR_4 in DC_2 is determined as 5 + $\lceil\frac{5}{2}\rceil$ = 8, in which twelve resource slots are

RE and four resource slots are NRE. Here, the ACO is calculated as 0.8. Both DC_1 and DC_2 give the same TNRE, UR_4 is assigned to DC_2 based on least ACO.

In the similar process, UR_5 to UR_9 are assigned to DC_1, DC_1, DC_1, DC_1 and DC_1, respectively. The ACOs are 0.6, 0.6, 0.6, 0.6 and 0.9, respectively. The OCO of DC_1 and DC_2 is 3.4 and 0.8, respectively. Note that the UNT and UNCO are 12 and 0.5, respectively. The TNRE is 33. The final Gantt chart for UN-HAREF is shown in Table 4.7.

4.2.5.3 Illustration of UN-RR

In the UN-RR, the first UR (i.e., UR_1) is assigned to DC_1 as 1 % 2 = 1. Here, DC_1 can provide four RE resource slots. However, the finish time of UR_1 in DC_1 is determined as 4 + $\lceil(4 \times 50\%)\rceil$ = 6 with UN, and the ACO is calculated as 0.

Next, UR_2 is assigned to DC_2. Here, DC_2 can provide one RE resource slot. However, the finish time of UR_2 in DC_2 is determined as 1 + $\lceil(1 \times 100\%)\rceil$ = 2 with UN, and the ACO is calculated as 0. Then, UR_3 is assigned to DC_1 as 3 % 2 = 1. Here, DC_1 can provide three RE resource slots and one NRE resource slot. However, the finish time of UR_3 in DC_1 is determined as 4 + $\lceil(3 \times 25\%) + (1 \times 10\%)\rceil$ = 5 with UN and the ACO is calculated as 0.1. This cost is due to the NRE resource slot at t = 2.

Next, UR_4 is assigned to DC_2. Here, DC_2 can provide two NRE resource slots and eight RE resource slots. The UN time is $\lceil(8 \times 60\%) + (2 \times 10\%)\rceil$ = 5 for 2 nodes. Therefore, the finish time of UR_4 in DC_2 is determined as 5 + $\lceil\frac{5}{2}\rceil$ = 8 with UN and the ACO is calculated as 0.8.

In the similar process, UR_5 to UR_9 are assigned to DC_1, DC_2, DC_1, DC_2 and DC_1, respectively. The ACOs are 0.6, 0.9, 0.6, 2.0 and 0.7, respectively. The OCO of DC_1 and DC_2 is 2.0 and 3.7, respectively. Note that the UNT and UNCO are 12 and 0.5, respectively. The TNRE is 32. The final Gantt chart for UN-RR is shown in Table 4.8.

Table 4.7: Final Gantt chart of assigning the URs to the DCs using UN-HAREF

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6								UR_9		
	5					UR_7		UR_8	UR_9	UR_9	
	4					UR_6	UR_7	UR_8	UR_9	UR_9	
	3				UR_5	UR_5	UR_6	UR_7	UR_8	UR_9	UR_9
	2	UR_3	UR_3	UR_3	UR_3	UR_3	UR_5	UR_6	UR_8	UR_8	UR_9
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_7	UR_8	UR_9
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5										
	4										
	3										
	2			UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4
	1	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

Table 4.8: Final Gantt chart of assigning the URs to the DCs using UN-RR

		0.1	0.1	0.3	0.4	0.4	0.2	0.2	0.1	0.1	0.3
DC_1	7										
	6										
	5										
	4					UR_7			UR_9		
	3				UR_5	UR_5	UR_7		UR_9	UR_9	UR_9
	2	UR_3	UR_3	UR_3	UR_3	UR_3	UR_5	UR_7	UR_9	UR_9	UR_9
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_7	UR_9	UR_9
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

		0.1	0.1	0.5	0.3	0.2	0.3	0.4	0.5	0.1	0.3
DC_2	7										
	6										
	5							UR_8			
	4							UR_8	UR_8	UR_8	
	3					UR_6	UR_6	UR_6	UR_8	UR_8	
	2			UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4
	1	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$

4.3 Performance Metrics, Simulation Configuration and Simulation Results

This section discusses the performance metrics used to evaluate the proposed algorithms, the simulation configuration and the simulation results.

4.3.1 Performance Metrics

We use four performance metrics, namely OCO, TNRE, UNT and UNCO. The OCO is the sum of the cost incurred for using RE and NRE resources in all the datacenters. However, the cost of RE resources is assumed to be negligible. Mathematically,

$$OCO = \sum_{k=1}^{n} \sum_{i=1}^{m} ACO[i] \times X[k,i] \tag{4.2}$$

$$\text{where } X[k,i] = \begin{cases} 1 & \text{if } UR_k \text{ is assigned to } DC_i \\ 0 & \text{Otherwise} \end{cases}$$

Note that resources are not explicitly shown here. The UNT is the sum of the extended time duration of URs to fulfil their UN. Mathematically,

$$UNT = \sum_{k=1}^{n} UN[k] \tag{4.3}$$

The $UNCO$ is the sum of the cost incurred for using RE and NRE resources in the UN time duration. Note that the time duration is between ST + D and ST + D + $\lceil \frac{UN}{N} \rceil$. Mathematically,

$$UNCO = \sum_{k=1}^{n} \sum_{i=1}^{m} \sum_{l=ST[k]+D[k]}^{ST[k]+D[k]+\lceil \frac{UN[k]}{N[k]} \rceil} CO[l] \times X[k,i] \tag{4.4}$$

The $TNRE$ is the sum of RE resource slots that are assigned to the URs.

$$TNRE = \sum_{k=1}^{n} slots\_re[k] \tag{4.5}$$

4.3.2 Simulation Configuration

The simulation is carried out in a system with the following configuration. 1) System processor: Intel(R) Core(TM) i3-7020U CPU @ 2.30 GHz 2.30 GHz 2) Installed system RAM: 4.00 GB 3) System type: x64-based processor 4) System operating system: 64-bit 5) Windows edition: Windows 10 Home 6) System software: MATLAB R2021a. As there are no benchmark datasets, we generate datasets by taking the number of URs in the range of [200 $\sim$ 2000] and the number of datacenters in the range of [20 $\sim$ 200]. Alternatively, we generate ten datasets, namely 200 $\times$ 20, 400 $\times$ 40, 600 $\times$ 60, 800 $\times$ 80, 1000 $\times$ 100, 1200 $\times$ 120, 1400 $\times$ 140, 1600 $\times$ 160, 1800 $\times$ 180 and 2000 $\times$ 200 using uniformly distributed pseudorandom integers. Note the first values (i.e., 200, 400, 600,. . ., 2000) represent the number of URs, and the second values represent the number of datacenters. In each dataset, there are five instances, namely i1 to i5. These instances are generated in the same range. The results of these instances are averaged and shown as the result of the dataset. We consider 10 to 400 resources per datacenter. It indicates that the number of resources in the dataset ranges from 200 to 80000. The cost of using NRE resources is generated in the range of [1 $\sim$ 100], and the cost of RE resources is assumed as 0. The parameters of URs are configured as follows. 1) ST: [1 $\sim$ 100] 2) D: [10 $\sim$ 25] 3) N: [10 $\sim$ 100] 4) UN-RE: [10 $\sim$ 90] 5) UN-NRE: 10%. It is noteworthy to mention that all these parameters are combined together and given in a single file for all the URs. In a similar fashion, all the parameters associated with the datacenters are given in another file. Each algorithm takes these files to produce the simulation results using four performance metrics.

4.3.3 Simulation Results

In the simulation process, the results of the proposed algorithms are carried out using the generated datasets. The OCO of the UN-FABEF, UN-HAREF, and UN-RR algorithms is

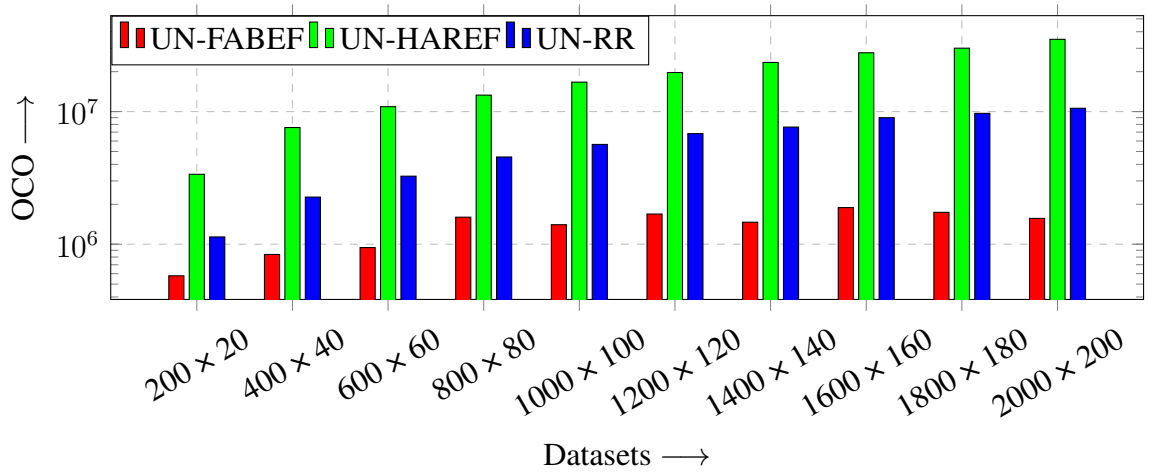

Figure 4.1: Pictorial performance comparison for UN-FABEF, UN-HAREF and UN-RR algorithms in terms of OCO.

compared and shown in Figure 4.1. Here, the x-axis represents the datasets, and the y-axis represents the overall cost on the logarithmic scale. As seen in the figure, UN-FABEF achieves the least OCO compared to UN-HAREF and UN-RR. The rationality behind this performance is that it assigns the URs to the least ACO datacenter. On the contrary, UN-HAREF performs poorly once the RE resource slots are occupied. Therefore, the OCO of UN-HAREF is drastically increased. It is important to notice that UN-RR achieves better performance than UN-HAREF, even if the scheduling is performed in a circular fashion.

UN-FABEF, UN-HAREF, and UN-RR are compared using the TNRE (Fig. 4.2). It is noticeable that UN-HAREF performs better than UN-FABEF and UN-RR as it assigns the UR to the datacenter that provides the highest TNRE. However, UN-FABEF also performs closely with UN-HAREF. On the contrary, UN-RR performs poorly as it circularly assigns the UR without looking into TNRE and TNNRE.

In the UNT matrix, the UN-FABEF and UN-RR perform very closely. However, the performance of UN-HAREF is poor as UNT is directly proportional to the RE resource slots. Alternatively, if the number of RE slots is increased, then the UNT is also increased, and we know that UN-HAREF uses more TNRE compared to other algorithms.

The UNT and UNCO of the UN-FABEF, UN-HAREF, and UN-RR algorithms are compared and shown in Figure 4.3 and Figure 4.4, respectively. Like OCO, the performance of UN-FABEF, UN-HAREF, and UN-RR algorithms remains the same in the UNCO. Specifi-

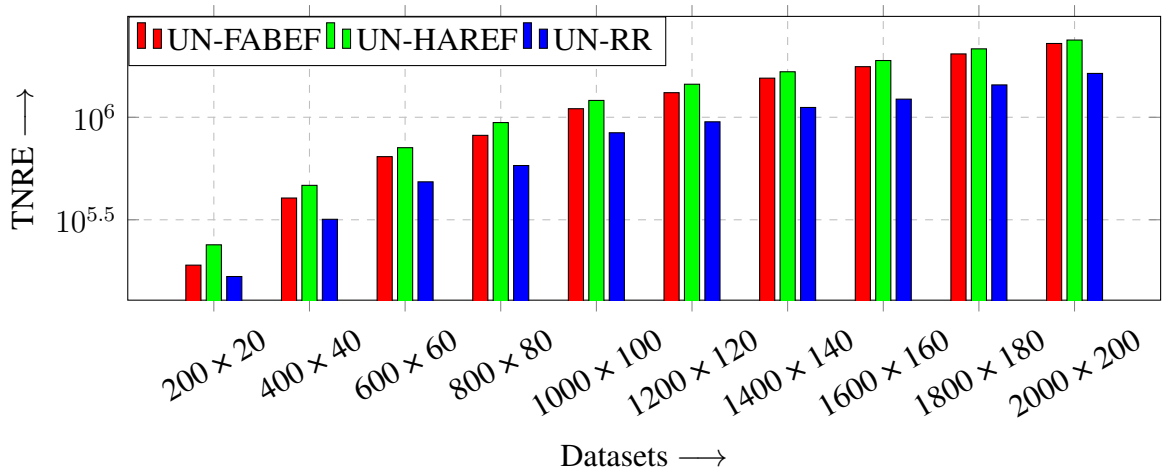

Figure 4.2: Pictorial performance comparison for UN-FABEF, UN-HAREF and UN-RR algorithms in terms of TNRE.

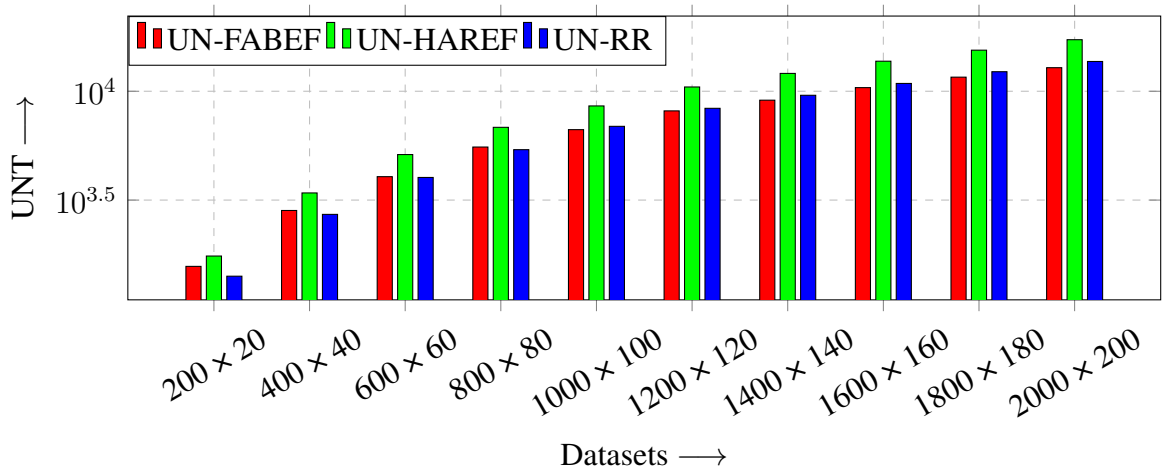

Figure 4.3: Pictorial performance comparison for UN-FABEF, UN-HAREF and UN-RR algorithms in terms of UNT.

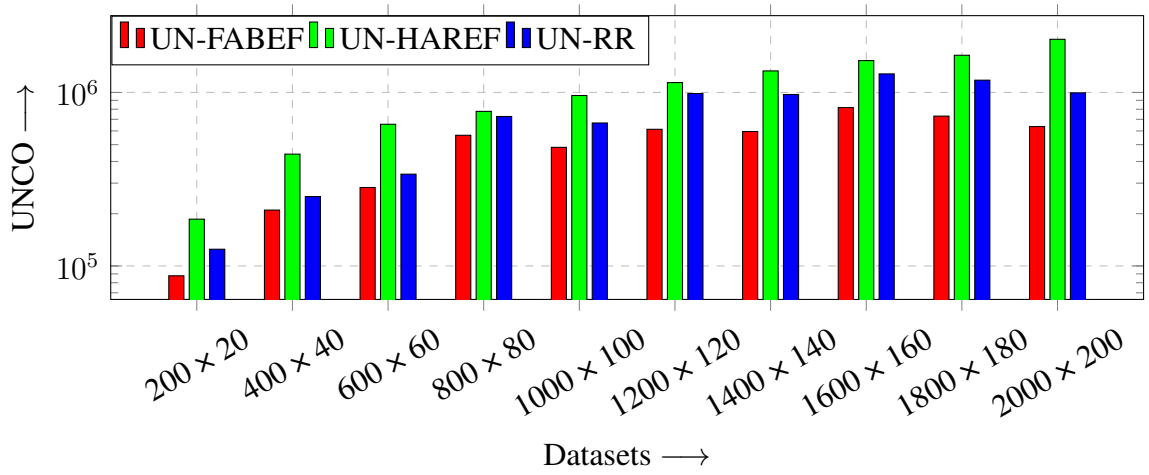

Figure 4.4: Pictorial performance comparison for UN-FABEF, UN-HAREF and UN-RR algorithms in terms of UNCO.

cally, UN-FABEF performs better than UN-HAREF and UN-RR even if the UNT duration is assigned based on the availability of any resource slots without looking into the RE and NRE resource slots.

4.4 Summary

In this chapter, we have presented three RE-based scheduling algorithms, namely UN-FABEF, UN-HAREF, and UN-RR. The aim of these algorithms is to minimize the OCO, UNT and UNCO, and maximize the TNRE. The UN is modelled in terms of the percentage of UN-RE and UN-NRE, in which UN-RE is considered as variable, and UN-NRE is fixed. Further, UNT is determined based on the number of RE and NRE resource slots. The complexity analysis of UN-FABEF, UN-HAREF, and UN-RR algorithms is performed and shown as $O(nmdo)$, $O(nmdo)$, and $O(ndo)$, respectively. These algorithms are rigorously compared using four performance metrics by taking ten different datasets with fifty different instances. The simulation results show that UN-FABEF performs better in the OCO, UNT, and UNCO, whereas UN-HAREF performs better in the TNRE. However, the proposed algorithms do not model UN from the user and CSP perspective. We can model them by defining different levels from the user's perspective and UN percentages from the CSP's perspective. Therefore, in the next chapter, we explore UN from both perspectives

and develop UN-level-based algorithms by managing UN for geo-distributed datacenters.

Chapter 5

Uncertainty Level-Based Algorithms by Managing Renewable Energy for Geo-Distributed Datacenters

This chapter aims to solve the problem of assigning n URs to m datacenters that are powered by both RE and NRE sources. This problem also considers the UN of the resources of datacenters. We extend three benchmark algorithms, namely FABEF, HAREF and RR, to solve this problem. These extended algorithms are named as UNL-FABEF, UNL-HAREF and UNL-RR. These algorithms model the UN from the user and CSP perspectives. From the user's perspective, it models the request as a UNL and percentage for RE and NRE resources, respectively. The UNL is considered as low, medium and high, respectively, whereas the UN percentage is between 1% to 100%, respectively. On the other hand, from the CSP perspective, the UN is modelled with respect to the time instance.

UNL-FABEF aims to minimize the OCO of datacenters by considering the UNCO. UNL-HAREF seeks to maximize available RE usage of datacenters, including the UN period. UNL-RR assigns the datacenters to the URs in a roundabout fashion without looking into the OCO and available RE of datacenters. Therefore, UNL-MOSA is introduced by taking advantage of UNL-FABEF and UNL-HAREF. Alternatively, UNL-MOSA forms a balance between the OCO and the available RE usage. Like UNL-FABEF, UNL-HAREF and UNL-RR, UNL-MOSA models the UN from the user and CSP perspectives and the

UNL. The three extended algorithms and UNL-MOSA are implemented and compared among them. The rationality behind this comparison is that no algorithm deals with UNL in the literature. These algorithms are tested by generating ten datasets. Each dataset has five instances, and the number of URs and datacenters of datasets is 200 to 2000 and 20 to 200, respectively. All four algorithms are thoroughly compared using five performance metrics: the OCO, TNRE, TNNRE, UNT and UNCO to demonstrate their applicability. We use resources and resource slots interchangeably without loss of generality throughout this chapter. The uniqueness of this chapter is listed as follows.

1. We develop four scheduling algorithms for geo-distributed datacenters in which UN is modelled from the user and CSP perspectives. The user's perspective categorizes the UNL into low, medium, and high, whereas the CSP's perspective presents the UN between 1% to 100%.

2. The UN of RE generation is modelled as dynamic, whereas NRE generation is fixed as per their relevance in real-life scenarios.

3. UNL-MOSA balances the performance of UNL-FABEF and UNL-HAREF using their objectives' linear combination.

4. The proposed algorithms are examined using ten datasets and five performance metrics to show their supremacy and applicability.

The next sections are outlined as follows. Section 5.1 presents the system model and the problem statement. Section 5.2 introduces the four proposed algorithms with their illustration and time complexity analysis. Section 5.3 defines five performance metrics, and discusses the datasets and simulation results. Section 5.4 concludes the presented work.

5.1 System Model and Problem Statement

This section discusses the system model focusing on RE with UN, followed by the problem statement addressed in this chapter.

5.1.1 System Model

We consider a CSP that deploys a set of datacenters around the globe. Each datacenter contains a set of homogeneous servers. Each server is powered by either RE or NRE sources. The UN of these sources varies over time instance. However, RE's UN is quite more than NRE. On the other hand, RE costs less than NRE. As a result, our system tries to fulfil the requirements of users using resources powered by RE followed by NRE. Without loss of generality, a datacenter has unlimited resources to accommodate the requirements of the URs. However, the cost of resources in each datacenter is different with respect to the time instance.

