Performance Analysis of Real-time Task Scheduling in Cloud System

Sampa Sahoo

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Performance Analysis of Real-time Task Scheduling in Cloud System

Dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Sampa Sahoo

(Roll Number: 514CS1013)

based on research carried out

under the supervision of

Dr. Bibhudatta Sahoo

and

Dr. Ashok Kumar Turuk

September, 2020

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Department of Computer Science and Engineering
National Institute of Technology Rourkela

September, 2020

Certificate of Examination

Roll Number: *514CS1013*
Name: *Sampa Sahoo*
Title of Dissertation: *Performance Analysis of Real-time Task Scheduling in Cloud System*

We the below signed, after checking the dissertation mentioned above and the official record book (s) of the student, hereby state our approval of the dissertation submitted in partial fulfillment of the requirements of the degree of *Doctor of Philosophy* in *Computer Science and Engineering* at *National Institute of Technology Rourkela*. We are satisfied with the volume, quality, correctness, and originality of the work.

Ashok Kumar Turuk Co-Supervisor	Bibhudatta Sahoo Principal Supervisor
Pabitra Mohan Khilar Member, DSC	Siba Sankar Mohapatra Member, DSC
Debiprasad Priyabrata Acharya Member, DSC	Deo Prakash Vidyarthi External Examiner
Santanu Kumar Rath Chairperson, DSC	Ashok Kumar Turuk Head of the Department

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Dr. Bibhudatta Sahoo

Dr. Ashok Kumar Turuk

September, 2020

Supervisors' Certificate

This is to certify that the work presented in the dissertation entitled *Performance Analysis of Real-time Task Scheduling in Cloud System* submitted by *Sampa Sahoo*, Roll Number 514CS1013, is a record of original research carried out by her under our supervision and guidance in partial fulfillment of the requirements of the degree of *Doctor of Philosophy* in *Computer Science and Engineering*. Neither this dissertation nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.

Dr. Ashok Kumar Turuk

Dr. Bibhudatta Sahoo

Dedication

This dissertation is dedicated to my family members, and friends, who had encouraged me to pursue my dreams and finish my dissertation.

Signature

Declaration of Originality

I, *Sampa Sahoo*, Roll Number *514CS1013* hereby declare that this dissertation entitled *Performance Analysis of Real-time Task Scheduling in Cloud System* presents my original work carried out as a doctoral student of NIT Rourkela and, to the best of my knowledge, contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma of NIT Rourkela or any other institution. Any contribution made to this research by others, with whom I have worked at NIT Rourkela or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections "Reference" or "Bibliography". I have also submitted my original research records to the scrutiny committee for evaluation of my dissertation.

I am fully aware that in case of any non-compliance detected in future, the Senate of NIT Rourkela may withdraw the degree awarded to me on the basis of the present dissertation.

September, 2020
NIT Rourkela

Sampa Sahoo

Acknowledgment

Ph.D. is a once-in-a-lifetime opportunity and experience. It is tough at times and may seem like an eternity, but it experiences us a lot, and I am delighted that I have had a chance to complete it. It would not have happened without all those people who helped me along the way. First of all, I would like to thank my supervisor, Dr. Bibhudatta Sahoo, and co-supervisor Dr. Ashok Kumar Turuk, who has allowed me to undertake a Ph.D. and provided with invaluable guidance and advice throughout my Ph.D. candidature. I want to express my gratitude to the Ph.D. scrutiny committee members, Dr. S.K. Rath, Dr. P.M. Khillar, Dr. S.S. Mohapatra, and Dr. D.P. Acharya for their constructive comments and suggestions on improving my work. My sincere regard to all my teachers who taught me in the Department of CSE, National Institute of Technology Rourkela.

I would also like to thank all the past and current members of the Cloud Computing Research Laboratory, NIT Rourkela. In particular, I thank Sambit Kumar Mishra, Kshira Sagar Sahoo, Shreeya Swagatika Sahoo, and Dibya Sundar Das for their friendship and help during my Ph.D. I thank the external examiners for their excellent reviews and suggestions on improving this thesis.

I am heartily thankful to my parents for their support and encouragement at all times. Eventually, I thank all my friends and well-wishers who have directly or indirectly helped me in this journey of doctoral research.