Our system model enforces the UN in user and CSP perspectives. As the UN of RE resources varies over time, the user can select a UNL (i.e., low, medium and high) for RE resources while submitting the request based on its importance. The UNL is 1% to 30% in low, 31% to 60% in medium and 61% to 100% in high. Suppose a user selects a UNL of low (or medium or high). In that case, the CSP assigns the RE resources with less than or equal to 30% (or 60% or 100%) UN in the given time interval. Otherwise, NRE resources can be assigned.

We assume that the UN of RE and NRE resources is variable (i.e., 1% to 100%) and static (say, 10%), respectively. The UN value is calculated once the RE and NRE resources are estimated. Note that the value is calculated by taking the floor of the average value of the UNLs, i.e., 15% ($\lfloor \frac{1+30}{2} \rfloor$), 45% ($\lfloor \frac{31+60}{2} \rfloor$) and 80% ($\lfloor \frac{61+100}{2} \rfloor$), respectively, for RE resources and 10% for NRE resources. Based on the UN value, the duration of each UR is recalculated. The above system model is expanded using the models presented in [46, 63, 64, 115].

5.1.2 Problem Formulation

Consider an ordered set UR of n URs and a set DC of m datacenters. A UR $UR_k \in UR$ is characterized by its unique ID, ST, D, N, UNL of RE ($UNL\text{-}RE$) and UN percentage of

NRE. The UNL is defined as follows.

$$UNL = \begin{cases} 1 & \text{If UN percentage is between } 1\% \text{ and } 30\% \\ 2 & \text{If UN percentage is between } 31\% \text{ and } 60\% \\ 3 & \text{Otherwise} \end{cases} \tag{5.1}$$

The UN of RE and NRE is variable (i.e., 1% to 100%) and fixed (say, 10%), respectively. A datacenter $DC_i \in DC$ is characterized by its unique ID, o, a sequence of RE and NRE resources, a cost sequence of RE and NRE resources, and a UN percentage sequence of RE and NRE resources with respect to a time window. It is noteworthy to mention that the resources in one datacenter differ from other datacenters. Mathematically, $|DC_i| \neq |DC_{i'}|$, $i \neq i'$ and $\{DC_i, DC_{i'}\} \in DC$. The problem is finding a many-to-one function between the n URs and m datacenters to achieve the following goals.

1. Minimize the OCO of executing the n URs in m datacenters.
2. Maximize the $TNRE$ of m datacenters for executing the n URs.
3. Minimize the UNT and $UNCO$ of n URs.

The problem is framed with some conditions as follows.

1. A UR, $UR_k \in UR$, can only be assigned to one datacenter.
2. The required number of resources of UR, $UR_k \in UR$, can be fulfilled by mixing RE and NRE resources.
3. A UR, $UR_k \in UR$, can only be assigned to a RE resource if its UNL is satisfied.
4. A UR, $UR_k \in UR$, cannot migrate from one datacenter to another datacenter to avail the RE resources.

5.2 Proposed Scheduling Algorithms

We present four UNL-based algorithms, UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA, by managing RE for geo-distributed datacenters. These UNL-based algorithms

goal to minimize the OCO of datacenters, the UNT and UNCO of URs, and maximize the TNRE. The main principle of these algorithms is to model the UN from both user and CSP perspectives. The pseudo-codes of these algorithms are presented in Algorithm 5.1 to Algorithm 5.4.

5.2.1 UNL-FABEF

UNL-FABEF aims to minimize the OCO of datacenters. The algorithm checks the availability of UR in the URQ (Line 1). If the URQ is not empty, then it picks a UR from the URQ and determines its $UNL\text{-}RE$ (Line 3 and Line 4). Then UNL-FABEF checks the datacenter individually to determine the ACO (Line 5). In each datacenter, it checks the suitable RE and NRE resource slots from the ST of a UR to the ST + D - 1 of a UR (Line 7). Now, UNL-FABEF calls Procedure 3 to determine the resource slots.

Algorithm 5.1 UNL-FABEF

Input: URQ, n, $UNL\text{-}RE$, m, ST, D, o, R, $UNL\text{-}R$, ACO, CO, N and $UN\text{-}NRE$
Output: OCO, $TNRE$, $TNNRE$, UNT and $UNCO$

1: **while** URQ is not $NULL$ **do**
2: Set $OCO \leftarrow 0$, $TNRE \leftarrow 0$, $TNNRE \leftarrow 0$, $UNT \leftarrow 0$ and $UNCO \leftarrow 0$
3: **for** $k \leftarrow 1$ to n **do**
4: Determine $UNL\text{-}RE[k]$
5: **for** $i \leftarrow 1, 2, 3, \ldots, m$ **do**
6: Set $ACO[i] \leftarrow 0$, $UN\text{-}COST[i] \leftarrow 0$, $UN\text{-}TIME[i] \leftarrow 0$, $slots\_re[i] \leftarrow 0$ and $slots\_nre[i] \leftarrow 0$
7: **for** $l \leftarrow ST[k], ST[k] + 1, ST[k] + 2, \ldots, ST[k] + D[k] - 1$ **do**
8: Call $DETERMINE\text{-}RESOURCE\text{-}SLOTS(o, R, l, UNL\text{-}R, UNL\text{-}RE, k, slots\_re, i, ACO, CO, slots\_nre, N)$
9: **end for**
10: Call $DETERMINE\text{-}UN\text{-}TIME(UNL\text{-}RE, k, slots\_re, i, \tau_1, slots\_nre, UN\text{-}NRE, \tau_2, \tau_3, ST, D, N, UN\text{-}T, o, R, UNL\text{-}R, ACO, i, CO)$
11: **end for**
12: Determine the best datacenter i' by calculating $min(ACO)$
13: The datacenter i' is assigned with the UR k.
14: Set $OCO \leftarrow OCO + ACO[i']$, $TNRE \leftarrow TNRE + slots\_re[i']$, $TNNRE \leftarrow TNNRE + slots\_nre[i']$, $UNT \leftarrow UNT + UN\text{-}TIME[i']$ and $UNCO \leftarrow UNCO + UN\text{-}COST[i']$
15: **end for**
16: **end while**

Procedure 3 checks the resource slot individually that is not assigned to any UR and matches the UNL of resource with UNL of UR (Line 2 and Line 3). If both conditions are satisfied, it counts that resource slot (Line 4). Further, it checks the RE and NRE power

Procedure 3 *DETERMINE-RESOURCE-SLOTS*(o, R, l, $UNL\text{-}R$, $UNL\text{-}RE$, k, $slots\_re$, i, ACO, CO, $slots\_nre$, N)

```
1: Set slots ← 0
2: for j ← 1, 2, 3,. . ., o do
3:     if R[l, j] is unassigned to any UR and UNL-R[j] = UNL-RE[k] then
4:         slots ← slots + 1
5:         if R[l, j] is taking power from the RE sources then
6:             Set slots_re[i] ← slots_re[i] + 1 and ACO[i] ← ACO[i] + CO[l]
7:         else
8:             Set slots_nre[i] ← slots_nre[i] + 1 and ACO[i] ← ACO[i] + CO[l]
9:         end if
10:    end if
11:    if N[k] = slots then
12:        break
13:    end if
14: end for
15: return
```

supply to that resource slot and calculates the ACO (Line 5 to Line 9). Once the resource slot requirement of a UR is satisfied, Procedure 3 breaks the loop (Line 11 to Line 13). The above process is repeated for the entire duration of UR. At last, Procedure 3 returns in line 15 to the main algorithm.

UNL-FABEF calls the Procedure 4 to determine the UNT of a UR (Line 10). Procedure 4 checks the UNL and accordingly calculates the UN (Line 1 to Line 9). Once UN is determined, this procedure calls Procedure 5 to select the resource slots for the UN period (Line 10 to Line 12) and calculates the UNT (Line 11) and cost. At last, Procedure 4 returns in line 14 to the main algorithm.

Procedure 5 finds the resource slots and UNCO. It is similar to Procedure 3 except for the UNCO calculation in line 6 and line 8. Now, UNL-FABEF contains the ACO of a UR in each datacenter. Therefore, it determines the best datacenter by calculating the minimum ACO (Line 12 of Algorithm 5.1). Finally, it assigns that UR to the selected datacenter (Line 13) and updates OCO, TNRE, TNNRE, UNT and UNCO (Line 14).

5.2.2 UNL-HAREF

UNL-HAREF aims to maximize the TNRE of datacenters. The algorithm of UNL-HAREF (Algorithm 5.2) is similar to the algorithm of UNL-FABEF (Algorithm 5.1). The main difference is seen in line 12 in which the best datacenter is determined by calculating the

Procedure 4 $DETERMINE\text{-}UN\text{-}TIME(UNL\text{-}RE, k, slots\_re, i, \tau_1, slots\_nre, UN\text{-}NRE, \tau_2, \tau_3, ST, D, N, UN\text{-}TIME, o, R, UNL\text{-}R, ACO, i, CO)$

1: **if** $UNL\text{-}RE[k]$ = 1 **then**
2: Set $UN[k] \leftarrow \lceil slots\_re[i] \times \tau_1 + slots\_nre[i] \times UN\text{-}NRE[k] \rceil$
3: **else**
4: **if** $UNL\text{-}RE[k]$ = 2 **then**
5: Set $UN[k] \leftarrow \lceil slots\_re[i] \times \tau_2 + slots\_nre[i] \times UN\text{-}NRE[k] \rceil$
6: **else**
7: Set $UN[k] \leftarrow \lceil slots\_re[i] \times \tau_3 + slots\_nre[i] \times UN\text{-}NRE[k] \rceil$
8: **end if**
9: **end if**
10: **for** $l \leftarrow ST[k] + D[k], ST[k] + D[k] + 1, \ldots, ST[k] + D[k] + \lceil \frac{UN[k]}{N[k]} \rceil$ **do**
11: Set $UN\text{-}TIME[i] \leftarrow UN\text{-}TIME[i] + \lceil \frac{UN[k]}{N[k]} \rceil$
12: Call $DETERMINE\text{-}RESOURCE\text{-}SLOTS\text{-}AND\text{-}UN\text{-}COST(o, R, l, UNL\text{-}R, UNL\text{-}RE, k, slots\_re, i, ACO, CO, UN\text{-}COST, slots\_nre, N)$
13: **end for**
14: **return**

maximum $slots\_re$ instead of the minimum ACO.

5.2.3 UNL-RR

UNL-RR follows the circular order for assigning the URs to the datacenters. Like UNL-FABEF and UNL-HAREF, it checks the URQ, selects a UR and determines the $UNL\text{-}RE$ (Line 1 to Line 4). Then UNL-RR selects a datacenter for a UR in a circular order (Line 5 to Line 9) and checks the suitable RE and NRE resource slots using Procedure 3 in that datacenter based on the UR requirement (Line 11 to Line 13). Further, it calls the Procedure 4 to determine the UNT, and thereby assign the UR to that datacenter and updates OCO, TNRE, TNNRE, UNT and UNCO (Line 14 to Line 16).

5.2.4 UNL-MOSA

UNL-MOSA aims to minimize the OCO of datacenters as well as maximize the TNRE of datacenters. It makes a trade-off between UNL-FABEF and UNL-HAREF. The process of UNL-MOSA is similar to UNL-FABEF and UNL-HAREF till the determination of resource slots (Procedure 3) and UNT (Procedure 4) (Line 1 to Line 10). Note that Procedure 3 determines the ACO and Procedure 4 calls Procedure 5 to determine the updated ACO and $slots\_re$. Then, UNL-MOSA calls Procedure 6 to determine the multi-objective

Procedure 5 $DETERMINE\text{-}RESOURCE\text{-}SLOTS\text{-}AND\text{-}UN\text{-}COST(o,\ R,\ l,\ UNL\text{-}R, UNL\text{-}RE, k, slots\_re, i, ACO, CO, UN\text{-}COST, slots\_nre, N)$

1: Set $slots \leftarrow 0$
2: **for** $j \leftarrow 1, 2, 3, \ldots, o$ **do**
3: **if** $R[l, j]$ is unassigned to any UR and $UNL\text{-}R[j] = UNL\text{-}RE[k]$ **then**
4: Set $slots \leftarrow slots + 1$
5: **if** $R[l, j]$ is taking power from the RE sources **then**
6: Set $slots\_re[i] \leftarrow slots\_re[i] + 1$, $ACO[i] \leftarrow ACO[i] + CO[l]$ and $UN\text{-}COST[i] \leftarrow UN\text{-}COST[i] + CO[l]$
7: **else**
8: Set $slots\_nre[i] \leftarrow slots\_nre[i] + 1$, $ACO[i] \leftarrow ACO[i] + CO[l]$ and $UN\text{-}COST[i] \leftarrow UN\text{-}COST[i] + CO[l]$
9: **end if**
10: **end if**
11: **if** $N[k] = slots$ **then**
12: break
13: **end if**
14: **end for**
15: **return**

Algorithm 5.2 UNL-HAREF

Input: URQ, n, $UNL\text{-}RE$, m, ST, D, o, R, $UNL\text{-}R$, ACO, CO, N and $UN\text{-}NRE$
Output: OCO, $TNRE$, $TNNRE$, UNT and $UNCO$

1: **while** URQ is not $NULL$ **do**
2: Set $OCO \leftarrow 0$, $TNRE \leftarrow 0$, $TNNRE \leftarrow 0$, $UNT \leftarrow 0$ and $UNCO \leftarrow 0$
3: **for** $k \leftarrow 1$ to n **do**
4: Step 4 to Step 11 is same as Algorithm 5.1
12: Determine the best datacenter i' by calculating $max(slots\_re)$
13: The datacenter i' is assigned with the UR k.
14: Set $OCO \leftarrow OCO + ACO[i']$, $TNRE \leftarrow TNRE + slots\_re[i']$, $TNNRE \leftarrow TNNRE + slots\_nre[i']$, $UNT \leftarrow UNT + UN\text{-}TIME[i']$ and $UNCO \leftarrow UNCO + UN\text{-}COST[i']$
15: **end for**
16: **end while**

function value in each datacenter (Line 12). This value is determined by normalizing the ACO and reciprocal of $slots\_re$, using their maximum and minimum values, respectively, so they are not dominating each other (Line 1 to Line 3 of Procedure 6). The rationality behind this is that ACO needs to be minimized, and $slots\_re$ needs to be maximized, and we convert the maximization to the minimization problem. After normalization, the linear combination of weighted ACO and weighted $slots\_re$ (i.e., f) is determined for each datacenter (Line 5.4). At last, Procedure 6 returns in line 6 to the main algorithm. UNL-MOSA determines the best datacenter by calculating the minimum f (Line 13). Finally, it assigns that UR to the selected datacenter (Line 14) and updates OCO, $TNRE$, $TNNRE$, UNT and $UNCO$ (Line 15).

Algorithm 5.3 UNL-RR

Input: URQ, n, $UNL\text{-}RE$, m, ST, D, o, R, $UNL\text{-}R$, ACO, CO, N and $UN\text{-}NRE$
Output: OCO, $TNRE$, $TNNRE$, UNT and $UNCO$

1: **while** URQ is not $NULL$ **do**
2: Set $OCO \leftarrow 0$, $TNRE \leftarrow 0$, $TNNRE \leftarrow 0$, $UNT \leftarrow 0$ and $UNCO \leftarrow 0$
3: **for** $k \leftarrow 1$ to n **do**
4: Determine $UNL\text{-}RE[k]$
5: **if** k is a multiple of m **then**
6: Set $i \leftarrow m$
7: **else**
8: Set $i \leftarrow k \; mod \; m$
9: **end if**
10: Set $ACO[i] \leftarrow 0$, $UN\text{-}COST[i] \leftarrow 0$, $UN\text{-}TIME[i] \leftarrow 0$, $slots\_re[i] \leftarrow 0$ and $slots\_nre[i] \leftarrow 0$
11: **for** $l \leftarrow ST[k]$, $ST[k] + 1$, $ST[k] + 2$,. . ., $ST[k] + D[k] - 1$ **do**
12: Call $DETERMINE\text{-}RESOURCE\text{-}SLOTS(o, R, l, UNL\text{-}R, UNL\text{-}RE, k, slots\_re, i, ACO, CO, slots\_nre, N)$
13: **end for**
14: Call $DETERMINE\text{-}UNCERTAINTY\text{-}TIME(UNL\text{-}RE, k, slots\_re, i, \tau_1, slots\_nre, UN\text{-}NRE, \tau_2, \tau_3, ST, D, N, UN\text{-}TIME, o, R, UNL\text{-}R, ACO, i, CO)$
15: The datacenter i is assigned with the UR k.
16: Set $OCO \leftarrow OCO + ACO[i]$, $TNRE \leftarrow TNRE + slots\_re[i]$, $TNNRE \leftarrow TNNRE + slots\_nre[i]$, $UNT \leftarrow UNT + UN\text{-}TIME[i]$ and $UNCO \leftarrow UNCO + UN\text{-}COST[i]$
17: **end for**
18: **end while**

Algorithm 5.4 UNL-MOSA

Input: URQ, n, $UNL\text{-}RE$, m, ST, D, o, R, $UNL\text{-}R$, ACO, CO, N, $UN\text{-}NRE$, λ_1 and λ_2
Output: OCO, $TNRE$, $TNNRE$, UNT and $UNCO$

1: **while** URQ is not $NULL$ **do**
2: Set $OCO \leftarrow 0$, $TNRE \leftarrow 0$, $TNNRE \leftarrow 0$, $UNT \leftarrow 0$ and $UNCO \leftarrow 0$
3: **for** $k \leftarrow 1$ to n **do**
4: Determine $UNL\text{-}RE[k]$
5: **for** $i \leftarrow 1, 2, 3, \ldots, m$ **do**
6: Set $ACO[i] \leftarrow 0$, $UN\text{-}COST[i] \leftarrow 0$, $UN\text{-}TIME[i] \leftarrow 0$, $slots\_re[i] \leftarrow 0$ and $slots\_nre[i] \leftarrow 0$
7: **for** $l \leftarrow ST[k]$, $ST[k] + 1$, $ST[k] + 2$,. . ., $ST[k] + D[k] - 1$ **do**
8: Call $DETERMINE\text{-}RESOURCE\text{-}SLOTS(o, R, l, UNL\text{-}R, UNL\text{-}RE, k, slots\_re, i, ACO, CO, slots\_nre, N)$
9: **end for**
10: Call $DETERMINE\text{-}UN\text{-}TIME(UNL\text{-}RE, k, slots\_re, i, \tau_1, slots\_nre, UN\text{-}NRE, \tau_2, \tau_3, ST, D, N, UN\text{-}TIME, o, R, UNL\text{-}R, ACO, i, CO)$
11: **end for**
12: Call $MOSA(ACO, slots\_re, m, \lambda_1, \lambda_2)$
13: Determine the best datacenter i' by calculating $min(f)$
14: The datacenter i' is assigned with the UR k.
15: Set $OCO \leftarrow OCO + ACO[i']$, $TNRE \leftarrow TNRE + slots\_re[i']$, $TNNRE \leftarrow TNNRE + slots\_nre[i']$, $UNT \leftarrow UNT + UN\text{-}TIME[i']$ and $UNCO \leftarrow UNCO + UN\text{-}COST[i']$
16: **end for**
17: **end while**

Procedure 6 $MOSA(ACO, slots\_re, m, \lambda_1, \lambda_2)$

1: Determine $max(ACO)$ and $min(slots\_re)$
2: **for** $i \leftarrow 1, 2, 3, \ldots, m$ **do**
3: Set $NACO[i] \leftarrow \frac{ACO[i]}{max(ACO)}$ and $Nslots\_re[i] \leftarrow \frac{\frac{1}{slots\_re[i]}}{\frac{1}{min(slots\_re)}}$
4: Set $f[i] \leftarrow \lambda_1 \times NACO[i] + (1 - \lambda_1) \times Nslots\_re[i]$
5: **end for**
6: **return**

Table 5.1: Time complexity of the proposed algorithms

Time complexity							
Line	UNL-FABEF	Line	UNL-HAREF	Line	UNL-RR	Line	UNL-MOSA
1-16	$O(nmdo)$	1-16	$O(nmdo)$	1-18	$O(ndo)$	1-17	$O(nmdo)$
2	$O(1)$	2	$O(1)$	2	$O(1)$	2	$O(1)$
3-15	$O(nmdo)$	3-15	$O(nmdo)$	3-17	$O(ndo)$	3-16	$O(nmdo)$
4	$O(1)$	4	$O(1)$	4	$O(1)$	4	$O(1)$
5-11	$O(mdo)$	5-11	$O(mdo)$	5-9	$O(1)$	5-11	$O(mdo)$
6	$O(1)$	6	$O(1)$	10	$O(1)$	6	$O(1)$
7-9	$O(do)$	7-9	$O(do)$	11-13	$O(do)$	7-9	$O(do)$
8	$O(o)$	8	$O(o)$	12	$O(o)$	8	$O(o)$
10	$O(d)$	10	$O(d)$	14	$O(d)$	10	$O(d)$
12	$O(m)$	12	$O(m)$	15-16	$O(1)$	12	$O(m)$
13-14	$O(1)$	13-14	$O(1)$			13	$O(m)$
						14-15	$O(1)$
Overall	$O(nmdo)$	Overall	$O(nmdo)$	Overall	$O(ndo)$	Overall	$O(nmdo)$

5.2.5 Time Complexity Analysis

The step-by-step time complexity of the proposed algorithms, UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA, is shown in Table 5.1. Note that $|UR| = n$, $|DC| = m$, d is the maximum duration of any UR, including UNT, and o is the maximum number of resources under any datacenter.

5.2.6 Illustration

Let us assume that there are nine URs, UR_1 to UR_9, with their ST, D, N, UNL-RE and UN-NRE, as depicted in Table 5.2. As mentioned earlier, UNL-RE has three levels, 1 to 3, representing the maximum UN of RE resources as 30%, 60% and 100%, respectively. We also assume that there are two datacenters, DC_1 and DC_2, with their initial RE resource slots (i.e., $t = 1$ to $t = 11$) as shown in Table 5.3. Note that the green colour indicates the RE resource slots, and the white colour indicates the NRE resource slots. We underline (_) the RE resource slots occupied by the URs.

Table 5.2: ST, D, N, UNL-RE and UN-NRE of nine URs

UR ID	ST	D	N	UNL-RE	UN-NRE
UR_1	1	4	1	2	10%
UR_2	1	3	2	1	10%
UR_3	1	4	1	2	10%
UR_4	3	5	2	3	10%
UR_5	4	3	1	2	10%
UR_6	5	2	1	2	10%
UR_7	5	3	1	3	10%
UR_8	7	2	2	1	10%
UR_9	8	2	3	3	10%

Table 5.3: An initial setup of two datacenters

DC_1		0.1	0.1	0.3	0.2	0.1	0.2	0.2	0.1	0.1	0.3	0.1
	R#	30%	20%	40%	50%	60%	80%	30%	70%	10%	20%	30%
	8											
	7											
	6											
	5											
	4											
	3											
	2											
	1											
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$
DC_2		0.2	0.1	0.2	0.1	0.1	0.3	0.4	0.5	0.2	0.3	0.1
	R#	20%	30%	30%	50%	40%	30%	20%	40%	70%	30%	60%
	8											
	7											
	6											
	5											
	4											
	3											
	2											
	1											
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

Table 5.4: Gantt chart for UNL-FABEF

		0.1	0.1	0.3	0.2	0.1	0.2	0.2	0.1	0.1	0.3	0.1
	R#	30%	20%	40%	50%	60%	80%	30%	70%	10%	20%	30%
	8								UR_9			
	7								UR_9	UR_9		
DC_1	6							UR_8	UR_9	UR_9		
	5					UR_7	UR_7	UR_8	UR_8	UR_9		
	4					UR_6	UR_6	UR_7	UR_8	UR_8		
	3			UR_4	UR_4	UR_4	UR_4	UR_6	UR_7	UR_8	UR_9	
	2			UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_9	
	1	UR_3	UR_3	UR_3	UR_3	UR_3	UR_3	UR_4	UR_4	UR_4	UR_9	
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

		0.2	0.1	0.2	0.1	0.1	0.3	0.4	0.5	0.2	0.3	0.1
	R#	20%	30%	30%	50%	40%	30%	20%	40%	70%	30%	60%
	8											
	7											
DC_2	6				UR_2							
	5				UR_2							
	4											
	3	UR_2	UR_2	UR_2								
	2	UR_2	UR_2	UR_2	UR_5	UR_5	UR_5					
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_5				
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

Table 5.5: Gantt chart for UNL-HAREF

		0.1	0.1	0.3	0.2	0.1	0.2	0.2	0.1	0.1	0.3	0.1
	R#	30%	20%	40%	50%	60%	80%	30%	70%	10%	20%	30%
	8											
	7											
	6											
DC_1	5											
	4			UR_2	UR_4			UR_8	UR_8	UR_8		
	3			UR_2	UR_2			UR_8	UR_8	UR_8		
	2	UR_2	UR_2	UR_4	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	
	1	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

		0.2	0.1	0.2	0.1	0.1	0.3	0.4	0.5	0.2	0.3	0.1
	R#	20%	30%	30%	50%	40%	30%	20%	40%	70%	30%	60%
	8											
	7											
	6											
DC_2	5					UR_7	UR_7					
	4					UR_6	UR_6		UR_9			
	3				UR_5	UR_5	UR_5	UR_7	UR_9	UR_9	UR_9	UR_9
	2	UR_3	UR_3	UR_3	UR_3	UR_3	UR_3	UR_6	UR_9	UR_9	UR_9	UR_9
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_5	UR_7	UR_9	UR_9	UR_9
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

Table 5.6: Gantt chart for UNL-RR

		0.1	0.1	0.3	0.2	0.1	0.2	0.2	0.1	0.1	0.3	0.1
	R#	30%	20%	40%	50%	60%	80%	30%	70%	10%	20%	30%
	8											
	7											
	6											
DC_1	5											
	4					UR_7	UR_5		UR_9			
	3				UR_5	UR_5	UR_3		UR_9	UR_9	UR_9	
	2	UR_3	UR_3	UR_3	UR_3	UR_3	UR_1	UR_7	UR_9	UR_9	UR_9	
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_7	UR_5	UR_7	UR_9	UR_9	
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$
		0.2	0.1	0.2	0.1	0.1	0.3	0.4	0.5	0.2	0.3	0.1
	R#	20%	30%	30%	50%	40%	30%	20%	40%	70%	30%	60%
	8											
	7											
	6											
DC_2	5							UR_8		UR_8		
	4			UR_4	UR_4			UR_8	UR_8	UR_8		
	3			UR_4	UR_4	UR_6	UR_6	UR_6	UR_8			
	2	UR_2	UR_2	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4		
	1	UR_2	UR_2	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4		
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

Table 5.7: Gantt chart for UNL-MOSA while matching UR_1

		0.1	0.1	0.3	0.2	0.1	0.2	0.2	0.1	0.1	0.3	0.1
	R#	30%	20%	40%	50%	60%	80%	30%	70%	10%	20%	30%
	8											
	7											
	6											
DC_1	5											
	4											
	3											
	2						UR_1					
	1	UR_1	UR_1	UR_1	UR_1	UR_1						
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$
		0.2	0.1	0.2	0.1	0.1	0.3	0.4	0.5	0.2	0.3	0.1
	R#	20%	30%	30%	50%	40%	30%	20%	40%	70%	30%	60%
	8											
	7											
	6											
DC_2	5											
	4											
	3											
	2											
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1					
		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$

Table 5.8: Gantt chart for UNL-MOSA

		0.1	0.1	0.3	0.2	0.1	0.2	0.2	0.1	0.1	0.3	0.1
	R#	30%	20%	40%	50%	60%	80%	30%	70%	10%	20%	30%
	8											
	7											
	6								UR_9			
DC_1	5							UR_8	UR_9	UR_9		
	4					UR_7	UR_6	UR_8	UR_9	UR_9		
	3					UR_6	UR_5	UR_7	UR_8	UR_9	UR_9	
	2				UR_5	UR_5	UR_3	UR_6	UR_8	UR_8	UR_9	
	1	UR_3	UR_3	UR_3	UR_3	UR_3	UR_7	UR_5	UR_7	UR_8	UR_9	
		$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$	$t = 9$	$t = 10$	$t = 11$
		0.2	0.1	0.2	0.1	0.1	0.3	0.4	0.5	0.2	0.3	0.1
	R#	20%	30%	30%	50%	40%	30%	20%	40%	70%	30%	60%
	8											
	7											
	6				UR_2							
DC_2	5			UR_4	UR_2							
	4			UR_4								
	3	UR_2	UR_2	UR_2	UR_4	UR_4	UR_4					
	2	UR_2	UR_2	UR_2	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	UR_4	
	1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_1	UR_4	UR_4	UR_4	UR_4	
		$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$	$t = 9$	$t = 10$	$t = 11$

5.2.6.1 Illustration of UNL-FABEF

In the UNL-FABEF algorithm, the UR_1 demands service from t = 1 to t = 4 (i.e., from ST to ST + D - 1) with one node. The UR_1 can be executed in UNL 2 of RE, and its NRE UN percentage is 10%. It is mapped with each datacenter's RE UNL for the duration mentioned. Suppose the RE UNL of a UR is greater than or equal to the RE UNL of datacenter in a particular time slot. In that case, the RE resources in that time slot can be assigned to that UR. Otherwise, NRE resources can be assigned in that time slot. As RE UNL of DC_1 is 30%, 20%, 40% and 50% during t = 1 to t = 4, UR_1 can be accommodated in the DC_1. On the other hand, RE UNL of DC_2 is 20%, 30%, 30% and 50% during t = 1 to t = 4, UR_1 can also be accommodated in the DC_2. The UN of DC_1 and DC_2 is calculated as $4 \times 0.45 = \lceil 1.80 \rceil \approx 2$. Note that we took the average of higher (i.e., 60) and lower (i.e., 30) bounds to calculate the UN for the simplicity of calculation. Therefore, the completion time of UR_1 can be extended from 4 to 6 in both datacenters. However, the RE UNL of DC_1 is 60% and 80% in t = 5 and t = 6, respectively. On the contrary, the RE UNL of DC_2 is 40% and 30% in t = 5 and t = 6, respectively. Here, DC_2 can only fulfil the RE

UNL of UR_1, whereas DC_1 can fulfil using NRE resource. Alternatively, the ACO of UR_1 in DC_1 and DC_2 is 0.7 (i.e., 5 $\times$ 0.10 + 1 $\times$ 0.20) and 0.6 (i.e., 6 $\times$ 0.10), respectively. Therefore, UR_1 is assigned to DC_2.

Next, UR_2 demands service from t = 1 to t = 3 with two nodes. It can be executed in UNL 1 of RE, and its NRE UN percentage is 10%. As RE UNL of DC_1 is 30%, 20% and 40% during t = 1 to t = 3, UR_2 can be accommodated in the DC_1. On the other hand, RE UNL of DC_2 is 20%, 30% and 30% during t = 1 to t = 3, UR_2 can also be accommodated in the DC_2. The UN of DC_1 is calculated as 2 $\times$ 0.15 + (1 $\times$ 0.15 + 1 $\times$ 0.10) + 2 $\times$ 0.10 = $\lceil 0.75 \rceil \approx 1$. The UN of DC_2 is calculated as 2 $\times$ 0.15 + 2 $\times$ 0.10 + (1 $\times$ 0.15 + 1 $\times$ 0.10) = $\lceil 0.75 \rceil \approx 1$. Note that we took the average of higher (i.e., 30) and lower (i.e., 0) bounds to calculate the UN for the simplicity of calculation. Therefore, the completion time of UR_2 can be extended from 3 to 3 + $\lceil \frac{1}{2} \rceil$ = 4 in both datacenters. However, the RE UNL of DC_1 and DC_2 is 50% and 40% at t = 5. Here, both DC_1 and DC_2 can fulfil the demand of UR_2 by providing three RE resource slots and five NRE resource slots. However, the ACO of UR_2 in DC_1 and DC_2 is 1.40 (i.e., 2 $\times$ 0.10 + (1 $\times$ 0.10 + 1 $\times$ 0.10) + 2 $\times$ 0.30 + 2 $\times$ 0.20) and 0.90 (i.e., 2 $\times$ 0.10 + 2 $\times$ 0.10 + (1 $\times$ 0.10 + 1 $\times$ 0.20) + 2 $\times$ 0.10), respectively. Therefore, UR_2 is assigned to DC_2.

Next, UR_3 demands service from t = 1 to t = 4 with one node. It can be executed in UNL 2 of RE, and its NRE UN percentage is 10%. As RE UNL of DC_1 is 30%, 20%, 40% and 50% during t = 1 to t = 4, UR_3 can be accommodated in the DC_1. On the other hand, RE UNL of DC_2 is 20%, 30%, 30% and 50% during t = 1 to t = 4, UR_3 can also be accommodated in the DC_2. The UN of DC_1 is calculated as 4 $\times$ 0.45 = $\lceil 1.80 \rceil \approx 2$. The UN of DC_2 is calculated as 3 $\times$ 0.10 + 1 $\times$ 0.45 = $\lceil 0.75 \rceil \approx 1$. Therefore, the completion time of UR_1 can be extended from 4 to 6 in DC_1 and from 4 to 5 in DC_2. However, the RE UNL of DC_1 is 60% and 80% in t = 5 and t = 6, and the RE UNL of DC_2 is 40% at t = 5. Here, DC_1 can fulfil the demand of UR_3 by providing five RE resource slots and one NRE resource slot and DC_2 by providing two RE resource slots and three NRE resource slot. However, the ACO of UR_1 in DC_1 and DC_2 is 0.70 (i.e., 5 $\times$ 0.10 + 1 $\times$ 0.20) and 0.70 (i.e., 2 $\times$ 0.10 + 1 $\times$ 0.20 + 1 $\times$ 0.10 + 1 $\times$ 0.20), respectively. Therefore, UR_3 is assigned to DC_1. In the same way, UR_4 to UR_9 are assigned to DC_1, DC_2, DC_1, DC_1, DC_1 and

DC_1, respectively. The ACOs are 2.00, 0.60, 0.50, 0.60, 0.80 and 1.10, respectively. The OCO for DC_1 and DC_2 is 5.70 and 2.10, respectively. The TNRE is 26. The UNT and UNCO are 12 and 2.20, respectively. The UNL-FABEF's final Gantt chart is presented in Table 5.4.

5.2.6.2 Illustration of UNL-HAREF

Like the UNL-FABEF algorithm, the UNL-HAREF algorithm finds the UN and completion time of DC_1 and DC_2 for UR_1. The TNRE for UR_1 in DC_1 and DC_2 is 5 and 6, respectively. Therefore, UR_1 is assigned to DC_2. The ACO of UR_1 in DC_2 is 0.60 (i.e., 6 $\times$ 0.10). Next, the UNL-HAREF algorithm finds the UN and completion time of DC_1 and DC_2 for UR_2 like the UNL-FABEF algorithm. The TNRE for UR_2 in DC_1 and DC_2 is 3. Therefore, UR_2 is assigned to DC_1. The ACO of UR_2 in DC_1 is 1.4 (i.e., 2 $\times$ 0.10 + (1 $\times$ 0.10 + 1 $\times$ 0.10) + 2 $\times$ 0.30 + 2 $\times$ 0.20). Similarly, the TNRE for UR_3 in DC_1 and DC_2 is 3 and 4, respectively. Therefore, UR_3 is assigned to DC_2. The ACO of UR_3 in DC_2 is 0.8 (i.e., 1 $\times$ 0.10 + 1 $\times$ 0.10 + 1 $\times$ 0.10 + 1 $\times$ 0.10 + 1 $\times$ 0.10 + 1 $\times$ 0.30). In the same way, UR_4 to UR_9 are assigned to DC_1, DC_2, DC_2, DC_2, DC_1 and DC_2, respectively. The TNRE in these datacenters is 12, 3, 1, 1, 0 and 7, respectively. The ACOs in these datacenters are 1.90, 0.60, 0.50, 0.90, 0.80, and 2.20, respectively. The OCO for DC_1 and DC_2 is 5.50 and 4.20, respectively. The TNRE is 37. The UNT and UNCO are 14 and 2.90, respectively. The UNL-HAREF's final Gantt chart is presented in Table 5.5.

5.2.6.3 Illustration of UNL-RR

In the UNL-RR algorithm, UR_1 is assigned to DC_1. Here, DC_1 can provide four RE resource slots without UN. As a result, the UN of DC_1 is calculated as 2. The completion time of UR_1 in DC_1 is calculated as 6 with UN. The ACO of UR_1 in DC_1 is 0.7, and the TNRE is 5. Similarly, UR_2 is assigned to DC_2. Here, DC_2 can provide five RE resource slots and one NRE resource slot without UN. As a result, the UN of DC_2 is calculated as 1. The completion time of UR_2 in DC_2 is calculated as 4 with UN. The ACO of UR_2 in DC_2 is 0.8, and the TNRE is 5. Next, UR_3 is assigned to DC_1. Here, DC_1 can deliver two RE resource slots and two NRE resource slots without UN. As a result, the UN of DC_1 is

calculated as 2. The completion time of UR_3 in DC_1 is calculated as 6 with UN. The ACO of UR_3 in DC_1 is 0.8, and the TNRE is 3. In the same way, UR_4 to UR_9 are assigned to DC_2, DC_1, DC_2, DC_1, DC_2 and DC_1, respectively. The ACOs in these datacenters are 1.80, 0.60, 0.80, 0.50, 2.20 and 1.10, respectively. The TNRE in these datacenters is 11, 1, 1, 2, 0 and 4, respectively. The OCO for DC_1 and DC_2 is 3.70 and 5.60, respectively. The TNRE is 34. The UNT and UNCO are 13 and 2.70, respectively. The UNL-RR's final Gantt chart is presented in Table 5.6.

5.2.6.4 Illustration of UNL-MOSA

In the UNL-MOSA algorithm, the UR_1 demands service from t = 1 to t = 4 with one node and it can be executed in UNL 2 of RE. The NRE UN percentage is 10%. RE UNL of DC_1 is 30%, 20%, 40% and 50% and RE UNL of DC_2 is 20%, 30%, 30% and 50% during t = 1 to t = 4. As a result, UR_1 can be accommodated in the DC_1 and DC_2 with UN as 2. Therefore, the completion time of UR_1 is 6 in both datacenters. The ACO of UR_1 in DC_1 and DC_2 is 0.7 and 0.6, respectively, and their normalized value is $\frac{0.7}{0.7}$ and $\frac{0.6}{0.7}$, respectively. The TNRE of UR_1 in DC_1 and DC_2 is 5 and 6, respectively, and their normalized value is $\frac{1/5}{1/5}$ and $\frac{1/6}{1/5}$, respectively. The linear combination value of normalized ACO and TNRE in DC_1 and DC_2 is 1 (i.e., 0.5 $\times$ 1 + 0.5 $\times$ 1) and 0.845 (i.e., 0.5 $\times$ 0.857 + 0.5 $\times$ 0.833), respectively. Therefore, UR_1 is assigned to DC_2 as shown in Table 5.7.

Next, the UR_2 demands service from t = 1 to t = 3 with two nodes and it can be executed in UNL 1 of RE. The NRE UN percentage is 10%. RE UNL of DC_1 is 30%, 20% and 40% and RE UNL of DC_2 is 20%, 30% and 30% during t = 1 to t = 3. The UN of DC_1 and DC_2 is calculated as 1 for two nodes. Therefore, the completion time of UR_2 is 4 in both datacenters. The ACO of UR_2 in DC_1 and DC_2 is 1.4 and 0.9, respectively, and their normalized value is 1 and 0.64, respectively. The TNRE of UR_2 in DC_1 and DC_2 is 3, and the normalized value is 1. The linear combination value of normalized ACO and TNRE in DC_1 and DC_2 is 1 and 0.82, respectively. Therefore, UR_2 is assigned to DC_2.

Next, the UR_3 demands service from t = 1 to t = 4 with one node and it can be executed in UNL 2 of RE. UR_3 can be accommodated in the DC_1 and DC_2 with UN as 2 and 1, respectively. Therefore, the completion time of UR_3 is 6 and 5 in DC_1 and DC_2, respec-

Table 5.9: Comparison of five performance metrics for the UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA algorithms

Performance metrics	UNL-FABEF	UNL-HAREF	UNL-RR	UNL-MOSA
OCO	7.80	9.70	9.30	7.90
TNRE	26	37	34	34
TNNRE	34	28	26	28
UNT	12	14	13	13
UNCO	2.20	2.90	2.70	2.40

tively. The ACO of UR_3 in DC_1 and DC_2 is 0.7, and the normalized value is 1. The number of available RE resource slots of UR_3 in DC_1 and DC_2 is 5 and 2, respectively, and their normalized value is 0.4 and 1, respectively. The linear combination value of normalized ACO and TNRE in DC_1 and DC_2 is 0.7 and 1, respectively. Therefore, UR_3 is assigned to DC_1. In the same way, UR_4 to UR_9 are assigned to DC_2, DC_1, DC_1, DC_1, DC_1 and DC_1, respectively. The ACOs in these datacenters are 2.2, 0.6, 0.5, 0.5, 0.8 and 1.1, respectively. The TNRE in these datacenters is 12, 2, 0, 2, 2 and 2, respectively. The OCO for DC_1 and DC_2 is 3.70 and 4.20, respectively. The TNRE is 34. The UNT and UNCO are 13 and 2.40, respectively. The UNL-MOSA's final Gantt chart is presented in Table 5.8. The comparison of the UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA algorithms is shown in Table 5.9. The comparison shows the superiority of the UNL-FABEF in terms of the OCO, the UNL-HAREF in terms of the TNRE and the UNL-MOSA in balancing the OCO and the TNRE.

5.3 Performance Metrics, Datasets and Simulation Results

This section presents the definition of the five performance metrics, the dataset generation process and the four algorithms' simulation results.