NIT Rourkela

Sampa Sahoo
Roll Number: 514CS1013

Abstract

Cloud computing is becoming an important computing paradigm due to its cost-efficiency, scalability, availability, and high resource utilization. Applications like financial transactions, health-care, scientific workflows, video streaming, Internet of Things (IoT), etc. with their real-time nature, need provisioning of cloud resources to guarantee timeliness and high availability. The Cloud Service Providers (CSPs) must support sufficient cloud resources to satisfy the demand from these real-time applications. Meanwhile, the ever-growing demand from applications forces CSPs to deploy more and more cloud resources, which consumes a considerable amount of energy. The high energy consumption affects the environment and other metrics like execution cost, makespan, and reliability of the cloud system. Hence, it necessitates employing some techniques to reduce cloud systems' energy consumption and make it energy-efficient along with other performance metrics like reliability, execution cost, makespan, etc. Real-time task scheduling is one of the methods to achieve energy-efficiency in the cloud system. Moreover, heterogeneous computing environments and application timing constraints add complexity to the real-time task scheduling. Therefore, the study of a cloud system's performance is necessary for real-time applications to ensure Quality of Service (QoS), defined in terms of energy consumption, makespan, execution cost, reliability, etc.

First, an Energy and Cost Aware task scheduling (ECA) algorithm based on the TOPSIS analysis method is proposed to reduce energy consumption and execution cost. Here, a scoring value is calculated for a VM based on its energy consumption and execution cost to execute a task. Then, a VM with the best score in terms of energy usage and execution cost is selected.

Next, a Learning Automata (LA)-based scheduling (LAS) algorithm is proposed to minimize energy consumption and makespan. It is a reinforcement-based method where the action, i.e., assignment of a task to a VM, is penalized if it contributes to scheduling objective degradation and rewarded if the action is suitable to improve scheduling objective. The above process is continued for a fixed number of iterations, and the actions with the best reward value are added in the scheduling decision.

Then, a game theory based scheduling algorithm is proposed to enhance system performance where energy consumption and reliability are considered as the performance metrics. The bi-objective scheduling algorithm is modeled as a non-cooperative scheduling game, named Real-time Task Scheduling Game (RTSG). The solution or Nash Equilibrium

of RTSG is presented using a Vickery auction mechanism. The proposed solution is compared with a cooperative game based solution and an auction-based approach.

Finally, a fault-tolerant scheduling algorithm is presented, taking into account energy consumption and reliability. First, an acceptance test mechanism is designed considering schedulability and response time failure to detect VM failure. A reliability and energy-aware fault-tolerant scheduling algorithm, REO, is proposed using the PB concept and BB overlapping technique. The performance metrics used for comparison of algorithms include Success Ratio, makespan, and total energy consumption. The outcomes of the simulation results signify the usefulness and effectiveness of the proposed algorithms for studying real-time task scheduling performance.

***Keywords*: *Cloud Computing*; *Energy*; *Execution cost*; *Makespan*; *Real-time Task*.**

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbreviation	Meaning
AWS	Amazon Web Service
ACO	Ant Colony Optimization
AEAP	As Early As Possible
ALAP	As Late As Possible
BB	Backup-Backup
B	Byte
CSP	Cloud Service Provider
CTMC	Continuous Time Markov Chain
DaaS	Data-as-a-Service
DES	Discrete Event Simulation
DVFS	Dynamic Voltage Frequency Scaling
ETC	Expected Time to Compute
GA	Genetic Algorithm
I	Number of Instructions
INF	Infinity
ICT	Information Communications Technology
IaaS	Infrastructure-as-a-Service
IDS	Intrusion Detection System
LA	Learning Automata
MTBF	Mean-Time-Between-Failure
MB	Mega Byte
MCT	Minimum Completion Time
MET	Minimum Execution Time
MI	Million Instruction
MIPS	Million Instructions Per Second
MCDM	Multi-Criteria Decision Making
MODM	Multi-Objective Decision Making
MOGA	Multi-Objective Genetic Algorithm

Abbreviation	Meaning
NE	Nash Equilibrium
PSO	Particle Swarm Optimization
PM	Physical Machine
PaaS	Platform-as-a-Service
PB	Primary-Backup
QoE	Quality of Experience
QoS	Quality of Service
SLA	Service Level Agreement
SAW	Simple Additive Weighting
SaaS	Software-as-a-Service
TOPSIS	Technique for Order Preference by Similarity to the Ideal solution
VM	Virtual Machine

List of Symbols

Symbol	Description
p_i	Action probability vector corresponding to α_i
α_i	Action set of A_i
δ	Aggregate energy consumption
λ	Arrival rate of task
tk_i^{ar}	Arrival time of i^{th} task
B_tk_i	Backup copy of tk_i
b_i^j	Bid value for tk_i by vm_j
d_i^j	Binary variable indicating whether tk_i met deadline on vm_j or not
tk_i^{dl}	Deadline of i^{th} task
P_est_i	Earliest start time of P_tk_i
P_eft_i	Earliest finish time of P_tk_i
ξ_j^a	Energy consumed by vm_j in active state
ξ_j^i	Energy consumed by vm_j in idle state
et_i^j	Expected execution time of tk_i on VM vm_j
$P\_et_i^j$	Expected execution time of P_tk_i on vm_j
$B\_et_i^j$	Expected execution time of B_tk_i on vm_j
ec_i^j	Execution cost of tk_i on vm_j
vm_j^{ec}	Execution cost of j^{th} VM per time unit
vm_j^{fr}	Failure rate of j^{th} VM
ft_i^j	Finish time of tk_i on vm_j
$P\_ft_i^j$	Finish time of P_tk_i on vm_j
$b\_ft_i^j$	Finish time of B_tk_i on vm_j
tk_i	i^{th} task
vm_{id}^+	Ideal VM
vm_j	j^{th} VM
pl_j	j^{th} player

Symbol	Description
B_lst_i	Latest start time of B_tk_i
B_lft_i	Latest finish time of B_tk_i
A_i	Learning automata associated with tk_i
$\mathcal{L}$	Learning or Reinforcement algorithm
τ	Makespan
vm_{id}^{-}	Negative ideal VM
vm_j^{rel}	Overall reliability of vm_j
vm_p^{score}	Overall score of a VM
φ	Penalty constant
P_tk_i	Primary copy of tk_i
vm_j^{rel}	Reliability of j^{th} VM
rel_i^j	Reliability of tk_i on vm_j
ϕ	Reward constant
R_i^j	Reward for executing tk_i on vm_j
tk_i^{sz}	Size of i^{th} task
vm_j^{sp}	Speed of j^{th} VM
st_i^j	Start time of tk_i on vm_j
$P\_st_i^j$	Start time of P_tk_i on vm_j
$B\_st_i^j$	Start time of B_tk_i on vm_j
s_i^j	Strategy of player pl_j
ψ_j	Task allocation strategy space for player vm_j
$\mathcal{T}$	Task set
ϱ_j	Total energy consumption by vm_j
ec	Total execution cost
μ_j	Total execution time
u_i^j	Utility value for executing tk_i on vm_j
$\mathcal{V}$	VM set

Chapter 1

Introduction

1.1 Overview

Cloud computing refers to delivering the application, hardware, and system software stored in data centers as a service in a pay-per-use pricing basis over the Internet. Virtualization is the key technology behind the cloud that creates an illusion of infinite computing resources. Cloud computing can also be viewed as a large scale distributed computing paradigm driven by the economy of scale and provides an abstracted, virtualized, scalable, managed computing power, storage, and platforms as a service to users on-demand over the Internet. Further, cloud-based services like Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) intend to extend support for a wide range of applications. SaaS is a software delivery process that allows access to software *via* a subscription model, whereas, in PaaS, software, and hardware needed for application development is delivered through the Internet. In IaaS, computational resources are provided to users in the form of a lease. When viewed from an organizational perspective, cloud services can be deployed in the following ways:

- *Public cloud* allows cloud services offered by the Cloud Service Provider (CSP) available to anyone who wants to use or procure them.
- In *Private cloud,* cloud services or computing resources are provisioned over private IT infrastructure for the dedicated use by a particular organization.
- *Hybrid cloud* is designed, taking into account the benefits of both private and public clouds.

Cloud computing advantages lead to hosting real-time applications like healthcare systems, video streaming, financial transaction system, IoT, etc. These applications need a timely (within a deadline) response to a request and computational correctness. As these applications need a timely response, depending on their nature (whether emergent or not), the cloud system decides whether to expand or shrink the count on cloud resources. For instance, a task of a healthcare application demands guarantees of timeliness strictly, whereas applications like video streaming can withstand some relaxation in time constraints. To

satisfy the need for time-constrained applications, a CSP must provide a sufficient number of cloud resources. Meanwhile, the evergrowing demand of applications forces a CSP to deploy more and more cloud resources. This increasing number of cloud resources in a cloud data center consumes a huge amount of energy [1, 2]. Following facts attribute to the reasons for high energy consumption in cloud data centers:

- Some computing resources are inevitably idle during different time slots, which indeed lowers resource utilization and raises idle energy consumption. In [1–3] low utilization of computing resources is affirmed as one of the crucial elements contributing to high energy consumption. Moreover, a study presented in [1, 4] shows that average resource utilization in a cloud system is not more than 30%. Still, the energy used by idle resources is at least $60 - 70\%$ of peak energy.

- Sometimes, cloud resources are over-provisioned to meet the worst-case user demand. The significantly higher number of resources reservation, in turn, increases energy consumption. Besides, inefficient and improper resource scheduling leads to the selection of Virtual Machines (VMs) that will cause high energy consumption while ensuring Quality of Service (QoS) demand (e.g., deadline) of the applications.

There is high energy consumption for increased resource utilization, whereas the absence of a cost factor in scheduling decisions may lead to high operational costs. However, reducing energy consumption increases the makespan and leads to user dissatisfaction. Besides, the reliability of the system is affected by high energy consumption [1, 5]. An unreliable cloud system with a higher probability of resource failures inevitably results in more interruption of running VMs, which implies a reduction in the performance of cloud services [6]. In this context, a fault-tolerant strategy helps guarantee the reliability of the cloud system [5, 7].