5.3.1 Performance Metrics

We use five performance metrics, namely the OCO, TNRE, TNNRE, UNT and UNCO. OCO comprises two costs of all the datacenters, i.e., RE and NRE resource costs. These costs are accumulated using ACO. Therefore, OCO is the sum of ACO of all the URs. It

can be mathematically expressed as follows.

$$OCO = \sum_{k=1}^{|UR|} \sum_{i=1}^{|DC|} ACO[i] \times X[k,i] \tag{5.2}$$

where

$$X[k,i] = \begin{cases} 1 & \text{If } DC_i \text{ accommodates } UR_k \text{ in its resource slots} \\ 0 & \text{Otherwise} \end{cases} \tag{5.3}$$

$TNRE$ is the sum of the number of RE resource slots used by all the URs. Mathematically,

$$TNRE = \sum_{k=1}^{n} slots\_re[k] \tag{5.4}$$

Alternatively, $TNNRE$ is the sum of the number of NRE resource slots used by all the URs. Mathematically,

$$TNNRE = \sum_{k=1}^{n} slots\_nre[k] \tag{5.5}$$

UNT is the sum of the additional time durations incurred towards satisfying the UN of URs. It can be defined as follows.

$$UNT = \sum_{k=1}^{n} Y[k] \tag{5.6}$$

where

$$Y[k] = \begin{cases} UN[k] & \text{If } N[k] = 1 \\ \left\lfloor \frac{UN[k]}{N[k]} \right\rfloor & \text{If } N[k] \geq 2 \end{cases} \tag{5.7}$$

$UNCO$ is the sum of the additional costs incurred for the usage of RE and NRE resource slots during UN time durations. It can be defined as follows.

$$UNCO = \sum_{k=1}^{n} \sum_{i=1}^{m} \sum_{l=ST[k]+D[k]}^{ST[k]+D[k]+Y[k]} \sum_{j=1}^{o} CO[l,j] \times X[k,i] \tag{5.8}$$

Note that $CO[l,j]$ is shown as $CO[l]$ in the algorithms because the cost of resource slots at a particular time instance is the same.

5.3.2 Datasets

The datasets are generated using a system with Intel(R) Core(TM) i5-10310U CPU @ 1.70 GHz 2.21 GHz, 8 GB RAM, 64-bit operating system and x64-based processor. The edition, version and operating system build of Windows is 11 Home single language, 22H2 and 22621.1702, respectively. The system is installed with MATLAB R2017a with version 9.2.0.538062. However, the datasets can be generated using any system configuration with any latest version after MATLAB R2014a. It is important to note that the works [46, 54, 58, 67, 88] used real workload traces, mostly Google cluster-usage traces. However, Toosi and Buyya [46] stated that there are no publicly available workload traces, and Google traces lack information regarding workload-containing requests. As a result, they have generated some of the parameters randomly. In line with [46], we generate the parameters, including Google trace parameters, using uniformly distributed random numbers in MATLAB, as no existing dataset/trace deals with the problem introduced in this chapter. Alternatively, we create a codistributed matrix of uniformly distributed random numbers using the $randi$ function. The details about the generation of datasets and their properties and attributes are discussed as follows. (1) We determine the number of URs and datacenters, say 200 and 20, respectively. (2) The UR IDs are generated chronologically from UR_1 to UR_{200} and kept in one of the columns of the Microsoft Excel file. (3) The ST of URs are generated between 1 and 100 using the randi([1, 100], [200, 1]) function and kept in another column of the Microsoft Excel file. Note that this function generates a 200 $\times$ 1 matrix in which the values are in the range of 1 to 100. (4) The D of URs are generated between 10 and 25 using the randi([10, 25], [200, 1]) function and kept in another column. (5) The N of URs are generated between 10 and 100 using the randi([10, 100], [200, 1]) function and kept in another column. (6) The UNL-RE of URs are generated between 1 and 3 using the randi([1, 3], [200, 1]) function and kept in another column. (7) The UN-NRE of URs are fixed as 10% and kept in the last column. The Excel file is given as input to all four scheduling algorithms. It is noteworthy to mention that the attribute of UR ID, ST, D, N, UNL-RE and UN-NRE is discrete, discrete, discrete, discrete, nominal and fixed, respectively. On the other hand, the datacenters are configured by taking a 3D matrix in which the dimensions

Table 5.10: Parameters and their range of values

Parameter	Values
Number of datacenters	[20 $\sim$ 200]
Number of resources per datacenter	[10 $\sim$ 400]
Number of URs	[200 $\sim$ 2000]
ST of URs	[1 $\sim$ 100]
D of URs	[10 $\sim$ 25]
N of URs	[10 $\sim$ 100]
UNL of RE	1, 2 and 3
UN of NRE	10%
NRE cost of datacenter	[1 $\sim$ 100]
RE cost of datacenter	1
UN percentage of RE resources	[1% $\sim$ 100%]

represent time instance, number of resources and datacenter ID as follows. (1) The range of these dimensions is [1, 125], [10, 400] and [20, 200], respectively. (2) The NRE cost of each datacenter is generated using the randi([1 100], [125, 1]) function. In contrast, the RE cost of each datacenter is fixed as 1. (3) The UN percentage of RE resources is generated using the randi([1 100], [125, 1]) function. It is noteworthy to mention that the attribute of time instance, number of resources, datacenter ID, NRE cost, RE cost and UN percentage is discrete, discrete, discrete, discrete, fixed and discrete, respectively. The parameters that are used in the dataset generation are listed in Table 5.10. Using these parameters, we generate ten datasets and present them using the number of URs and datacenters. For instance, 200 $\times$ 20 shows that 200 URs are assigned to 20 datacenters and 2000 $\times$ 200 shows that 2000 URs are assigned to 200 datacenters. Each dataset contains five instances of the same size to carry out the simulation results of the four algorithms. Then, the results are averaged to present in the pictorial form.

5.3.3 Simulation Results

The results carried out during the simulation runs of UNL-FABEF, UNL-HAREF, UNL-RR, and UNL-MOSA algorithms are compared using ten datasets in terms of five performance metrics. The extended baseline algorithms and UNL-MOSA are compared using the same environmental setup and parameters. The discussion on the results is stated as

follows.

- UNL-FABEF performs better in OCO compared to the other three algorithms, as shown in Fig. 5.1. More specifically, UNL-FABEF reduces the OCO by 53%, 54% and 8% on average compared to UNL-HAREF, UNL-RR and UNL-MOSA. On the other hand, the percentage of reduction in OCO by UNL-FABEF is 50% to 55%, 48% to 56% and 6% to 9% compared to UNL-HAREF, UNL-RR and UNL-MOSA. The better performance is because UNL-FABEF selects the datacenter with the least ACO, including the UNT period. UNL-MOSA performs better after UNL-FABEF as it aims to minimize the OCO. On the other hand, UNL-HAREF is the worst performing as it aims to maximize the TNRE. As a result, it can be concluded that maximizing RE usage does not lead to minimizing the OCO. Note that the same observation is also reported in [46, 70, 71].

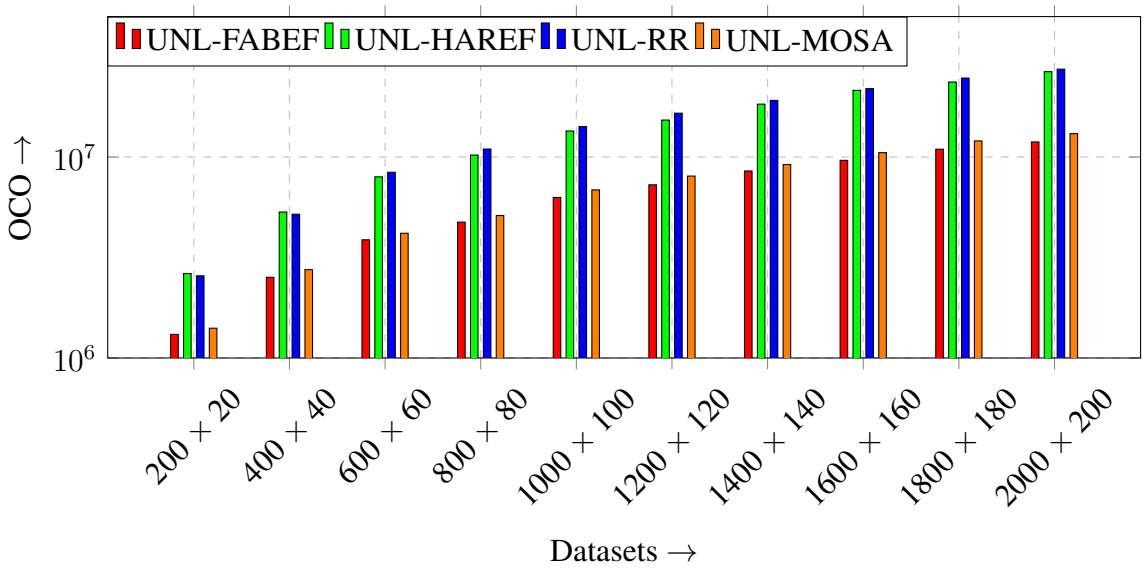

Figure 5.1: Pictorial comparison of OCO for UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA using ten datasets.

- As seen in Fig. 5.2, UNL-HAREF outperforms in TNRE compared to the other algorithms. UNL-HAREF achieves an average increase of 22%, 33% and 19% in TNRE compared to UNL-FAREF, UNL-RR, and UNL-MOSA. Conversely, UNL-HAREF results in an increase in TNRE ranging from 19% to 24%, 31% to 36%, and 12% to 21% when compared to UNL-HAREF, UNL-RR, and UNL-MOSA. It

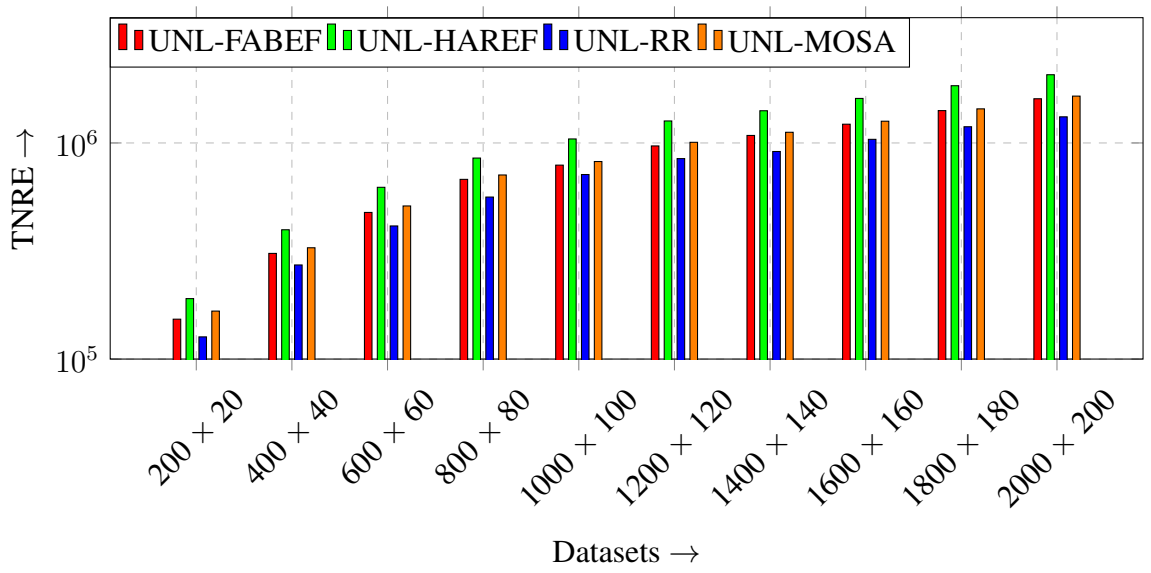

Figure 5.2: Pictorial comparison of TNRE for UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA using ten datasets.

is evident as UNL-HAREF selects the datacenter with the maximum TNRE. UNL-MOSA is next to UNL-HAREF as it aims to maximize TNRE. On the contrary, UNL-RR performs worst as it considers no factor while assigning the URs.

- We know that RE generation is uncertain and unpredictable. It is clearly observed from Fig. 5.3 that UNL-HAREF is the worst-performing algorithm in terms of UNT compared to other algorithms. On the other hand, UNL-RR outperforms all, and the performance of UNL-FABEF is close to UNL-RR. UNL-RR demonstrates an average reduction of 12%, 19%, and 13% in UNT when compared to UNL-FABEF, UNL-HAREF, and UNL-MOSA, respectively. Conversely, the reduction percentage in UNT achieved by UNL-RR ranges from 9% to 14%, 17% to 20%, and 10% to 15% when compared to UNL-FABEF, UNL-HAREF, and UNL-MOSA.

- All four algorithms are finally compared in terms of UNCO using Fig. 5.4. Here, UNL-FABEF performs better than other algorithms as it selects the least cost resource slots in the UNT period. More specifically, UNL-FABEF reduces the UNCO by 56%, 67% and 8% on average compared to UNL-HAREF, UNL-RR and UNL-MOSA. On the other hand, the percentage of reduction in UNCO by UNL-FABEF is 54% to 58%, 58% to 70% and 3% to 17% compared to UNL-HAREF, UNL-RR

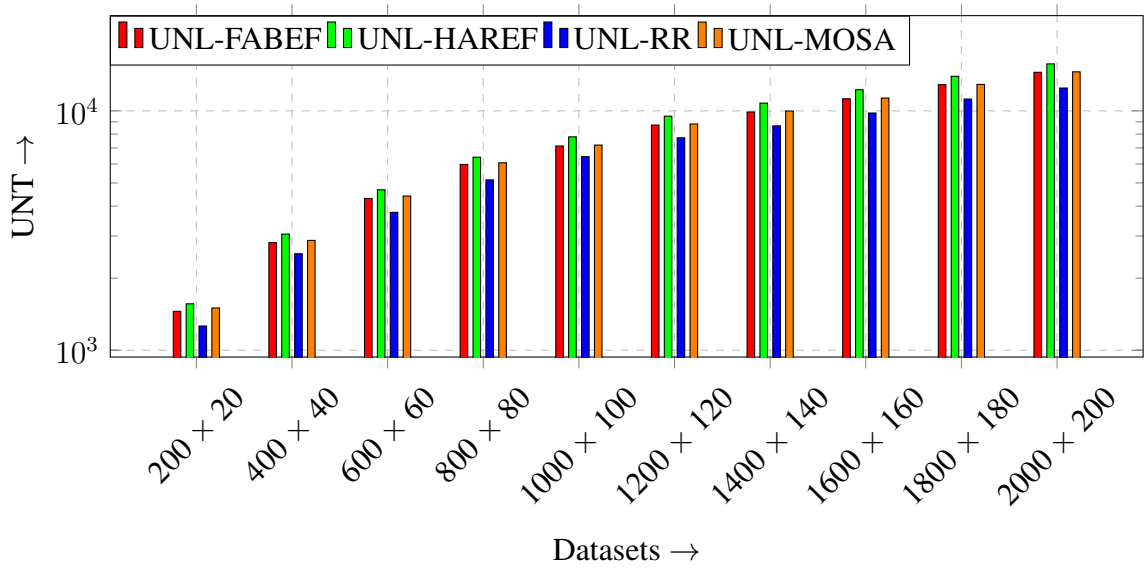

Figure 5.3: Pictorial comparison of UNT for UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA using ten datasets.

and UNL-MOSA. UNL-MOSA performs better next to UNL-FABEF because of its objective. On the other hand, UN-RR performs poorly and produces more UNCO. As a result, it can be concluded that minimizing the UNT does not lead to reducing the UNCO.

5.3.4 Analysis of Variance Statistical Test

Analysis of variance (called ANOVA) [56] is a well-known statistical test that compares more than two groups/algorithms and determines their relationship. It takes two hypotheses, namely null and alternate. In the null hypothesis, we consider that the means of four algorithms, UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA, are equal. On the contrary, one of the means differs in the alternate hypothesis.

ANOVA considers two sources of variations (SV), namely between and within groups, to determine the sum of square (SS), degree of freedom (DF), mean square (MS), F value, P value and F critical (FC) value. As mentioned earlier, there are ten datasets and four algorithms. As a result, the number of simulation results and tests performed are 40 and 4, respectively. The DF is calculated between and within groups as follows. In the between group, DF is calculated by subtracting one from the number of tests (i.e., 4 - 1 = 3). In

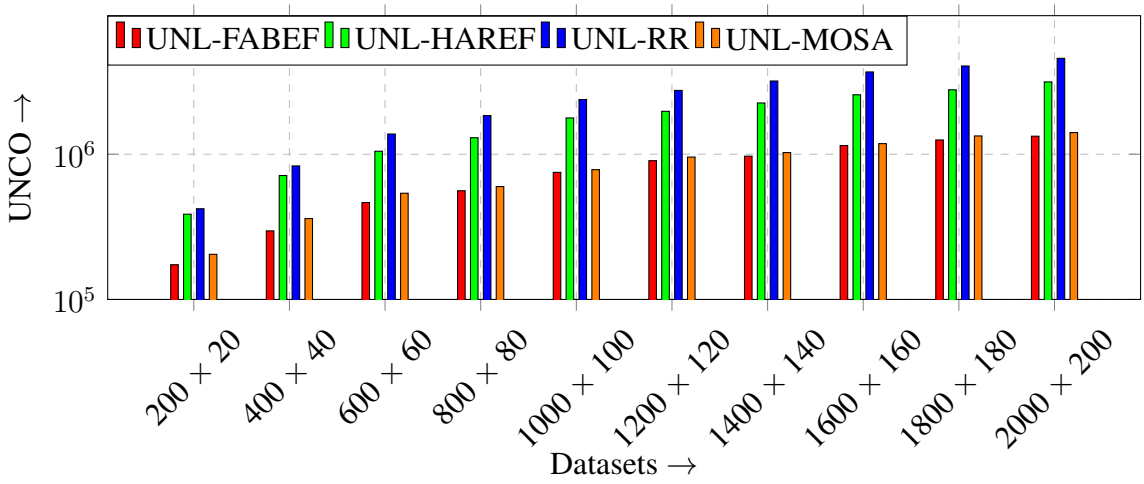

Figure 5.4: Pictorial comparison of UNCO for UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA using ten datasets.

the within group, DF is calculated by subtracting the number of tests from the number of simulation results (i.e., 40 - 4 = 36).

We perform the ANOVA statistical tests on OCO and UNCO as shown in Table 5.11 and Table 5.12, respectively, since these cost metrics are key performance indicators. In both metrics, we found that the F value is greater than the FC value, and the P value is less than 0.05. As a result, the null hypothesis is rejected, and the alternate hypothesis is accepted. It means that the means are unequal, and there is a significant difference between the four scheduling algorithms.

Table 5.11: Comparison of OCO for UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA using ANOVA statistical test

SV		SS	DF	MS	F	P	FC
Groups	Between	6.1015E+14	3	2.0338E+14	5.0074	0.0053	2.8663
	Within	1.4622E+15	36	4.0616E+13			
Total		2.0723E+15	39				

Table 5.12: Comparison of UNCO for UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA using ANOVA statistical test

SV		SS	df	MS	F	P	FC
Groups	Between	2.0564E+13	3	6.8545E+12	8.7963	0.0002	2.8663
	Within	2.8053E+13	36	7.7925E+11			
Total		4.8616E+13	39				

5.4 Summary

This chapter has presented four algorithms, UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA. The first three algorithms extend three benchmark algorithms, FABEF, HAREF and RR. These UNL-based algorithms focus to minimize the OCO, TNNRE, UNT and UNCO, and maximize the TNRE. The main principle behind these algorithms is to incorporate the UN from the user and CSP perspectives by considering three levels, low, medium and high. The algorithms have been shown to require $O(nmdo)$ for n URs, m datacenters, d duration and o resources except for UNL-RR. UNL-RR has been shown to require $O(ndo)$. All these algorithms have been simulated using ten datasets and compared using five performance metrics. The comparison results show the applicability of each algorithm concerning the performance metrics. More specifically, UNL-FABEF, UNL-HAREF and UNL-RR perform better in OCO and UNCO, TNRE and UNT, respectively. On the contrary, UNL-MOSA makes a trade-off between UNL-FABEF and UNL-HAREF and achieves a balance between OCO and TNRE. However, the proposed algorithms consider UN and UNL within the datacenter. An improvement introduced is the ability to relocate the UR from one datacenter to another in response to any UN. Additionally, this relocation can be optimized based on the availability of RE sources to maximize OCO savings. Consequently, the next chapter investigates the migration algorithm for geographically distributed datacenters.