1.2 Motivation of the Work

A report presented in [2, 8] states that data centers consume 1.5% of global electric energy in the year 2010, which will be doubled by 2020 if current trends continue. Another study reported in [9] states that "the computing resources in cloud consumes 55% energy while the remaining energy is consumed by other support systems such as cooling, power supply, etc.". Further, high energy consumption not only affects the environment but also lowers the system reliability [1], and increases operating costs [10]. According to [1] the Arrhenius life-stress model, "for every 10^oC increase in temperature, the failure rate of electronic devices rises by a factor of two," hence affecting the reliability of the system. Efficient fault-tolerance mechanisms can enhance the reliability of the cloud. Despite several advantages, the cloud resources incur a high failure probability due to the increased functionality and complexity of a large system. Further, the use of inexpensive commodity

hardware adds to the cause of resource failure. Faults in the cloud are inevitable, and the report presented in [11, 12] states that "8% of VMs encounter errors at run time". A study reported in [13] says that "cloud consisting of servers with Mean-Time-Between-Failure (MTBF) of 30 years fails at least once in a day." Hence, the cloud needs to provide a computing resource with high fault-tolerant capability, which enhances the reliability of the cloud system. The study presented in [10] says that "about 50% management budget of Amazon's data center is used for powering and cooling physical servers". The execution cost is also an indispensable contributor to the operating cost of the system. Therefore, the execution cost must be considered while generating a solution for improved system performance. Moreover, there is a trade-off between energy consumed and execution time, as a faster execution implies a higher energy consumption [9]. Hence, it is hugely requisite to employ some means to lessen the cloud system's energy consumption and make it energy-efficient along with other performance metrics like reliability, execution cost, makespan, etc.

1.3 Objective of the Work

There are several ways to reduce energy consumption in the cloud system: use of low power processor architectures or Dynamic Voltage Frequency Scaling (DVFS) [14, 15], VM consolidation, and energy-efficient task scheduling policies, etc. This thesis addresses the scheduling of real-time tasks to minimize energy consumption in the cloud system. Real-time task scheduling problems can be defined as follows: for a given set of VMs and tasks, obtain the best mapping of a task to VM such that deadline constraint is met. Further, this mapping should optimize system performance metrics, such as energy consumption, makespan, etc. Task scheduling is a widely used method by researchers to minimize the cloud system's energy consumption but is more challenging when it becomes multi-objective. For instance, to reduce the energy consumption and execution cost simultaneously, the scheduler must make the best possible decision; otherwise, reducing energy consumption may lead to a rise in execution cost or vice-versa. Furthermore, heterogeneous computing environments and deadline constraints of applications add to the complexity of scheduling. In this thesis, scheduling algorithms are proposed to solve the multi-objective real-time task scheduling problem in a heterogeneous cloud system. The objective of the thesis are enumerated below:

- Proposed a VM scoring based scheduling algorithm to minimize energy consumption and execution cost.
- Proposed a reinforcement based scheduling algorithm to reduce energy consumption and makespan.

- Addressed a bi-objective scheduling problem where optimization of energy consumption and reliability are two objectives.
- Proposed a fault-tolerant scheduling algorithm taking into account energy consumption and reliability.

Different methodologies used to achieve the objectives are discussed in the next section.

1.4 Methodology Used

Various methods used in this thesis to solve the multi-objective real-time task scheduling problem are discussed below.

1.4.1 Queuing Theory

Queuing theory is widely used to model and study the performance and QoS of various Information Communications Technology (ICT) systems [16, 17]. The mathematical study of waiting in line examines the arrival process, service process, number of servers, etc. The use of probabilistic distributions like Poisson and Exponential allows modeling complex phenomena of waiting in line as a simple mathematical equation that can be used to analyze a system's behavior. The efficient use of queuing theory leads to better staffing solutions, reduces customer waiting time, and finds its applications in scheduling, customer service, etc.

1.4.2 Multi Objective Decision Making Method

Multi-Objective Decision Making (MODM) or Multi-Criteria Decision Making (MCDM) refers to making decisions in the presence of multiple, usually conflicting objectives or criteria [18, 19]. It can also be defined as the method of choosing the best alternative from a set of decision alternatives. The necessary steps of MODM are: (i) list system evaluation criteria and generate alternatives based on these criteria, (ii) apply one of the multiple criteria analysis method, (iii) one of the alternatives is referred to as optimal (solution). Among many MODM techniques, Max-Min, Max-Max, Simple Additive Weighting (SAW), Technique for Order Preference by Similarity to the Ideal Solution (TOPSIS) are the most frequently used methods [19]. In the Max-Min and Max-Max technique, alternatives are selected by its weakest attribute and best attribute, respectively. SAW method uses the product of the normalized value of criteria and weight (importance) of the criteria to select the alternative with the highest criteria as the preferred one. In the TOPSIS method, an alternative is preferable, which is closest to the ideal solution and farthest from the negative ideal solution. The TOPSIS method is simple and maintains the same number of steps, regardless of problem size. Thus makes it an efficient decision making tool. Hence the TOPSIS analysis

method is appropriate for the real-time task scheduling problem that involves finding the best $\langle task, VM \rangle$ pair.