Chapter 6

A Renewable Energy-Oriented Migration Algorithm for Minimizing Cost in Geo-Distributed Cloud Datacenters

This chapter addresses assigning a set of n URs to a set of m datacenters, where each datacenter is supplied with RE and NRE sources. This problem, known as UR migration in RE-based scheduling, aims to minimize OCO. The primary motivation originates from the variability of RE sources, which depend on geographical location, time, cost, and climate conditions. Consequently, the cost of executing URs varies across different datacenters. From the user's perspective, the goal is to minimize billing costs while meeting the desired requirements. On the other hand, from the CSP's perspective, the goal is to reduce the cost of executing URs, thereby maximizing profit. Profit is the margin between the cost agreed upon with the user based on the SLA and the actual incurred cost. By selecting appropriate datacenters for each UR, the CSP can increase savings. The three benchmark algorithms, FABEF, HAREF, and RR, select datacenters based on lowest cost, highest RE resources, and circular order, respectively. These algorithms must effectively utilize the available RE resources among the datacenters. In contrast, CSPs can save on OCO by redi-

recting URs from one datacenter to another based on the availability of RE resources. This insight has led to the developing of a novel algorithm that integrates RE resources into the cost-minimizing FABEF algorithm by migrating URs between datacenters. Therefore, we present a novel REOMA to minimize the OCO for geographically distributed datacenters using strategic migration. REOMA calculates the cost incurred during the time window for each UR in the datacenters and identifies the least ACO datacenter. It then models each datacenter's cost over the time window as a polynomial curve and determines the slope and intercept. Next, it identifies migration points between the least ACO datacenter and other datacenters, which may be zero, one, or multiple points. Finally, REOMA selects the pair of datacenters to perform the migration with the lowest ACO. The effectiveness of REOMA is evaluated against three benchmark algorithms using OCO, TNRE, and TNNRE metrics. The evaluation uses datasets ranging from 200 to 2000 URs in increments of 200 and from 20 to 200 datacenters in increments of 20. The performance metrics demonstrate significant improvements in REOMA over all other benchmark algorithms. In summary, this chapter offers the following contributions to the research community.

1. We develop a novel migration algorithm that integrates RE and NRE sources. This algorithm facilitates the efficient relocation of URs from one datacenter to another within a geo-distributed cloud datacenter environment, aiming to minimize OCO.

2. The proposed migration algorithm fits the costs of datacenters over time intervals for a UR into a polynomial curve and calculates both the slope and intercept.

3. The proposed migration algorithm identifies the intersection point between the least ACO datacenter and other datacenters as the migration point.

4. The proposed migration algorithm is evaluated against three benchmark algorithms, FABEF, HAREF and RR, to demonstrate its enhancements in OCO reduction and optimization of TNRE and TNNRE across ten datasets.

The next sections are outlined as follows. Section 6.1 presents the system model and the problem statement. Section 6.2 introduces the proposed algorithms with time complexity

analysis. Section 6.3 defines three performance metrics, and discusses the datasets and simulation results. Section 6.4 concludes the presented work.

6.1 System Model and Problem Formulation

6.1.1 System Model

We consider an IaaS CSP that deploys datacenters in various locations worldwide. These datacenters are interconnected with high bandwidth. For example, the Google cloud platform (GCP) covers over 40 regions, 121 availability zones, and 173 points of presence. Similarly, Microsoft Azure spans over 60 regions and 140 points of presence [124]. Each availability zone includes multiple datacenters, each housing numerous servers, necessitating substantial energy consumption. Traditionally, these datacenters rely on NRE, which significantly impacts the environment. Consequently, CSPs seek RE-based solutions for sustainable development. However, RE generation depends on climate conditions, location, and time. Thus, CSPs analyze RE generation in the locations of their datacenters to migrate URs between datacenters for sustainable solutions. These solutions not only protect the environment but also reduce OCO. Determining an optimal migration point between datacenters is challenging and not well-studied. Our proposed model aims to migrate URs between datacenters by identifying an appropriate migration point to minimize OCO.

6.1.2 Problem Formulation

Given an ordered set of n URs, UR_k, $1 \leq k \leq n$ and a set of m datacenters, DC_i, $1 \leq i \leq m$. A UR, UR_k, $1 \leq k \leq n$, is defined as a 3-tuple, i.e., $<ST_k, D_k, N_k>$ in which ST_k, D_k and N_k indicate the start time, duration and number of required nodes/resources of UR UR_k. A datacenter, DC_i, $1 \leq i \leq m$, is defined as a 4-tuple, i.e., $<R_i, ARE_i, ANRE_i, CO_i>$ in which R_i, ARE_i, $ANRE_i$ and $COST_i$ indicate the number of available resources, an array of available RE resources, an array of available NRE resources and an array of costs over a period of time of datacenter DC_i. The problem is to match each UR with m datacenters and schedule the UR to one of the datacenters, such that the objectives

described below are achieved.

1. Minimize the OCO

2. Maximize the TNRE and minimize the TNNRE

The constraints below are enforced to solve the above problem.

1. The execution order of URs is unchanged.

2. A UR can relocate from one datacenter to another datacenter.

3. The required resources for each UR can be met by initially utilizing RE resources, followed by NRE resources if needed.

6.2 Renewable Energy-Oriented Migration Algorithm

We introduce REOMA into geo-distributed cloud datacenters to facilitate seamless migration of UR from one datacenter to another. The objective is to minimize OCO and TNNRE and maximize the TNRE. REOMA performs the migration using a three-step process: cost estimation, planning, and execution. In the cost estimation process, REOMA determines the ACO of a UR with respect to the time window in each datacenter. Subsequently, it identifies a datacenter with the least cost. In the planning process, REOMA takes the cost of a UR relative to its time window, generates a first-degree polynomial curve and provides the polynomial coefficients, namely slope and intercept. Furthermore, REOMA finds the migration points between the least cost datacenter and other available datacenters. Finally, it selects a pair of datacenter, resulting in the lowest cost. In the execution process, REOMA assigns the UR to the least cost datacenter till the migration point and migrates to another datacenter after the migration point. In this way, REOMA effectively reduces the OCO associated with executing URs across the datacenters. The pseudocode of REOMA is outlined in Algorithm 6.1. The pseudocode is described with respect to the three-step process in the following subsections.

Algorithm 6.1 REOMA

Input: n, m, 1-D matrix: ST, D, N and 2-D matrix: DCE, DCP
Output: OCO, $TNRE$, $TNNRE$

1: **for** $i \leftarrow 1, 2, 3,\ldots, n$ **do**
2: **for** $j \leftarrow 1, 2, 3, \ldots, m$ **do**
3: Set $t_1 \leftarrow 0$
4: **for** $k \leftarrow ST[i], ST[i]+1, ST[i]+2,\ldots, (ST[i]+D[i]-1)$ **do**
5: Set $x \leftarrow 0$, $m \leftarrow 1$, $ACO \leftarrow 0$ and $t_1 \leftarrow t_1 + 1$
6: **while** $N[i] \neq x$ **do**
7: **if** $ASSIGN\_UR[m,k,j] = 0$ **then**
8: **if** $m \leq DCE[j,k]$ **then**
9: Set $COST[j] \leftarrow COST[j] + \alpha$
10: Set $ARE[j] \leftarrow ARE[j] + 1$
11: Set $ACO \leftarrow ACO + \alpha$
12: **else**
13: Set $COST[j] \leftarrow COST[j] + DCP[j,k]$
14: Set $ANRE[j] \leftarrow ANRE[j] + 1$
15: Set $ACO \leftarrow ACO + DCP[j,k]$
16: **end if**
17: **end if**
18: Set $m \leftarrow m + 1$ and $x \leftarrow x + 1$
19: **end while**
20: Set $TCOST[j,t_1] \leftarrow ACO$
21: **end for**
22: **end for**
23: Call $FIND\_MIGRATION\_POINT(TCOST, ST, D)$
24: **end for**

6.2.1 Cost Estimation

REOMA estimates the ACO of a UR in all the datacenters using Algorithm 6.1. The ACO is the sum of the cost of RE resources and NRE resources. For this, REOMA iterates from the ST to (ST + D - 1) of each UR in all the datacenters (Line 1 to Line 4). Then, it checks whether the number of required N for the UR is non-zero (Line 6). If it is non-zero, it checks the availability of RE and NRE resource slots (Line 7 to Line 17). If the RE resource slot is available (Line 8), it calculates the cost and counts the RE slots (Line 9 to Line 11). Note that $COST$ is used to keep the cumulative costs, and ACO is used to keep the cost of assigning one UR. We assume that the cost of the RE slot is α, which is generally considered zero. On the other hand, it calculates the cost of the NRE resource slots and counts the NRE slots (Line 13 to Line 15). Finally, the cost of assigning the UR in all the datacenters is estimated and stored in $TCOST$ (Line 20).

6.2.2 Migration Planning

REOMA calls Procedure 7 from line 23 of Algorithm 6.1 to determine the migration points. For this, it determines the slope and intercept by taking the time window and the estimated cost of assigning the UR to the datacenter with respect to the time instances (Line 1 to Line 5). Note that we use a $polyfit()$ function to find the slope and intercept, which is not shown explicitly. This function generates a first-degree polynomial curve or straight line, providing the polynomial coefficients as slope and intercept. Then, REOMA finds the datacenter that estimates the least cost (Line 6 to Line 11). Subsequently, REOMA finds the possible migration points between the least cost datacenter and other available datacenters (Line 12 to Line 21). Note that if there are m datacenters, there are (m - 1) migration points in the worst case. It is noteworthy to mention that the non-least cost datacenters may intersect with each other. However, REOMA does not consider such intersection points as the cost is not the least. Then, it selects a migration point from the set of migration points as follows (Line 22 to Line 45). For instance, if there are (m - 1) migration points with their x-coordinate and y-coordinate as $\{(x_1, y_1), (x_2, y_2), (x_3, y_3),\ldots, (x_{m-1}, y_{m-1})\}$, then it selects the migration point as (x_i, y_i), $1 \leq i, j \leq m$ - 1, $y_i = min(y_1, y_2, y_3,\ldots, y_{m-1})$. It is important to note that the migration points falling outside the time window are not considered (Line 26 and Line 37). The rationality behind selecting the minimum y-coordinate is that it gives the least cost over all the points.

6.2.3 Execution

REOMA calls Procedure 8 from line 46 of Procedure 7 to map the UR to a pair of datacenters by considering the selected migration point (MP). First, it checks whether the migration point is found or not. If not found, it calls Procedure 9 to assign the UR to the least cost datacenter like FABEF (Line 1 to Line 3 of 8). If found, it rounds up the x-coordinate as the time interval is considered a whole number (Line 4). Suppose the slope of the least cost datacenter is greater than that of the datacenter selected for migration. In that case, it assigns the UR to the least cost datacenter between the ST of the UR to the migration point (Line 5 to Line 7), followed by the other datacenter between the migration point to (ST +

Procedure 7 $FIND\_MIGRATION\_POINT(TCOST, ST, D)$

for $j \leftarrow$ 1, 2, 3,. . ., m **do**
    $XVAL \leftarrow [ST[i], ST[i] + 1, ST[i] + 2,. . ., (ST[i] + D[i] - 1)]$
    $YVAL \leftarrow [TCOST[j, 1], [TCOST[j, 2], [TCOST[j, 3],. . ., [TCOST[j, k]]$
    $[SLOPE[j], INTERCEPT[j]] \leftarrow POLYFIT[XVAL, YVAL, 1]$
end for
Set $MCOST \leftarrow COST[1]$ and $MCOST\_DC \leftarrow 1$
for $j \leftarrow$ 1, 2, 3,. . ., m **do**
    if $MCOST > COST[j]$ **then**
        $MCOST \leftarrow COST[j]$ and $MCOST\_DC \leftarrow j$
    end if
end for
for $j \leftarrow$ 1, 2, 3,. . ., m **do**
    if $MCOST\_DC \leftarrow j$ **then**
        Set $XT[j] \leftarrow$ -1 and $YT[j] \leftarrow$ -1
    else if $SLOPE[MCOST\_DC] \leftarrow SLOPE[j]$ **then**
        Set $XT[j] \leftarrow$ -1 and $YT[j] \leftarrow$ -1
    else
        Set $XT[j] \leftarrow \frac{INTERCEPT[j] - INTERCEPT[MCOST\_DC]}{SLOPE[MCOST\_DC] - SLOPE[j]}$
        Set $YT[j] \leftarrow SLOPE[MCOST\_DC] \times XT[j]$ + $INTERCEPT[MCOST\_DC]$
    end if
end for
Set $FMIN\_DC \leftarrow 0$ and $MIG\_CHK \leftarrow 0$
for $j \leftarrow$ 1, 2, 3,. . ., m **do**
    if $XT[j] \leftarrow$ -1 **then**
        continue
    else if $YT[j] \leq 0$ || $XT[j] \leq ST[i]$ || $XT[j] \geq ST[i] + D[i]$ **then**
        continue
    else
        Set $FMIN\_DC \leftarrow j$, $MIN\_VAL \leftarrow YT[j]$ and $MIG\_CHK \leftarrow 1$
        break
    end if
end for
if $FMIN\_DC \leftarrow 0$ **then**
    Set $MIG\_CHK \leftarrow 0$
else
    for $j \leftarrow$ 1, 2, 3,. . ., m **do**
        if $YT[j] \leq 0$ || $XT[j] \leq ST[i]$ || $XT[j] \geq ST[i] + D[i]$ **then**
            continue
        else
            if $MIN\_VAL \geq YT[j]$ **then**
                $MIN\_VAL \leftarrow YT[j]$ and $FMIN\_DC \leftarrow j$
            end if
        end if
    end for
end if
Call $MAPPING\_UR\_DC(MIG\_CHK, ST, D, MCOST\_DC, FMIN\_DC)$

D - 1). Otherwise, it assigns the UR to the datacenter selected for migration, followed by the least cost datacenter (Line 8 to Line 10). In summary, Procedure 8 calls Procedure 9 to assign the UR to a pair of datacenters that results in the least cost.

Procedure 8 $MAPPING\_UR\_DC(MIG\_CHK, ST, D, MCOST\_DC, FMIN\_DC)$

1: **if** MIG_CHK = 0 **then**
2: Call $ASSIGN\_UR\_DC(ST[i]$, $ST[i]$ + $D[i]$ - 1, $N[i]$, $MCOST\_DC)$
3: **else**
4: Set $MP \leftarrow ROUND(XT[FMIN\_DC])$
5: **if** $SLOPE[MCOST\_DC] > SLOPE[FMIN\_DC]$ **then**
6: Call $ASSIGN\_UR\_DC(ST[i]$, MP, $N[i]$, $MCOST\_DC)$
7: Call $ASSIGN\_UR\_DC(MP$ + 1, $ST[i] + D[i] - 1$, $N[i]$, $FMIN\_DC)$
8: **else**
9: Call $ASSIGN\_UR\_DC(ST[i]$, MP, $N[i]$, $FMIN\_DC)$
10: Call $ASSIGN\_UR\_DC(MP$ + 1, $ST[i] + D[i] - 1$, $N[i]$, $MCOST\_DC)$
11: **end if**
12: **end if**

REOMA calls Procedure 9 from line 8, line 6, line 7, line 9 and line 10 of Procedure 8 to assign the UR to a pair of datacenters. For this, it begins with ST to the migration point or the (ST + D - 1) (Line 1). Then, it assigns the nodes to the UR till the required number of nodes is satisfied and updates the OCO, TNRE and TNNRE (Line 1 to Line 13). The above process is repeated from the migration point to the (ST + D - 1).

Procedure 9 $ASSIGN\_UR\_DC(ST, ET, N', SEL\_DC)$

1: **for** $k \leftarrow ST$, ST + 1, ST + 2,. . ., ET **do**
2: Set $x \leftarrow 0$
3: **while** $N' \neq x$ **do**
4: **if** $ASSIGN\_UR[m, k, SEL\_DC]$ = 0 **then**
5: Set $ASSIGN\_UR[m, k, SEL\_DC] \leftarrow i$ and $x \leftarrow x$ + 1
6: **if** $m \leq DCE[SEL\_DC, k]$ **then**
7: Set $OCO \leftarrow OCO + \alpha$ and $TNRE \leftarrow TNRE$ + 1
8: **else**
9: Set $OCO \leftarrow OCO$ + $DCP[SEL\_DC, k]$ and $TNNRE \leftarrow TNNRE$ + 1
10: **end if**
11: **end if**
12: **end while**
13: **end for**

6.2.4 Time Complexity Analysis

The time complexity of REOMA can be determined as follows. Let $|UR| = n$, $|DC| = m$, d be the maximum duration of a set of URs, and o be the maximum amount of resources

among a set of datacenters. In Algorithm 6.1, Line 1 iterates for n URs, requiring $O(n)$ time. Similarly, Line 2 iterates for m datacenters, requiring $O(m)$ time. The inner for loop in Line 4 iterates for d time in the worst case, resulting in $O(d)$ time. The while loop in Line 6 executes o times in the worst case, resulting in $O(o)$ time. Procedure 7 takes $O(m)$ time to find the migration point. Procedure 8 takes $O(d)$ time to map the UR to the datacenter, and Procedure 9 takes $O(d)$ time to assign the UR to the datacenter. Therefore, the overall time complexity of REOMA is $O(nmdo)$ for executing all the URs in the datacenters.

6.2.5 Illustration

We illustrate the existing algorithm, FABEF, and proposed algorithm, REOMA, using seven URs and three datacenters. Each UR is presented as a 3-tuple, namely ST, D and N (see Table 6.1), whereas each data centre is presented as a set of four resources and sixteen-time intervals (see Table 6.2). The RE and NRE resource slots are presented in green and white color, respectively. The cost of NRE resource slots is shown in the first row of each datacenter. The mapping of the URs to the datacenters is explained as follows.

Table 6.1: A set of seven URs with their properties

UR ID	ST	D	N
UR_1	1	5	1
UR_2	1	7	1
UR_3	4	7	2
UR_4	8	5	1
UR_5	9	4	2
UR_6	10	6	1
UR_7	13	4	1

The UR_1 needs one node from time interval 1 to time interval 5. It is matched with three datacenters, as shown in Table 6.3. The ACO in three datacenters is 0.7, 0.6 and 1.0, respectively. As datacenter DC_2 results the minimum ACO, UR_1 is assigned to DC_2, as per FABEF. The OCO of assigning UR_1 is 0.6. The RE and NRE resource slots of assigning UR_1 are 2 and 3, respectively, in FABEF.

In the proposed algorithm, REOMA, the cost is fitted to a polynomial curve using a

Table 6.2: Initial configuration of datacenters with RE and NRE resources

DC_1																
$R\#$	0.1	0.1	0.1	0.3	0.2	0.1	0.5	0.2	0.3	0.2	0.5	0.3	0.3	0.5	0.6	0.5
4																
3																
2																
1																
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC_2																
$R\#$	0.1	0.3	0.2	0.1	0.3	0.5	0.3	0.3	0.4	0.1	0.5	0.4	0.4	0.2	0.5	0.6
4																
3																
2																
1																
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC_3																
$R\#$	0.3	0.6	0.2	0.3	0.1	0.5	0.3	0.3	0.4	0.2	0.1	0.2	0.5	0.3	0.2	0.1
4																
3																
2																
1																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

set of time intervals and cost values. Note that it can be performed using MATLAB's $polyfit(x, y, n)$ function in which the x represents the time intervals, y represents the cost values, and n is the polynomial degree to fit. The slope and intercept values of datacenters are (0.06, -0.04), (0.07, -0.09) and (-0.01, 0.23), respectively. REOMA determines the intersection points between the least cost datacenter and other datacenters, i.e., (DC_2, DC_1) and (DC_2, DC_3). Here, the intersection point of DC_2 and DC_1 is [5.00 (i.e., $\frac{-0.04-(-0.09)}{0.07-0.06}$), 0.26 (i.e., 0.06 $\times$ 5.00 + (-0.04) or 0.07 $\times$ 5.00 + (-0.09)), and intersection point of DC_2 and DC_3 is [4.00 (i.e., $\frac{0.23-(-0.09)}{0.07-(-0.01)}$), 0.19 (i.e., 0.07 $\times$ 4.00 + (-0.09) or (-0.01) $\times$ 4.00 + 0.23)]. As the y represents the cost values, the minimum cost value is the result between datacenters DC_2 and DC_3 (i.e., 0.19). Therefore, migration occurs between these two datacenters, and the migration point is determined as the time interval four based on the corresponding y-coordinate value, as shown in Fig. 6.1. However, the slope of datacenter DC_2 is greater than the slope of datacenter DC_3. Therefore, UR_1 is assigned to datacenter DC_2 from time intervals one to four and subsequently migrated to DC_3 after time interval four, as shown in Table 6.4. The cost of assigning UR_1 to datacenter DC_2 is 0.3 (0 + 0 + 0.2

Table 6.3: Matching of UR_1 to three datacenters

t	1	2	3	4	5	OCO
DC_1	0.0	0.1	0.1	0.3	0.2	0.7
DC_2	0.0	0.0	0.2	0.1	0.3	**0.6**
DC_3	0.0	0.6	0.0	0.3	0.1	1.0

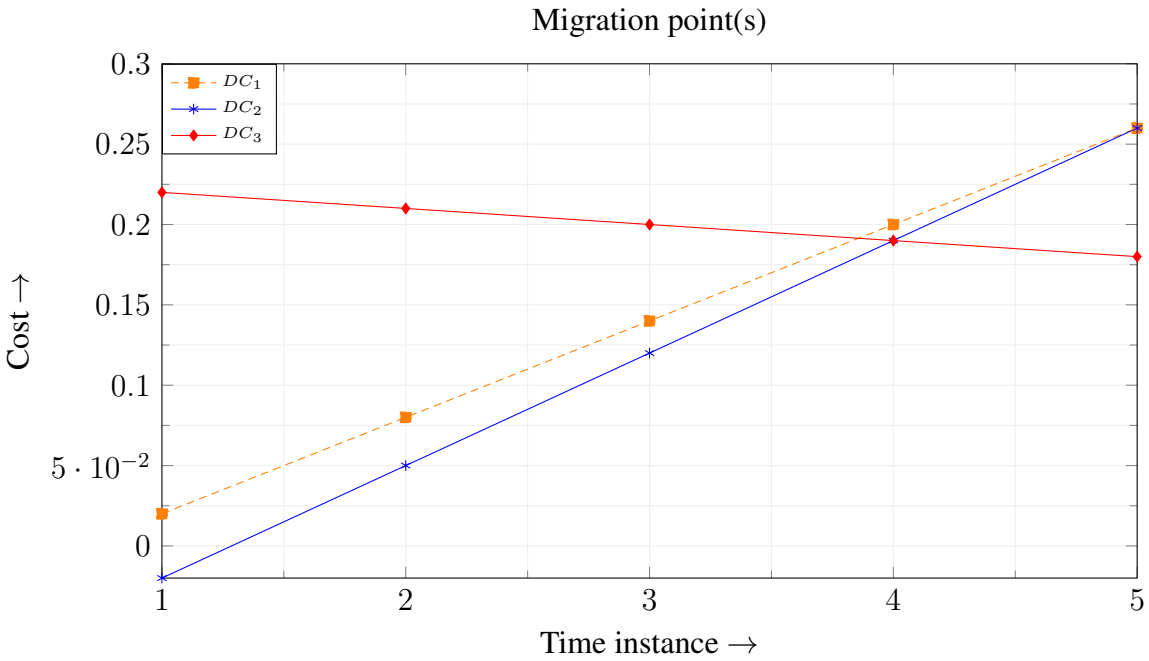

Figure 6.1: Migration points among the datacenters, especially (DC_2, DC_3) and (DC_2, DC_1).