1.4.3 Reinforcement Learning

Reinforcement learning is a technique where an agent learns behavior through trial and error interactions with a dynamic environment. Based on how the learning process is implemented, reinforcement learning can be classified as follows: probability vector algorithm, associative algorithm, and learning with delayed reinforcement [20]. The probability vector algorithm uses action probability and reinforcement signals from the environment to select an optimal action for finding a solution, e.g., Learning Automata (LA). In LA, a process starts with a stochastic policy that selects actions randomly in the beginning. The probability of choosing an action is updated based on the environment's feedback until optimal action is found. The associative algorithm takes both reinforcement signal value and internal state to find a solution, e.g., neural net architecture. In learning with delayed reinforcement, an agent receives an evaluation of its behavior following the entire sequence of steps and is solved using dynamic programming variations, e.g., Q-learning. Agents in Q-learning choose action either by higher Q-value (exploitation) or by action randomly (exploration). Hence, reinforcement learning-based methods can be employed in real-time task scheduling to generate optimal task attribution policies. Further, learning methods help to get long-term system's performance improvement as an agent adapts itself with environmental conditions and unsteady requests.

1.4.4 Game Theory

Game theory can be viewed as a tool to analyze the introduction among decision-makers with conflicting objectives [21]. A game can be cooperative where players cooperate to reach the goal or non-cooperative where players work independently without the information about other players' strategies. Auctions are a type of game where bidders strategically select the best bid. Different types of auctions are: English auction, Dutch auction, the first-price sealed-bid auction, and second-price sealed-bid auction or Vickery auction [22, 23]. Both English and Dutch auction are iterative, and price signals are continuously being fed back to the bidders. The first and second-price sealed-bid auctions are a single round auction, where bidders submit a sealed bid, and the bidder with the highest bid is selected as the winner. In the first-price sealed-bid auction, the winner pays the bid price itself whereas, in the second-price sealed-bid, the winner pays the price of the second-highest bid. According to the revenue equivalence theorem, even though the auction mechanism varies from each other, they yield the same expected revenue for the auctioneer when one item is being sold [24]. A Vickery auction or second-price sealed-bid auction allocates an item to the bidder who values it the most. Hence, Vickery auction for task scheduling can be justified, as it is

employed to find the best VM for a task. Nash Equilibrium (NE) is a concept used in game theory where every player chooses their best strategy and hence helps to get an optimal outcome of a game. Like game theory, a task scheduling problem involves decision makers (e.g., task or VM) to obtain the best mapping between task and VM. Further, NE points can be considered the best scheduling decision possible for tasks and VMs.

1.4.5 Primary-Backup Approach

Two popular techniques used to support fault-tolerant scheduling is resubmission and replication. Resubmission usually leads to a long finish time for tasks and may cause missing the task's deadline [25]. Whereas in replication, multiple copies of a task are allocated to different resources to ensure task completion before its deadline. It is noteworthy that two copies of replication can realize the balance between fault-tolerance and resource utilization. Two copy replication, i.e., Primary-Backup (PB) model, is widely used by researchers. In the PB model, two copies of one task run on two different VMs, and an acceptance test is used to check the correctness of results [5, 25–28]. If VM with the primary copy failed, the backup copy of the task is still running on another VM to ensure the task's successful completion within the deadline. The backup copy can be active or passive. A task's backup copy is executed in a passive backup copy scheme, only if its primary copy fails. It needs larger laxity, which is the gap between the deadline and finish time of the primary copy of a task so that backup copy can finish its execution before the task's deadline. In the active backup scheme, primaries and backup copies are executed simultaneously. Besides, to improve system schedulability and resource utilization, the overlapping method is employed in the PB model. Backup-Backup (BB) overlapping and PB overlapping are two popular overlapping techniques. In BB overlapping, multiple distinct backup copies are overlapped in a single computing resource whereas, in PB overlapping, primary copies are overlapped with backup copies of another task.

1.5 Performance Evaluation

This section presents the design of simulation environment and performance metrics used to evaluate the algorithms.

1.5.1 Simulation Environment

This thesis employs a Discrete Event Simulation (DES) method to evaluate the proposed algorithms. Execution of real-time application with timing constraint and numerous requirements (e.g., energy, cost, etc.) on real cloud infrastructure is time-consuming and costly. Thus, a DES is employed to evaluate the algorithm, allowing us to experiment and control various metrics to assess the algorithms under multiple scenarios effectively.