+ 0.1) and datacenter DC_3 is 0.1 by ignoring the migration cost. The total cost of assigning UR_1 is 0.4. The RE and NRE resource slots of assigning UR_1 are 2 and 3, respectively.

The UR_2 needs one node from time interval 1 to time interval 7. It is matched with three datacenters, as shown in Table 6.5. The ACO in three datacenters is 1.2, 1.2 and 1.0, respectively. As datacenter DC_3 results the minimum ACO, UR_2 is assigned to DC_3, as per FABEF. The OCO of assigning UR_1 is 1.0. The RE and NRE resource slots of assigning UR_2 is 4 and 3, respectively, in FABEF.

In the proposed algorithm, REOMA, the slope and intercept values of datacenters are (0.0500, -0.0286), (0.0607, -0.0286) and (-0.0393, 0.3000), respectively. Then, REOMA determines the intersection points between the least cost datacenter and other datacenters, i.e., (DC_3, DC_1) and (DC_3, DC_2). Here, the intersection point of DC_3 and DC_1 is (3.6797,

Table 6.4: Migration of UR_1 from DC_2 to DC_3

DC_1																
$R\#$	0.1	0.1	0.1	0.3	0.2	0.1	0.5	0.2	0.3	0.2	0.5	0.3	0.3	0.5	0.6	0.5
4																
3																
2																
1																
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC_2																
$R\#$	0.1	0.3	0.2	0.1	0.3	0.5	0.3	0.3	0.4	0.1	0.5	0.4	0.4	0.2	0.5	0.6
4																
3																
2																
1	UR_1	UR_1	UR_1	UR_1												
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC_3																
$R\#$	0.3	0.6	0.2	0.3	0.1	0.5	0.3	0.3	0.4	0.2	0.1	0.2	0.5	0.3	0.2	0.1
4																
3																
2																
1					UR_1											
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

0.1554), and intersection point of DC_3 and DC_2 is (3.2860, 0.1709). As the y represents the cost values, the minimum cost value is the result between datacenters DC_3 and DC_1 (i.e., 0.1554). Therefore, migration occurs between these two datacenters, and the migration point is determined as the time interval four (i.e., round to nearest whole number of 3.6), as shown in Fig. 6.2. However, the slope of datacenter DC_1 is greater than the slope of datacenter DC_3. Therefore, UR_2 is assigned to datacenter DC_1 from time intervals one to four and subsequently migrated to DC_3 after time interval four, as shown in Table 6.6. The cost of assigning UR_2 to datacenter DC_1 is 0.5 and datacenter DC_3 is 0.1 by ignoring the migration cost. The OCO of assigning UR_2 is 0.6. The TNRE and TNNRE of assigning UR_2 is 3 and 4, respectively. Similarly, UR_3 to UR_7 are assigned to (DC_1, DC_2), (DC_1, DC_3), (DC_1, DC_3), (DC_3, DC_1) and (DC_2, DC_3), respectively. The OCO of assigning these URs are shown in Table 6.7 along with the TNRE and TNNRE. The Gantt chart for assigning seven URs to three datacenters is shown in Table 6.5. We compare the proposed algorithm, REOMA, with the FABEF, HAREF, and RR algorithms, as illustrated in Table 6.6. The comparison demonstrates the superiority of REOMA over the other three

Table 6.5: Matching of UR_2 to three datacenters

t	1	2	3	4	5	6	7	OCO
DC_1	0.0	0.1	0.1	0.3	0.2	0	0.5	1.2
DC_2	0.1	0.0	0.2	0.1	0.3	0.5	0.3	1.2
DC_3	0.0	0.6	0.0	0.3	0.1	0.0	0.0	**1.0**

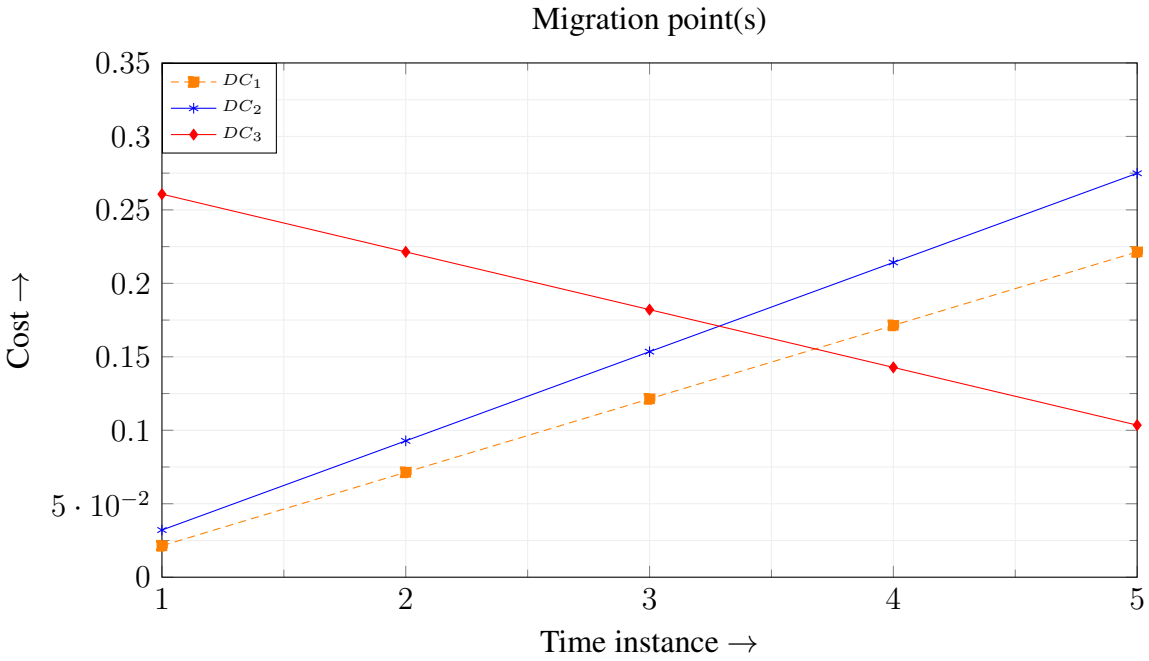

Figure 6.2: Migration points among the datacenters, especially (DC_1, DC_3).

algorithms in terms of OCO, TNRE, and TNNRE.

6.3 Performance Metrics, Datasets and Simulation Results

This section discusses three performance metrics used to compare the proposed algorithm, REOMA, and three benchmark algorithms, followed by a description of datasets and simulation results.

6.3.1 Performance Metrics

Three performance metrics, namely OCO, TNRE and TNNRE are used to compare the proposed and existing algorithms. They are defined as follows. OCO is the total cost incurred to execute all the URs irrespective of the datacenters and energy sources (i.e., RE

Table 6.6: Gantt chart for REOMA

DC_1																
R #	0.1	0.1	0.1	0.3	0.2	0.1	0.5	0.2	0.3	0.2	0.5	0.3	0.3	0.5	0.6	0.5
4																
3				UR_3					UR_5	UR_5						
2				UR_3	UR_3	UR_3			UR_5	UR_5						
1	UR_2	UR_2	UR_2	UR_2	UR_3	UR_3		UR_4	UR_4	UR_4			UR_6	UR_6	UR_6	
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC_2																
R #	0.1	0.3	0.2	0.1	0.3	0.5	0.3	0.3	0.4	0.1	0.5	0.4	0.4	0.2	0.5	0.6
4																
3																
2							UR_3	UR_3	UR_3	UR_3						
1	UR_1	UR_1	UR_1	UR_1			UR_3	UR_3	UR_3	UR_3			UR_7	UR_7		
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC_3																
R #	0.3	0.6	0.2	0.3	0.1	0.5	0.3	0.3	0.4	0.2	0.1	0.2	0.5	0.3	0.2	0.1
4											UR_6	UR_6				
3											UR_5	UR_5				
2					UR_2						UR_5	UR_5				
1					UR_1	UR_2	UR_2			UR_6	UR_4	UR_4			UR_7	UR_7
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Table 6.7: Comparison of OCO, TNRE and TNNRE for REOMA, FABEF, HAREF and RR

UR ID	REOMA	FABEF	HAREF	RR
UR_1	0.4	0.6	0.6	0.7
UR_2	0.6	1.0	1.0	1.4
UR_3	1.8	2.5	2.5	3.1
UR_4	0.3	0.8	0.8	0.8
UR_5	1.6	1.8	2.6	1.9
UR_6	0.8	1.3	1.3	1.5
UR_7	0.5	1.1	1.3	0.8
OCO	6.0	9.1	10.1	10.2
TNRE	17	16	17	14
TNNRE	32	33	32	35

and NRE). If a UR is executed in two datacenters, the cost of both datacenters is summed to calculate the OCO. However, we ignore the cost of RE resources and migration for simplicity in calculation. Mathematically,

$$OCO = \sum_{k=1}^{|UR|} \sum_{i=1}^{|DC|} \sum_{j=ST}^{ST+D-1} COST[i,j] \times Y[k,i,j] \tag{6.1}$$

where

$$Y[k,i,j] = \begin{cases} 1 & \text{If } DC_i \text{ accommodates } UR_k \text{ at time instance } j \\ 0 & \text{Otherwise} \end{cases} \tag{6.2}$$

$TNRE$ is the number of RE resource slots used for executing all the URs irrespective of the datacenters.

$TNNRE$ is the number of NRE resource slots used for executing all the URs irrespective of the datacenters. Note that the sum of $TNRE$ and $TNNRE$ is the total number of resource slots used for executing all the URs.

6.3.2 Datasets

To our knowledge, no existing dataset considers all the properties of URs and datacenters. Therefore, a dataset is prepared by taking all the properties of URs, namely ST, D and N and datacenters, namely R, ARE, ANRE and CO. The range of ST, D and N is [1 $\sim$ 100], [10 $\sim$ 25] and [10 $\sim$ 100], respectively. On the other hand, the range of R, ARE, ANRE, and COST is [10 $\sim$ 400], [10 $\sim$ 200], [300 $\sim$ 400] and [1 $\sim$ 100], respectively.

The number of URs is generated between 200 and 2000, with a gap of 200, whereas the number of datacenters is generated between 20 to 200, with a gap of 20. Ten datasets are created, ranging from 200 URs assigned to 20 datacenters (shown as 200 $\times$ 20) to 2000 URs assigned to 200 datacenters (shown as 2000 $\times$ 200). In each dataset, we generate five instances and average the results. Each instance of the dataset is given as input to produce the results in terms of OCO, TNRE and TNNRE. Notably, these datasets are generated using uniform distribution and following the Monte Carlo simulation process to avoid bias.

6.3.3 Simulation Results

We perform all the simulations using a system with an Intel(R) Core(TM) i5-10310U CPU @ 1.70 GHz 2.21 GHz, 8 GB RAM, a 64-bit Windows 11 operating system, an x64-based processor and MATLAB R2021a licensed version. However, our simulation is not limited to the above specifications. We conduct 50 simulations per algorithm and a total of 200 simulations. The simulation times are ranging from seconds to hours. Note that the simulation times depend on the system configuration.

We categorize the datasets into two types, namely low and high datasets. The first five datasets are referred to as low datasets, whereas the last five are called high datasets. We compare the proposed algorithm REOMA with three benchmark algorithms, namely FABEF, HAREF and RR, in terms of OCO as shown in Figure 6.3 and Figure 6.4 for low and high datasets, respectively. It is noteworthy that FABEF is the best-performing benchmark algorithm regarding OCO as stated in [46, 70, 71, 115]. However, the proposed algorithm REOMA outperforms the FABEF and other benchmark algorithms, as seen from Figure 6.3 and Figure 6.4.

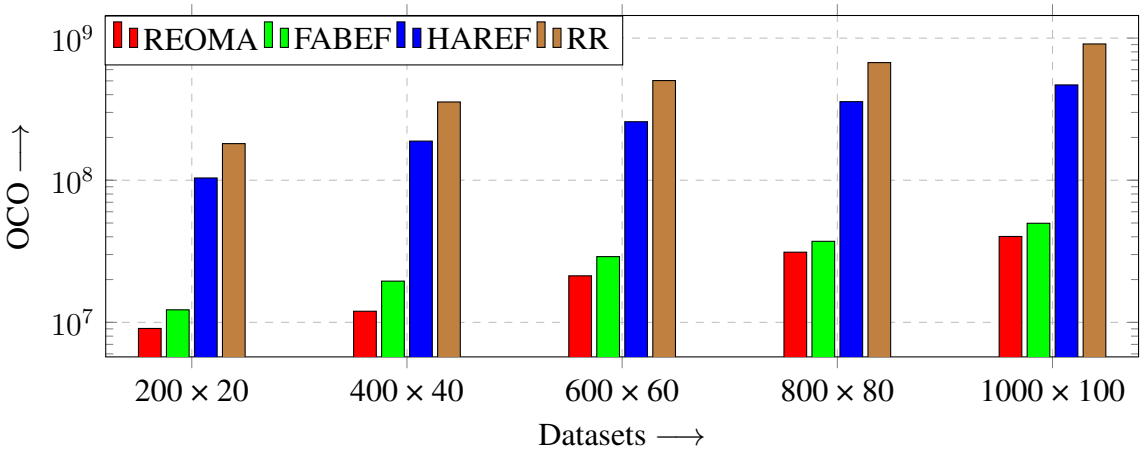

Figure 6.3: Comparison of OCO for REOMA, FABEF, HAREF and RR using low datasets.

The rationality behind the better performance is as follows.

1. The migration point is determined between the least-cost datacenter and other datacenters. The cost cannot be optimized further without a viable migration point. If a viable point is found, it signifies that there is a datacenter that can provide better cost

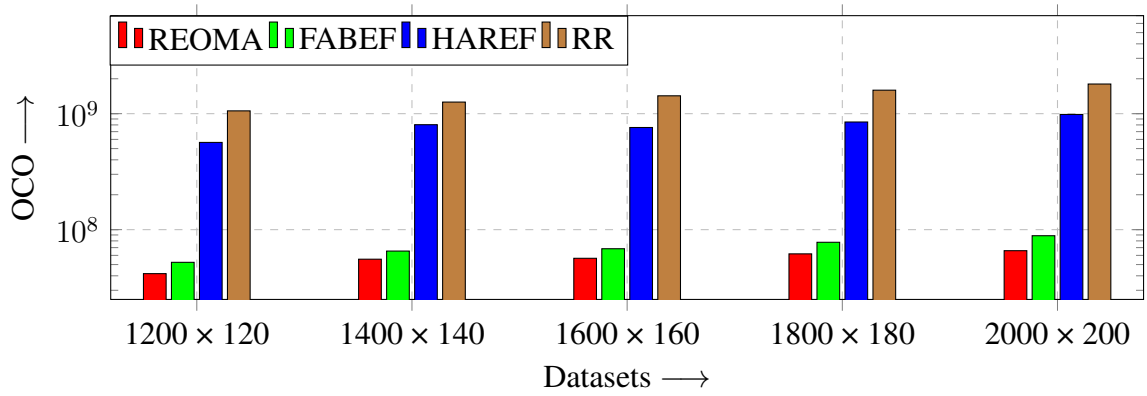

Figure 6.4: Comparison of OCO for REOMA, FABEF, HAREF and RR using high datasets.

than the least-cost datacenter. As a result, when a UR migrates from one datacenter to another, it results in the better OCO.

2. The assignment cost of a UR depends on the usage of RE and NRE resources. If all the resources are RE, then the cost is negligible. As the migration point is determined based on cost, resource usage positively impacts the proposed algorithm's performance.

Next, we compare the proposed and existing benchmark algorithms for TNRE by considering ten datasets, as shown in Figure 6.5 and Figure 6.6, respectively. As seen in the figures, HAREF outperforms all the algorithms as it is the best-performing algorithm for TNRE. REOMA outperforms all other algorithms except HAREF and utilizes the RE resources efficiently. The rationality behind this is that REOMA aims to minimize the OCO and not fully emphasize the RE resources in the scheduling decision process. However, OCO is a critical factor compared to TNRE. Therefore, REOMA is preferable compared to HAREF.

Next, the proposed and existing algorithms are compared with another performance matrix, TNNRE, by taking both low and high datasets, as shown in Figure 6.7 and Figure 6.8, respectively. It is obvious to see that HAREF outperforms all the algorithms. REOMA performs next to the HAREF. The rationality behind this is that maximizing the TNRE is proportional to minimizing the TNNRE. As OCO is the primary concern, REOMA is more suitable than all other algorithms.

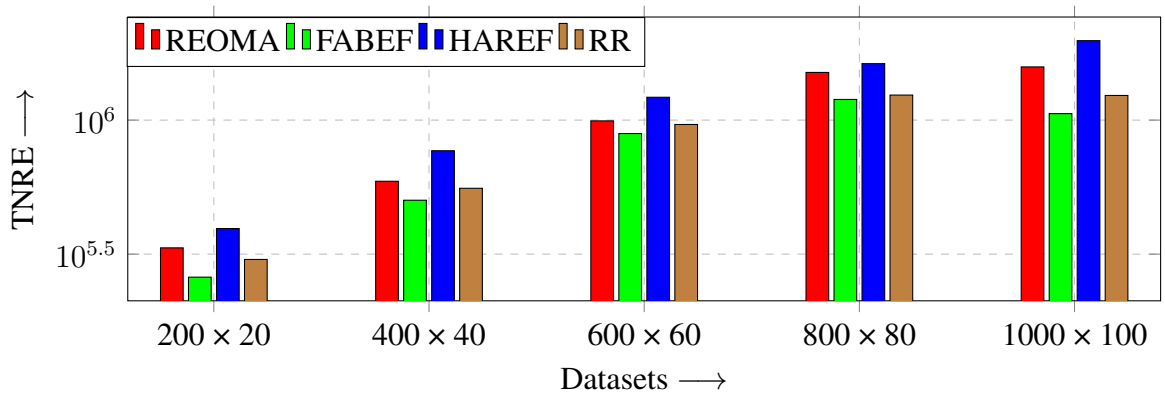

Figure 6.5: Comparison of TNRE for REOMA, FABEF, HAREF and RR using low datasets.

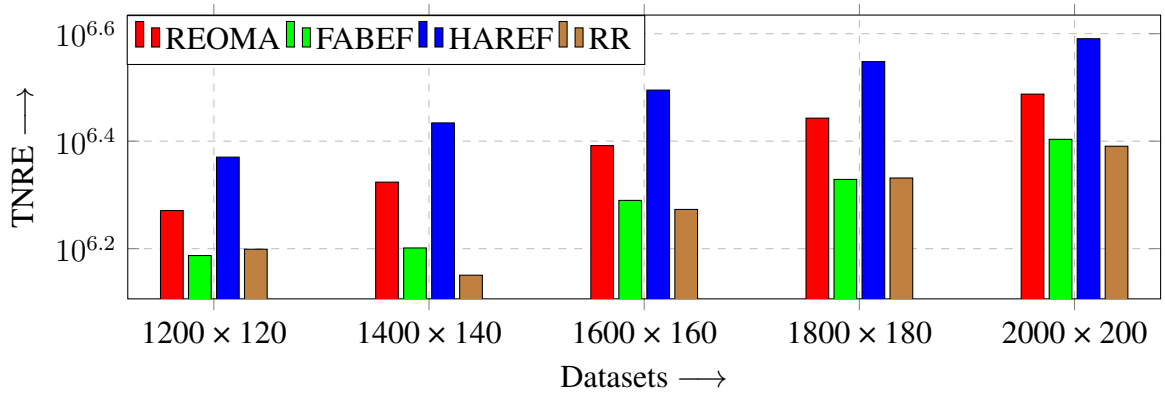

Figure 6.6: Comparison of TNRE for REOMA, FABEF, HAREF and RR using high datasets.

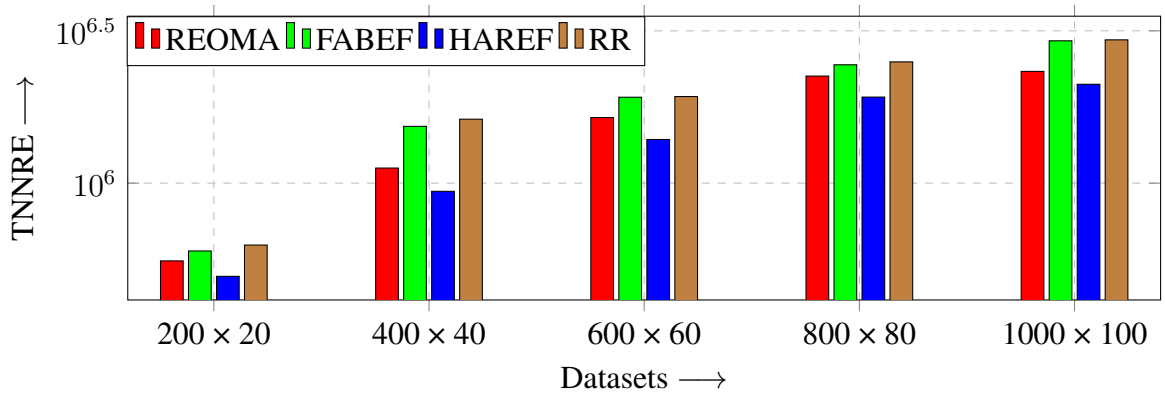

Figure 6.7: Comparison of TNNRE for REOMA, FABEF, HAREF and RR using low datasets.

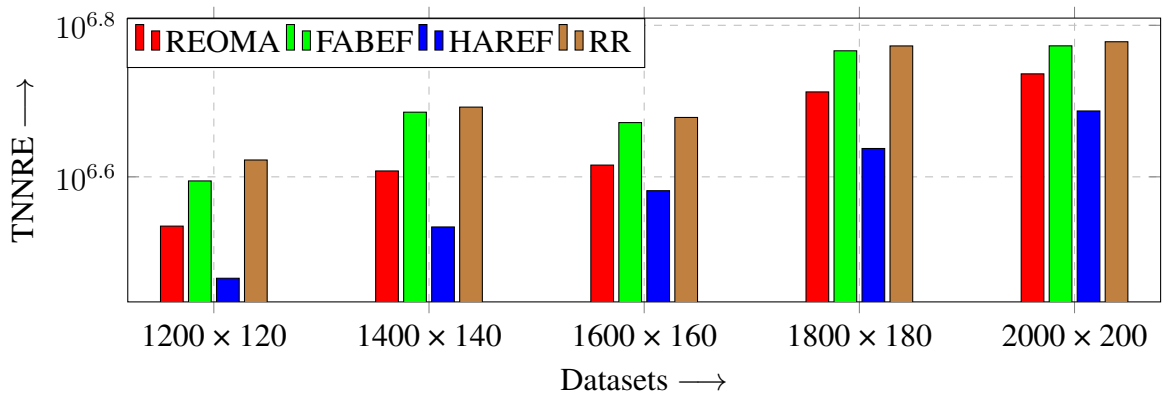

Figure 6.8: Comparison of TNNRE for REOMA, FABEF, HAREF and RR using high datasets.

6.4 Summary

This chapter has presented a migration algorithm, REOMA, for geo-distributed datacenters. The algorithm is an extension of one of the benchmark algorithms, FABEF. REOMA focuses on minimizing the OCO by relocating the URs from one datacenter to another datacenter. The main principle behind this algorithm is to determine a suitable migration point between the least-cost datacenter and other datacenters. The algorithm has been shown to require $O(nmdo)$ for n URs, m datacenters, d duration and o resources. All four algorithms have been evaluated by carrying out 200 simulations using ten datasets and three performance metrics. Each simulation took seconds to hours based on our system configuration. The comparison results show the applicability of REOMA over three benchmark algorithms concerning the performance metrics. However, the proposed algorithm currently supports only a one-time migration between two datacenters, regardless of the UR size. To improve this, we could implement multiple migrations among a set of datacenters based on the UR size, incorporating thresholds on both the migration process and the number of datacenters involved to prevent excessive overhead.

Chapter 7

Conclusion and Future Scope

In this chapter, we present the conclusion of the thesis along with notable remarks, followed by a discussion of future research directions.

7.1 Conclusion

In this thesis, we primarily address various RE-based scheduling algorithms for geographically distributed datacenters. We begin with an overview of cloud computing, RE and NRE generation, along with relevant statistics and challenges. We then discuss the motivation for developing scheduling algorithms that incorporate RE and NRE sources, followed by outlining four specific objectives. The contributions of this thesis in response to these objectives are briefly highlighted. Subsequently, we provide a detailed review of various scheduling algorithms, focusing on cost, RE and NRE generation, UN and migration, along with their pros, cons, and potential areas for improvement. The main contributions of this thesis are detailed from Chapter 3 to Chapter 6, where we propose ten algorithms aimed at minimizing processor and memory costs, UN, UN levels, and migration to reduce OCO, TNNRE, UNT and UNCO, while maximizing TNRE.

In Chapter 3, we propose two algorithms for geo-distributed datacenters. First, we introduce a scheduling algorithm called PM-FABEF, designed to minimize the OCO of assigning URs to datacenters. In PM-FABEF, the costs of processor and memory nodes are calculated for each datacenter, and the datacenter with the lowest ACO is selected for UR

assignment. PM-FABEF has a time complexity of $O(nmdo)$, where n is the number of URs, m is the number of datacenters, d is the maximum duration, and o is the maximum number of resources. Second, we propose another scheduling algorithm, PM-HAREF, which aims to maximize the TNRE while assigning URs to datacenters. In PM-HAREF, the availability of processor and memory nodes powered by RE is assessed for each datacenter, and the datacenter with the highest TNRE is chosen for UR assignment. PM-HAREF also operates with a time complexity of $O(nmdo)$. We simulated these two RE-based scheduling algorithms using fifty instances of ten datasets and compared the results with three benchmark algorithms: FABEF, HAREF, and RR. The simulations demonstrate that the proposed algorithms are more efficient than the benchmark algorithms in terms of OCO, TNRE, and TNNRE, depending on their applicability. Our findings indicate that the relationship between OCO and TNRE is non-linear.

In Chapter 4, we present three RE-based scheduling algorithms: UN-FABEF, UN-HAREF, and UN-RR. These algorithms extend three benchmark algorithms (FABEF, HAREF, and RR) by incorporating the management of UN for RE and NRE resources in geo-distributed datacenters. The primary objective of these proposed algorithms is to minimize the OCO, TNNRE, UNT and UNCO, and maximize the TNRE. UNT is calculated based on the availability of RE and NRE resources. UN-FABEF and UN-HAREF have a time complexity of $O(nmdo)$, while UN-RR, due to its simplicity, has a time complexity of $O(ndo)$. Our simulations, comprising 150 runs using ten datasets, demonstrate that UN-FABEF outperforms in terms of OCO, UNT, and UNCO. At the same time, UN-HAREF excels in maximizing the TNRE.

In Chapter 5, we have presented four algorithms, UNL-FABEF, UNL-HAREF, UNL-RR and UNL-MOSA, by incorporating user and CSP perspectives for geo-distributed datacenters. These algorithms consider three levels, low, medium and high, from a user perspective and the UN percentage between 1% and 100% from the CSP perspective. The proposed algorithms incorporate both user and CSP perspectives in three benchmark algorithms, FABEF, HAREF and RR, to minimize the OCO, TNNRE, and UNT and UNCO and maximize the TNRE. UNL-MOSA is designed to balance the trade-offs between UNL-FABEF and UNL-HAREF, achieving a balance between OCO and TNRE. UNL-FABEF,

UNL-HAREF, and UNL-MOSA have a time complexity of $O(nmdo)$, whereas UNL-RR operates with a time complexity of $O(ndo)$. Simulation results were obtained using ten datasets, which varied in size from 200 to 2000 URs, 20 to 200 datacenters, and 10 to 400 resources per datacenter. The performance metrics demonstrate the applicability and efficiency of the proposed algorithms.

In Chapter 6, we have proposed a RE-based migration algorithm, called REOMA, for geo-distributed datacenters. REOMA improves the performance of the best performing least cost benchmark algorithm FABEF by migrating the UR between the datacenters. It takes advantage of the cost estimation of a UR in various datacenters, thereby determining a suitable migration point. The algorithm is shown to require $O(nmdo)$ time. REOMA is compared with three benchmark algorithms, including FABEF, using fifty instances of ten datasets (i.e., 200 to 2000 URs and 20 to 200 geo-distributed datacenters) and three performance metrics. The comparison results show that REOMA performs better than other algorithms in terms of OCO.

7.2 Future Scope

The proposed RE-based scheduling algorithms perform significantly well according to their applicability. However, they can be further extended in the future by adopting the following enhancements.

1. PM-FABEF, UN-FABEF, UNL-FABEF, REOMA and FABEF aim for the minimum OCO, while PM-HAREF, UN-HAREF, UNL-HAREF, and HAREF aim for the maximum TNRE. However, reducing NRE usage doesn't always equate to cost reduction. Although their relationship is expected to be linear, it's actually nonlinear. Consequently, we intend to investigate this relationship and integrate it into our proposed algorithms as a future extension of our research.

2. All proposed algorithms consider the availability of RE based on time windows. However, they haven't been thoroughly examined regarding various RE sources, such as solar, wind, hydropower, and biomass and their possible costs. Alternatively, RE

sources are not explicitly mentioned in the proposed works and are not equally capable. It may be further explored by considering the generation of RE in the location of the datacenters, thereby determining the cost to make it more realistic. This aspect needs exploration in future research endeavours.

3. UN-FABEF, UN-HAREF, UN-RR, UNL-FABEF, UNL-HAREF and UNL-RR scheduling algorithms are proposed to manage the UN and its level. UN can be modelled by considering internal and external factors. However, the detailed study of these factors is not explicitly shown in the context of these scheduling algorithms. We intend to investigate such factors as a future extension of our research.

4. UN is shown in terms of the percentage of UN-RE and UN-NRE in the case of UN-FABEF, UN-HAREF and UN-RR. However, the percentage of different RE sources (i.e., solar, hydropower, wind, etc.) is not considered for simplicity. Moreover, the UN about datacenters varies concerning locations. However, it is not considered to avoid the complexity of the algorithms. These can be further explored in future work.

5. UNL-FABEF, UNL-HAREF, and UNL-RR are proposed to incorporate UN levels from user and CSP perspectives. Here, the resource slots are considered only in terms of nodes. However, they can be considered in terms of processor, memory and storage. In this case, the UR modelling should be in terms of processor, memory and storage nodes instead of only nodes.

6. The URs considered in all the proposed algorithms can have different duration and node requirements concerning the datacenters as the type of VMs varies with respect to the CSPs. These can be further explored in future work.

7. REOMA relocates the UR from one datacenter to another datacenter based on a migration point. However, it only allows for a one-time migration between two datacenters, regardless of the size of the UR. Further exploration could involve considering multiple migrations among a set of datacenters based on feasibility criteria. Additionally, establishing an appropriate threshold for the number of migrations and defining zones for datacenters could help mitigate overhead concerns. Moreover, the

migration cost is not explicitly shown in the proposed algorithm, which can be further explored to make it realistic.

8. As there is no benchmark dataset, the proposed algorithms are validated using synthetic datasets by following the uniform distribution and the Monte Carlo simulation process. However, these works can be validated by taking some parameters from the real workload traces (e.g., Google cluster-usage traces [125–127]).

9. The proposed algorithms have considered various constraints to describe the user requirements for executing URs. However, these algorithms can be extended by introducing further constraints, such as latency, deadline, criticality level, priority, failover time, recovery time, security factors, encryption time, decryption time, and authentication time, to model real-time execution and security needs.

10. The proposed algorithms can be extended by applying machine learning-based forecasting models, which can predict renewable sources' short-term and long-term availability by incorporating data from satellites, weather stations and historical patterns, to deal with highly dynamically variable availability of renewable resources.

In our future work, we will focus on these enhancements to develop novel RE-based scheduling algorithms that are more realistic.

Bibliography

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

[2] Mohit Kumar, Subhash Chander Sharma, Anubhav Goel, and Santar Pal Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 143:1–33, 2019.

[3] AR Arunarani, Dhanabalachandran Manjula, and Vijayan Sugumaran. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, 91:407–415, 2019.

[4] Jing Wei and Xin-fa Zeng. Optimal computing resource allocation algorithm in cloud computing based on hybrid differential parallel scheduling. *Cluster Computing*, 22(3):7577–7583, 2019.

[5] Sanjaya K Panda and Prasanta K Jana. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing*, 71:1505–1533, 2015.

[6] Sanjaya K Panda, Indrajeet Gupta, and Prasanta K Jana. Task scheduling algorithms for multi-cloud systems: allocation-aware approach. *Information Systems Frontiers*, pages 1–19, 2017.

[7] Sanjaya K Panda and Prasanta K Jana. Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment. *Information Systems Frontiers*, 20(2):373–399, 2018.

[8] Pawan Kumar and Rakesh Kumar. Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 51(6):1–35, 2019.

[9] Sanjaya K Panda and Prasanta K Jana. Sla-based task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing*, 73(6):2730–2762, 2017.

[10] Sanjaya K Panda and Prasanta K Jana. Load balanced task scheduling for cloud computing: a probabilistic approach. *Knowledge and Information Systems*, pages 1–25, 2018.

[11] Madhu Sudan Kumar, Indrajeet Gupta, Sanjaya K Panda, and Prasanta K Jana. Granularity-based workflow scheduling algorithm for cloud computing. *The Journal of Supercomputing*, 73(12):5440–5464, 2017.

[12] Prajwol Sangat, Maria Indrawan-Santiago, and David Taniar. Sensor data management in the cloud: Data storage, data ingestion, and data retrieval. *Concurrency and Computation: Practice and Experience*, 30(1):e4354, 2018.

[13] Sushant Goel, Hema Sharda, and David Taniar. Replica synchronisation in grid databases. *International Journal of Web and Grid Services*, 1(1):87–112, 2005.

[14] David Taniar and Sushant Goel. Concurrency control issues in grid databases. *Future Generation Computer Systems*, 23(1):154–162, 2007.

[15] David Taniar, Clement HC Leung, Wenny Rahayu, and Sushant Goel. *High-performance parallel database processing and grid databases*. John Wiley & Sons, 2008.

[16] Vladimir Sumina. Cloud computing statistics, facts and trends for 2022. https://www.cloudwards.net/cloud-computing-statistics/. Accessed on 20th April 2022.

[17] Kai Hwang, Xiaoying Bai, Yue Shi, Muyang Li, Wen-Guang Chen, and Yongwei Wu. Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *IEEE Transactions on parallel and distributed systems*, 27(1):130–143, 2015.

[18] Farrukh Nadeem. Evaluating and ranking cloud iaas, paas and saas models based on functional and non-functional key performance indicators. *IEEE Access*, 10:63245–63257, 2022.

[19] Munmun Saha, Sanjaya Kumar Panda, and Suvasini Panigrahi. A hybrid multi-criteria decision making algorithm for cloud service selection. *International Journal of Information Technology*, 13(4):1417–1422, 2021.

[20] Centrify and CensusWide. Pandemic has accelerated cloud transformation for nearly half of organizations. https://www.securitymagazine.com/articles/93654. Accessed on 27th April 2023.

[21] Zhiwei Cao, Xin Zhou, Han Hu, Zhi Wang, and Yonggang Wen. Towards a systematic survey for carbon neutral data centers. *IEEE Communications Surveys & Tutorials*, 2022.

[22] Sanjaya K Panda and Prasanta K Jana. An efficient request-based virtual machine placement algorithm for cloud computing. In *International Conference on Distributed Computing and Internet Technology*, pages 129–143. Springer, 2017.

[23] Sanjaya K Panda and Prasanta K Jana. An efficient energy saving task consolidation algorithm for cloud computing systems. In *2014 International Conference on Parallel, Distributed and Grid Computing*, pages 262–267. IEEE, 2014.

[24] Priyanka Panigrahi, S Panda, and C Tripathy. Energy efficient task consolidation algorithms for cloud computing systems. *International Journal of Information Processing*, 94:34–45, 2015.

[25] Sanjaya Kumar Panda, Sanju Parida, Sourav Kumar Bhoi, Sanjib Kumar Nayak, and Satyabrata Das. An efficient virtual machine management algorithm for vehicular clouds. In *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 682–688. IEEE, 2018.

[26] Sanjaya Kumar Panda, Shradha Surachita Nanda, and Sourav Kumar Bhoi. A pair-based task scheduling algorithm for cloud computing environment. *Journal of King Saud University-Computer and Information Sciences*, 34(1):1434–1445, 2022.

[27] Keke Gai, Meikang Qiu, Hui Zhao, and Xiaotong Sun. Resource management in sustainable cyber-physical systems using heterogeneous cloud computing. *IEEE Transactions on Sustainable Computing*, 3(2):60–72, 2017.

[28] Benazir Neha, Sanjaya Kumar Panda, and Pradip Kumar Sahu. An efficient task mapping algorithm for osmotic computing-based ecosystem. *International Journal of Information Technology*, 13(4):1303–1308, 2021.

[29] Conn. STAMFORD. Gartner forecasts worldwide public cloud end-user spending to reach nearly $600 billion in 2023. https://www.gartner.com/en/newsroom/. Accessed on 23rd April 2024.

[30] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. In *Advances in computers*, volume 82, pages 47–111. Elsevier, 2011.

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

[32] Ching-Hsien Hsu, Kenn D Slagter, Shih-Chang Chen, and Yeh-Ching Chung. Optimizing energy consumption with task consolidation in clouds. *Information Sciences*, 258:452–462, 2014.

[33] Atefeh Khosravi, Adel Nadjaran Toosi, and Rajkumar Buyya. Online virtual machine migration for renewable energy usage maximization in geographically distributed cloud data centers. *Concurrency and Computation: Practice and Experience*, 29(18):e4125, 2017.

[34] Md Asif Thanedar and Sanjaya Kumar Panda. An energy-efficient resource allocation algorithm for managing on-demand services in fog-enabled vehicular ad hoc networks. *International Journal of Web and Grid Services*, 20(2):135–158, 2024.

[35] Sohan Kumar Pande, Sanjaya Kumar Panda, and Satyabrata Das. An energy-aware service management algorithm for vehicular cloud computing. In *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2021*, pages 22–33. Springer, 2022.

[36] Benazir Neha, Sanjaya Kumar Panda, Pradip Kumar Sahu, and David Taniar. Energy and latency-balanced osmotic-offloading algorithm for healthcare systems. *Internet of Things*, 26:101176, 2024.

[37] Munmun Saha, Sanjaya Kumar Panda, and Suvasini Panigrahi. Distributed computing security: issues and challenges. *Cyber security in parallel and distributed computing: concepts, techniques, applications and case studies*, pages 129–138, 2019.

[38] Kshira Sagar Sahoo, Sanjaya Kumar Panda, Sampa Sahoo, Bibhudatta Sahoo, and Ratnakar Dash. Toward secure software-defined networks against distributed denial of service attack. *The Journal of Supercomputing*, pages 1–46, 2019.

[39] Tian Wang, Yuzhu Liang, Yujie Tian, Md Zakirul Alam Bhuiyan, Anfeng Liu, and A Taufiq Asyhari. Solving coupling security problem for sustainable sensor-cloud systems based on fog computing. *IEEE Transactions on Sustainable Computing*, 6(1):43–53, 2019.

[40] Ehsan Ahvar, Anne-Cécile Orgerie, and Adrien Lebre. Estimating energy consumption of cloud, fog, and edge computing infrastructures. *IEEE Transactions on Sustainable Computing*, 7(2):277–288, 2019.

[41] Md Sabbir Hasan, Frederico Alvares, Thomas Ledoux, and Jean-Louis Pazat. Investigating energy consumption and performance trade-off for interactive cloud application. *IEEE Transactions on Sustainable computing*, 2(2):113–126, 2017.