Figure 1.1 shows the simulation framework that consists of a task generator, task scheduler, and VM pool. Different components are added to the scheduler component based on the needs of the proposed approach. A task generator is designed to generate real-time tasks where task arrival follows the Poisson distribution. VM pool consists of a finite set of VMs and represents a cloud environment for task scheduling. Task scheduler uses scheduling mechanisms (algorithms) and various constraints to map a task to an appropriate VM in the VM pool. The development of IT infrastructure and applications make task and VM heterogeneity obvious in the cloud system. Different scenarios are considered varying arrival rate, deadline, task and VM count, task, and VM heterogeneity to show the effectiveness of proposed algorithms.

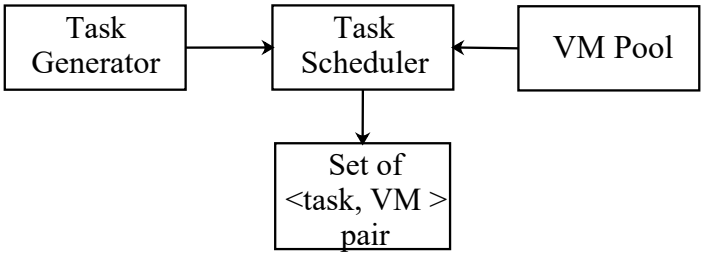


Figure 1.1: Simulation Framework

1.5.2 Performance Metrics

The metrics considered to evaluate the performance of the algorithm are Success Ratio (SR), total energy consumption (δ), and makespan (τ). The Success Ratio is defined as the ratio between the number of tasks meet the deadline by the total number of tasks. Total energy consumption is defined as the summation of energy consumed during the execution of all tasks in the system. Makespan is defined as the total execution time of all the tasks in the system.

1.6 Thesis Organization

The thesis is organized into seven Chapters. Contributions of each Chapter are summarized below:

Chapter 1 presents the objective, methodology used, and structure of the thesis.

Chapter 2 outlines cloud system model and a brief review of real-time task scheduling in the cloud system.

Chapter 3 proposes a VM scoring based scheduling algorithm using the TOPSIS analysis method for real-time tasks to minimize energy consumption and execution cost.

In **Chapter 4**, the Learning Automata (LA)-based scheduling algorithm is introduced for the real-time task to reduce energy consumption and makespan.

In **Chapter 5**, the game theory is applied to find a solution of bi-objective (minimization

of energy consumption while ensuring high reliability) real-time task scheduling problem in the cloud system.

In **Chapter 6**, Primary-Backup (PB) based fault-tolerant scheduling algorithm is presented for real-time tasks taking into account energy consumption and reliability.

Chapter 7 concludes the thesis by highlighting the contributions made in the thesis and the future scope of the work.

Chapter 2

Literature Survey

2.1 Introduction

Cloud computing manages a variety of virtualized resources, which makes the scheduler a significant component. The scheduler uses scheduling algorithms to allocate VM to execute a task. There exist various types of scheduling based on the task type, such as non-real-time and real-time. This thesis has focused only on real-time task scheduling. The basic idea of real-time task scheduling is to enhance system performance while guaranteeing timing constraints. Figure 2.1 shows the taxonomy of real-time task scheduling, as considered by various researchers. Depending on the architecture type, real-time scheduling can be classified as centralized and decentralized. In *centralized* approach, scheduling decision is made by a central entity like cloud scheduler, whereas in *decentralized* approach, scheduling decision is distributed among nodes participating in the system. Depending on scheduling technique, real-time task scheduling can be classified as heuristic based, metaheuristic based, reinforcement learning based, and game theory based. *Heuristic* technique is an approach to problem-solving where the solution is generated by exploiting the heuristic. The obtained solution may not be optimal but sufficient to reach the goal. It is a problem specific approach, and its performance relies on the effectiveness of heuristic. Min-Min, Max-Min, list scheduling, etc. are well-known heuristic techniques. *Metaheuristic* technique imitates a natural, physical, or biological principle resembling a search or an optimization process. It is a problem independent approach. The Metaheuristic technique produces near-optimal solutions but may take more execution time due to the absence of a convergence point. Genetic Algorithm (GA)([29]), [30],[31], Particle Swarm Optimization (PSO) ([32],[33]) , etc. are popularly used metaheuristic techniques. In *reinforcement learning* based method ([34] [35]), scheduling decision is made by continuously interacting with the environment (e.g., cloud system) that gives a reinforcement value for each of the action (e.g., mapping of a task to a VM) until the best decision possible for a problem is not found.

Game theory can be viewed as 'games of strategy' where players interact according to a particular strategy to obtain the optimal strategy that maximizes or minimizes its payoff. In a game-theoretic task scheduling problem ([36] [37]), the machine or users can act as a player or strategy, and the payoff can be represented by makespan or execution cost, etc.

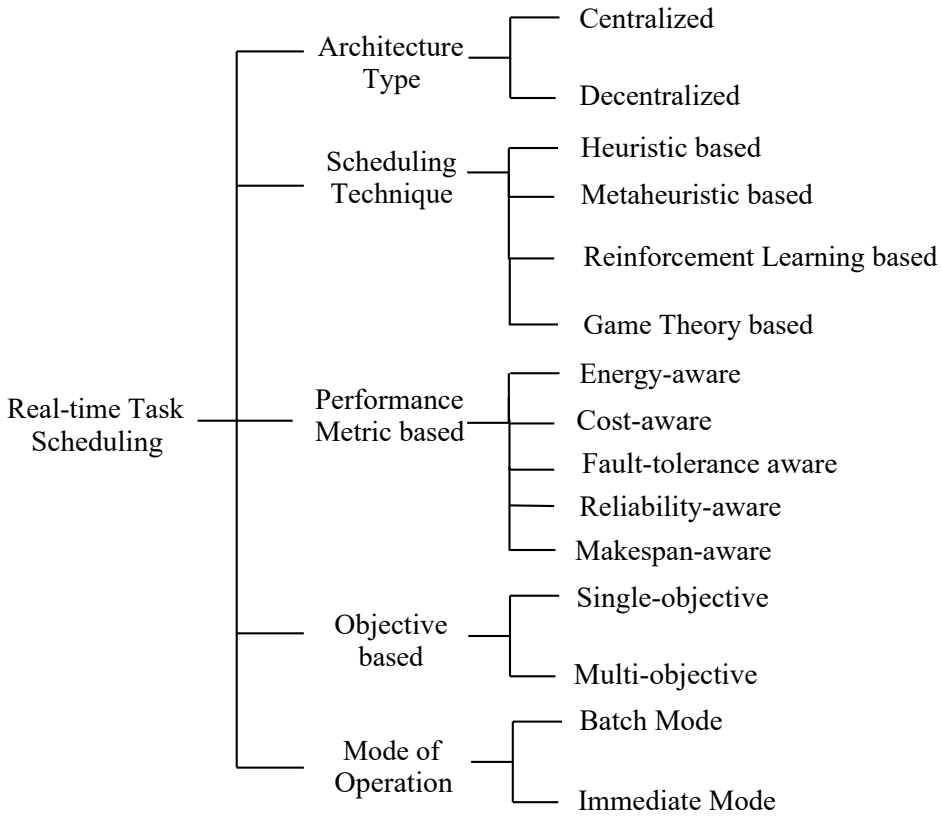


Figure 2.1: Taxonomy of Real-time Task Scheduling

Here, the players can interact cooperatively or non-cooperatively to minimize its payoff (e.g., makespan). The outcome of the game is considered as a solution to the scheduling problem. Based on the metrics used to measure the performance of the system, the task scheduling algorithms can also be categorized as *energy-aware* ([1],[15]), *cost-aware*([10], [7]), *makespan-aware*([38], [39]), *fault-tolerance aware* ([11], [13]), etc. Further, task scheduling algorithms can be designed to optimize only a single performance metric or multiple performance metrics. Based on the number of performance metrics used in task scheduling algorithms' objective, it can be *single-objective* or *multi-objective*. Similarly, the mode of operation divides a task scheduling algorithm as a *batch mode* or *immediate mode*. Ready to execute tasks are scheduled in *batch mode*, whereas tasks are scheduled as soon as they arrive in *immediate mode*. Minimum Execution Time (MET) and Minimum Completion Time (MCT) are batch mode algorithms, whereas Min-Min, Max-Min, etc. are immediate mode scheduling algorithms.

2.2 Cloud System Model

This section details the various models used in this work, which include the VM model, task model, scheduling framework, analytical model, energy model, cost model, and reliability model.

2.2.1 VM Model

The heterogeneous cloud environment is defined by the set $\mathcal{V} = \{vm_1, vm_2, ..., vm_m\}$, where $vm_j \in \mathcal{V}$ is characterized by following attributes:

- Processing capacity or speed vm_j^{sp} which is measured in terms of Million Instructions Per Second (MIPS),
- Execution cost vm_j^{ec} per time unit,
- Failure rate vm_j^{fr},

2.2.2 Task Model

Let $\mathcal{T} = \{tk_1, tk_2, ..., tk_n\}$ be the set of independent tasks that arrive dynamically at a particular instant. Each task, $tk_i \in \mathcal{T}$, has the following attributes:

- Arrival time tk_i^{ar},
- Task size tk_i^{sz} which is measured in terms of Million Instruction (MI) or Mega Byte (MB). MI of the task can be calculated using formula specified in [40] as: MI = tsk_r^{sz} in MB * (Number of Instruction (I) / Byte (B)),
- Task deadline tk_i^{dl}.