[42] Uptime Institute. 2021 data center industry survey. https://uptimeinstitute.com/about-ui/press-releases/2022-global-data-center-survey-reveals-strong-industry-growth. Accessed on 20th April 2022.

[43] Sanjib Nayak, Sanjaya Panda, and Satyabrata Das. Renewable energy-based resource management in cloud computing: A review. In *Advances in Distributed Computing and Machine Learning*, pages 45–56. Springer, 2020.

[44] Amanda Jayanetti, Saman Halgamuge, and Rajkumar Buyya. Multi-agent deep reinforcement learning framework for renewable energy-aware workflow scheduling on distributed cloud data centers. *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[45] Hwei-Ming Chung, Sabita Maharjan, Yan Zhang, Frank Eliassen, and Kai Strunz. Optimal energy trading with demand responses in cloud computing enabled virtual power plant in smart grids. *IEEE Transactions on Cloud Computing*, 10(1):17–30, 2021.

[46] Adel Nadjaran Toosi and Rajkumar Buyya. A fuzzy logic-based controller for cost and energy efficient load balancing in geo-distributed data centers. In *Proceedings of the 8th International Conference on Utility and Cloud Computing*, pages 186–194. IEEE Press, 2015.

[47] Hari Sowrirajan and George Wang. Application of auction theory in cloud computing and renewable energy. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 1–3. IEEE, 2023.

[48] Aref Karimiafshar, Massoud Reza Hashemi, Mohammad Reza Heidarpour, and Adel N Toosi. A request dispatching method for efficient use of renewable energy in fog computing environments. *Future Generation Computer Systems*, 114:631–646, 2021.

[49] Aref Karimiafshar, Massoud Reza Hashemi, Mohammad Reza Heidarpour, and Adel N Toosi. Effective utilization of renewable energy sources in fog computing environment via frequency and modulation level scaling. *IEEE Internet of Things Journal*, 7(11):10912–10921, 2020.

[50] Minxian Xu, Adel N. Toosi, Behrooz Bahrani, Reza Razzaghi, and Martin Singh. Optimized renewable energy use in green cloud data centers. In *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*, pages 314–330. Springer, 2019.

[51] Borja Martinez and Xavier Vilajosana. Exploiting the solar energy surplus for edge computing. *IEEE Transactions on Sustainable Computing*, 7(1):135–143, 2021.

[52] Dominic Dudley. Renewable energy costs take another tumble, making fossil fuels look more expensive than ever. https://www.forbes.com/sites/dominicdudley/2019/05/29/renewable-energy-costs-tumble/?sh=7ebb30b1e8ce. Accessed on 27th April 2023.

[53] Daniel Oberhaus. Amazon, google, microsoft: Here's who has the greenest cloud. https://www.wired.com/story/amazon-google-microsoft-green-clouds-and-hyperscale-data -centers/. Accessed on 19th May 2023.

[54] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 22. ACM, 2011.

[55] Changbing Chen, Bingsheng He, and Xueyan Tang. Green-aware workload scheduling in geographically distributed data centers. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 82–89. IEEE, 2012.

[56] Sanjaya K Panda and Prasanta K Jana. Uncertainty-based qos min–min algorithm for heterogeneous multi-cloud environment. *Arabian Journal for Science and Engineering*, 41(8):3003–3025, 2016.

[57] HM Dipu Kabir, Abbas Khosravi, Subrota K Mondal, Mustaneer Rahman, Saeid Nahavandi, and Rajkumar Buyya. Uncertainty-aware decisions in cloud computing: Foundations and future directions. *ACM Computing Surveys (CSUR)*, 54(4):1–30, 2021.

[58] Mengmeng Zhao, Xiaoying Wang, and Junrong Mo. Workload and energy management of geo-distributed datacenters considering demand response programs. *Sustainable Energy Technologies and Assessments*, 55:102851, 2023.

[59] Mirsaeid Hosseini Shirvani. An energy-efficient topology-aware virtual machine placement in cloud datacenters: A multi-objective discrete jaya optimization. *Sustainable Computing: Informatics and Systems*, 38:100856, 2023.

[60] Mary Zhang. Top 250 data center companies in the world as of 2023. https://dgtlinfra.com/top-data-center-companies/. Accessed on 27th April 2023.

[61] Benjamin K Sovacool, Chukwuka G Monyei, and Paul Upham. Making the internet globally sustainable: Technical and policy options for improved energy management, governance and community acceptance of nordic datacenters. *Renewable and Sustainable Energy Reviews*, 154:111793, 2022.

[62] Consumer Energy Solutions. What industries consume the most electricity? https://consumerenergysolutions.com/what-industries-consume-most-electricity/. Accessed on 27th April 2023.

[63] Sanjib Nayak, Sanjaya Panda, Satyabrata Das, and Sohan Pande. A multi-objective renewable energy-based algorithm for geographically distributed datacentres. *International Journal of Embedded Systems*, 15(2):119–131, 2022.

[64] Sanjib Kumar Nayak, Sanjaya Kumar Panda, and Satyabrata Das. Constrained-based power management algorithm for green cloud computing. *International Journal of Computational Science and Engineering*, 25(6):657–667, 2022.

[65] World Economic Forum. Iea: More than a third of the world's electricity will come from renewables in 2025. https://www.weforum.org/agenda/2023/03/electricity-generation-renewables-power-iea/. Accessed on 27th April 2023.

[66] Kaifeng Yu and Paul van Son. Review of trans-mediterranean power grid interconnection: A regional roadmap towards energy sector decarbonization. *Global Energy Interconnection*, 6(1):115–126, 2023.

[67] Chenhan Xu, Kun Wang, Peng Li, Rui Xia, Song Guo, and Minyi Guo. Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning. *IEEE Transactions on Network Science and Engineering*, 2018.

[68] S Nayak, S Panda, S Das, and S Pande. An efficient renewable energy-based scheduling algorithm for cloud computing. In *International Conference on Distributed Computing and Internet Technology*, pages 1–17. Springer, 2021.

[69] Sanjib Nayak, Sanjaya Panda, and Satyabrata Das. Unconstrained power management algorithm for green cloud computing. In *Advances in Distributed Computing and Machine Learning*, pages 1–10. Springer, 2021.

[70] Slokashree Padhi and RBV Subramanyam. Efficient renewable energy-based geographical load balancing algorithms for green cloud computing. *International Journal of Web and Grid Services*, 19(4):401–426, 2023.

[71] Slokashree Padhi and RBV Subramanyam. Uncertainty level-based algorithms by managing renewable energy for geo-distributed datacenters. *Cluster Computing*, pages 1–18, 2024.

[72] Sambit Kumar Mishra, Deepak Puthal, Bibhudatta Sahoo, Suraj Sharma, Zhi Xue, and Albert Y Zomaya. Energy-efficient deployment of edge dataenters for mobile clouds in sustainable iot. *Ieee Access*, 6:56587–56597, 2018.

[73] Haifeng Gu, Zishuai Ge, E Cao, Mingsong Chen, Tongquan Wei, Xin Fu, and Shiyan Hu. A collaborative and sustainable edge-cloud architecture for object tracking with convolutional siamese networks. *IEEE Transactions on Sustainable Computing*, 6(1):144–154, 2019.

[74] Xiaonan Wang and Yimin Lu. Sustainable and efficient fog-assisted iot cloud based data collection and delivery for smart cities. *IEEE Transactions on Sustainable Computing*, 7(4):950–957, 2022.

[75] Dan C Marinescu. Cloud service providers and the cloud ecosystem. *Cloud Computing*, pages 13–49, 2018.

[76] Amazon Web Service. Aws global infrastructure. https://aws.amazon.com/about-aws/global-infrastructure/. Accessed on 27th April 2023.

[77] Arva Arsiwala, Faris Elghaish, and Mohammed Zoher. Digital twin with machine learning for predictive monitoring of co2 equivalent from existing buildings. *Energy and Buildings*, 284:112851, 2023.

[78] Melissa Gregg and Yolande Strengers. Getting beyond net zero dashboards in the information technology sector. *Energy Research & Social Science*, 108:103397, 2024.

[79] Minxian Xu and Rajkumar Buyya. Managing renewable energy and carbon footprint in multi-cloud computing environments. *Journal of Parallel and Distributed Computing*, 135:191–202, 2020.

[80] MA Majid et al. Renewable energy for sustainable development in india: current status, future prospects, challenges, employment, and investment opportunities. *Energy, Sustainability and Society*, 10(1):1–36, 2020.

[81] Georgios Methenitis, Michael Kaisers, and Han La Poutré. Renewable electricity trading through slas. *Energy Informatics*, 1(1):1–17, 2018.

[82] Iqbal Ahmed. *Sustainable Green Service Level Agreement (GSLA) Framework Development for IT and ICT Based Industries.* PhD thesis, Saga University, Saga, Japan, 2018.

[83] Demetris Koutsoyiannis. The unavoidable uncertainty of renewable energy and its management. In *Proceedings of the EGU General Assembly Conference, Vienna, Austria*, pages 17–22, 2016.

[84] Patricia Stefan de Carvalho, Julio Cezar Mairesse Siluk, Jones Luis Schaefer, et al. Analysis of factors that interfere with the regulatory energy process with emphasis on the energy cloud. *International Journal of Energy Economics and Policy*, 12(2):325–335, 2022.

[85] Sanjaya Kumar Panda, Sahil Chopra, and Slokashree Padhi. Solar energy-based virtual machine placement algorithm for geo-distributed datacenters. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pages 481–490. IEEE, 2023.

[86] Sanjib Nayak, Sanjaya Panda, Satyabrata Das, and Sohan Pande. A renewable energy-based task consolidation algorithm for cloud computing. In *Control Applications in Modern Power System*, pages 453–463. Springer, 2020.

[87] Manas Kumar Mishra and Sanjaya Kumar Panda. A cost-variant renewable energy-based scheduling algorithm for cloud computing. In *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing*, pages 91–97, 2022.

[88] Tuhin Chakraborty, Adel N Toosi, and Carlo Kopp. Elastic power utilization in sustainable micro cloud data centers. *IEEE Transactions on Sustainable Computing*, 2023.

[89] Rym Regaieg, Mohamed Koubàa, Zacharie Ales, and Taoufik Aguili. Multi-objective optimization for vm placement in homogeneous and heterogeneous cloud service provider data centers. *Computing*, 103:1255–1279, 2021.

[90] Adel Nadjaran Toosi, Chenhao Qu, Marcos Dias de Assunção, and Rajkumar Buyya. Renewable-aware geographical load balancing of web applications for sustainable data centers. *Journal of Network and Computer Applications*, 83:155–168, 2017.

[91] Yu Luo, Lina Pu, and Chun-Hung Liu. Cpu frequency scaling optimization in sustainable edge computing. *IEEE Transactions on Sustainable Computing*, 2022.

[92] Jianwen Xu, Kaoru Ota, Mianxiong Dong, and Ai-Chun Pang. Efficiency-aware dynamic service pricing strategy for geo-distributed fog computing. *IEEE Transactions on Sustainable Computing*, 7(4):814–824, 2022.

[93] Mohamed Abd Elaziz, Ibrahim Attiya, Laith Abualigah, Muddesar Iqbal, Amjad Ali, Ala Al-Fuqaha, and Shaker El-Sappagh. Hybrid enhanced optimization-based intelligent task scheduling for sustainable edge computing. *IEEE Transactions on Consumer Electronics*, 2023.

[94] Zheng Li, Selome Tesfatsion, Saeed Bastani, Ahmed Ali-Eldin, Erik Elmroth, Maria Kihl, and Rajiv Ranjan. A survey on modeling energy consumption of cloud applications: deconstruction, state of the art, and trade-off debates. *IEEE Transactions on Sustainable Computing*, 2(3):255–274, 2017.

[95] Sanjaya Kumar Panda, Sohan Kumar Pande, and Satyabrata Das. Task partitioning scheduling algorithms for heterogeneous multi-cloud environment. *Arabian Journal for Science and Engineering*, 43(2):913–933, 2018.

[96] Sanjib Kumar Nayak, Sanjaya Kumar Panda, and Satyabrata Das. Renewable energy-based resource management in cloud computing: a review. *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2020*, pages 45–56, 2021.

[97] S Nithya, M Sangeetha, KN Apinaya Prethi, Kshira Sagar Sahoo, Sanjaya Kumar Panda, and Amir H Gandomi. Sdcf: A software-defined cyber foraging framework for cloudlet environment. *IEEE Transactions on Network and Service Management*, 17(4):2423–2435, 2020.

[98] Sohan Kumar Pande, Sanjaya Kumar Panda, Satyabrata Das, Mamoun Alazab, Kshira Sagar Sahoo, Ashish Kumar Luhach, and Anand Nayyar. A smart cloud service management algorithm for vehicular clouds. *IEEE Transactions on Intelligent Transportation Systems*, 22(8):5329–5340, 2020.

[99] Sohan Kumar Pande, Sanjaya Kumar Panda, Satyabrata Das, Kshira Sagar Sahoo, Ashish Kr Luhach, NZ Jhanjhi, Roobaea Alroobaea, and Sivakumar Sivanesan. A resource management algorithm for virtual machine migration in vehicular cloud computing. *Computers, Materials & Continua*, 67(2), 2021.

[100] Sanjib Kumar Nayak, Sanjaya Kumar Panda, Satyabrata Das, and Sohan Kumar Pande. A renewable energy-based task consolidation algorithm for cloud computing. In *Control Applications in Modern Power System: Select Proceedings of EPREC 2020*, pages 453–463. Springer, 2021.

[101] Ahmed Hadi Ali AL-Jumaili, Ravie Chandren Muniyandi, Mohammad Kamrul Hasan, Mandeep Jit Singh, Johnny Koh Siaw Paw, and Mohammad Amir. Advancements in intelligent cloud computing for power optimization and battery management in hybrid renewable energy systems: A comprehensive review. *Energy Reports*, 10:2206–2227, 2023.

[102] Arul Rajagopalan, Dhivya Swaminathan, Mohit Bajaj, Issam Damaj, Rajkumar Singh Rathore, Arvind R Singh, Vojtech Blazek, and Lukas Prokop. Empowering power distribution: Unleashing the synergy of iot and cloud computing for sustainable and efficient energy systems. *Results in Engineering*, page 101949, 2024.

[103] Mahmood ul Hassan, Amin A Al-Awady, Abid Ali, Muhammad Munawar Iqbal, Muhammad Akram, Jahangir Khan, and Ali Ahmad AbuOdeh. An efficient dynamic decision-based task optimization and scheduling approach for microservice-based cost management in mobile cloud computing applications. *Pervasive and Mobile Computing*, 92:101785, 2023.

[104] Paulius Tervydis, Linas Svilainis, Žilvinas Nakutis, and Alberto Rodríguez-Martínez. Cloudedgeassetoptimizer: Tool to optimize the cloud-edge computing network resources at given requirements of processing delay, battery capacity and cost. *SoftwareX*, 26:101714, 2024.

[105] Ya-Jun Leng and Huan Zhang. A novel evaluation method of renewable energy development based on improved rough set theory and grey cloud model. *Journal of Cleaner Production*, 428:139299, 2023.

[106] Shi-Fan He, Ying-Ming Wang, and Luis Martínez. Gaussian it2fss-based prospect theory method with application to the evaluation of renewable energy sources. *Computers & Industrial Engineering*, 169:108266, 2022.

[107] Daming Zhao and Jiantao Zhou. An energy and carbon-aware algorithm for renewable energy usage maximization in distributed cloud data centers. *Journal of Parallel and Distributed Computing*, 165:156–166, 2022.

[108] Dahye Jeong, Syjung Hwang, Jisu Kim, Hyerim Yu, and Eunil Park. Public perspective on renewable and other energy resources: Evidence from social media big data and sentiment analysis. *Energy Strategy Reviews*, 50:101243, 2023.

[109] Sanjaya K Panda and Prasanta K Jana. An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems. *Cluster Computing*, 22(2):509–527, 2019.

[110] Md Sabbir Hasan, Yousri Kouki, Thomas Ledoux, and Jean-Louis Pazat. Exploiting renewable sources: When green sla becomes a possible reality in cloud computing. *IEEE Transactions on Cloud Computing*, 5(2):249–262, 2015.

[111] P Pabitha, K Nivitha, C Gunavathi, and B Panjavarnam. A chameleon and remora search optimization algorithm for handling task scheduling uncertainty problem in

cloud computing. *Sustainable Computing: Informatics and Systems*, 41:100944, 2024.

[112] V Kumar Saini, Rajesh Kumar, Ameena Saad Al-Sumaiti, and BK Panigrahi. Uncertainty aware optimal battery sizing for cloud energy storage in community microgrid. *Electric Power Systems Research*, 222:109482, 2023.

[113] Vikash Kumar Saini, Ameena S Al-Sumaiti, and Rajesh Kumar. Data driven net load uncertainty quantification for cloud energy storage management in residential microgrid. *Electric Power Systems Research*, 226:109920, 2024.

[114] Andrei Tchernykh, Uwe Schwiegelsohn, El-ghazali Talbi, and Mikhail Babenko. Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability. *Journal of Computational Science*, 36:100581, 2019.

[115] Slokashree Padhi and RBV Subramanyam. User request-based scheduling algorithms by managing uncertainty of renewable energy. *Cluster Computing*, pages 1–18, 2023.

[116] T Rajeev and S Ashok. Dynamic load-shifting program based on a cloud computing framework to support the integration of renewable energy sources. *Applied Energy*, 146:141–149, 2015.

[117] Marcos De Melo da Silva, Abdoulaye Gamatié, Gilles Sassatelli, Michael Poss, and Michel Robert. Optimization of data and energy migrations in mini data centers for carbon-neutral computing. *IEEE Transactions on Sustainable Computing*, 8(1):68–81, 2022.

[118] Jordi Guitart. Practicable live container migrations in high performance computing clouds: Diskless, iterative, and connection-persistent. *Journal of Systems Architecture*, page 103157, 2024.

[119] Doraid Seddiki, Sebastián García Galán, J Enrique Muñoz Expósito, Manuel Valverde Ibañez, Tomasz Marciniak, J Rocío, and Pérez de Prado. Sustainable expert virtual machine migration in dynamic clouds. *Computers and Electrical Engineering*, 102:108257, 2022.

[120] Ting Yang, Han Jiang, Yucheng Hou, and Yinan Geng. Carbon management of multi-datacenter based on spatio-temporal task migration. *IEEE Transactions on Cloud Computing*, 11(1):1078–1090, 2021.

[121] Mohammed Anis Benblidia, Bouziane Brik, Moez Esseghir, and Leila Merghem-Boulahia. A renewable energy-aware power allocation for cloud data centers: A game theory approach. *Computer Communications*, 179:102–111, 2021.

[122] Minxian Xu, Adel Nadjaran Toosi, and Rajkumar Buyya. A self-adaptive approach for managing applications and harnessing renewable energy for sustainable cloud computing. *IEEE Transactions on Sustainable Computing*, 6(4):544–558, 2020.

[123] Léo Grange, Georges Da Costa, and Patricia Stolf. Green it scheduling for data center powered with renewable energy. *Future Generation Computer Systems*, 86:99–120, 2018.

[124] Subhendu Nayak. Regions and availability zones: Aws vs azure vs gcp. https://www.cloudoptimo.com/blog/regions-and-availability-zones-aws-vs-azure-vs-gcp/. Accessed on 27th Mar 2023.

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

[126] Measurement and instrumentation data center (midc). https://midcdmz.nrel.gov/. Accessed: 2024-08-20.

[127] Wholesale electricity and natural gas market data. https://www.eia.gov/electricity/wholesale/. Accessed: 2024-08-20.

List of Publications

List of International Journals:

1. **Slokashree Padhi** and R. B. V. Subramanyam, "User Request-Based Scheduling Algorithms by Managing Uncertainty of Renewable Energy", *Cluster Computing*, Springer, Vol. 27, pp. 1965-1982, 2024. (**Indexed in SCI/SCIE, Impact Factor: 4.4, Published**)
   DOI: https://doi.org/10.1007/s10586-023-04057-z

2. **Slokashree Padhi** and R. B. V. Subramanyam, "Efficient Renewable Energy-Based Geographical Load Balancing Algorithms for Green Cloud Computing", *International Journal of Web and Grid Services*, Inderscience, Vol. 19, No. 4, pp. 401-426, 2023. (**Indexed in SCI/SCIE, Impact Factor: 3.8, Published**)
   DOI: https://doi.org/10.1504/IJWGS.2023.135576

3. **Slokashree Padhi** and R. B. V. Subramanyam, "Uncertainty Level-Based Algorithms by Managing Renewable Energy for Geo-Distributed Datacenters", *Cluster Computing*, Springer, 2024. (**Indexed in SCI/SCIE, Impact Factor: 4.4, Published**)
   DOI: https://doi.org/10.1007/s10586-023-04216-2

4. **Slokashree Padhi** and R. B. V. Subramanyam, "A Renewable Energy-Oriented Migration Algorithm for Minimizing Cost in Geo-Distributed Cloud Datacenters", *The Journal of Supercomputing*, Springer, 2024. (**Indexed in SCI/SCIE, Impact Factor: 3.3, Submitted**)