Let et_i^j is the expected execution time of task tk_i on VM vm_j and is known before the execution. It is computed as:

$$et_i^j = \frac{tk_i^{sz}}{vm_j^{sp}}. \tag{2.1}$$

It is considered that a new task tk_i is affixed to the end of previously allocated tasks on that VM. Let, et_i^j represents an element of Expected Time to Compute (ETC) matrix, where $i = \{1, 2, ..., n\}$ and $j = \{1, 2, ..., m\}$. The start time st_i^j for tk_i on VM vm_j can be estimated as:

$$st_i^j = max\{ft_q^j, tk_i^{ar}\}. \tag{2.2}$$

Equation 2.2 says that, new task tk_i can start on vm_j either after the completion of previously assigned task tk_q or just after the arrival, whichever is later. It is updated after each task execution on vm_j. The finish time ft_i^j of tk_i on vm_j can be calculated as:

$$ft_i^j = st_i^j + et_i^j. \tag{2.3}$$

The finish time indicates whether a task's deadline can be guaranteed or not. Let d_i^j is the indicator used to show whether a task tk_i executing on vm_j met its deadline or not (Equation 2.4).

$$d_i^j = \begin{cases} 0 \;\; if \;\; ft_i^j > tk_i^{dl} \\ 1 \;\; if \;\; ft_i^j \leq tk_i^{dl} \end{cases} \tag{2.4}$$

2.2.3 Scheduling Framework

A generalized real-time task scheduling framework is shown in Figure 2.2. A user request

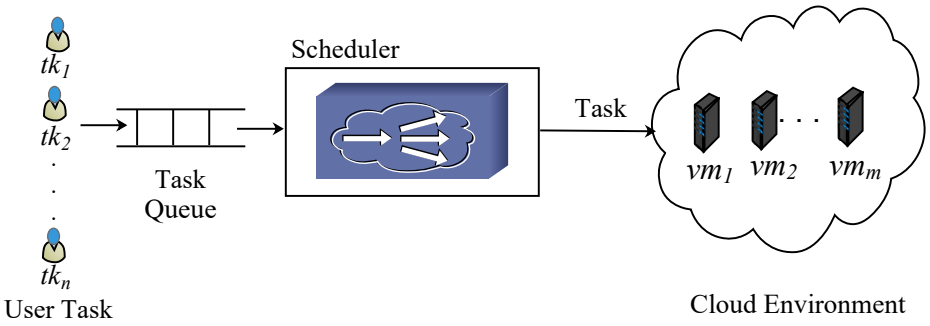


Figure 2.2: Generalized Real-time Task Scheduling Framework

is appended to the task priority queue. The scheduler removes a task from the queue and decides when and where the task to be executed. This decision is made, taking into account the scheduling algorithm and the availability of VMs in the cloud system.

2.2.4 Energy Model

The execution environment, cooling system, and power conditioning are the main contributors to energy consumption in the cloud system. The execution environment consists of VM and is the basis of the energy model used in this thesis. In general, the energy consumed by a VM depends on its state. Usually, a VM can be in active state or idle state. It is assumed that a VM is said to be in active state when it is executing a task; otherwise, it is idle. A study presented in [3, 4, 35] says that in the idle state, a VM consumes $[60-70]\%$ of energy consumed in the active state.

Let, the total energy consumed by VM vm_j is ϱ_j, which can be expressed as:

$$\varrho_j = (\xi_j^a + \xi_j^i) \times vm_j^{sp}, \tag{2.5}$$

where ξ_j^a and ξ_j^i are energy consumed by vm_j in active and idle state respectively. Let, the total execution time (μ_j) of all tasks assigned to vm_j is

$$\mu_j = \sum_{i=1}^{n} \mathcal{X}_i^j \times ft_i^j, \tag{2.6}$$

where binary indicator $\mathcal{X}_i^j = 1$ if tk_i is assigned to vm_j, or $\mathcal{X}_i^j = 0$ otherwise. Makespan (τ) is defined as the maximum execution time among all the VMs in the system, and it is expressed as:

$$\tau = max(\mu_j). \tag{2.7}$$

Now, the active and idle state energy consumption of vm_j can be computed as:

$$\xi_j^a = \mu_j \times \sigma_j, \tag{2.8}$$

$$\xi_j^i = (\tau - \mu_j) \times 0.6 \times \sigma_j, \tag{2.9}$$

where $\sigma_j = 10^{-8} \times (vm_j^{sp})^2$ Joules/MI [3, 41, 42]. The total energy consumption (δ) of the cloud system is evaluated as Equation 2.10 and measured in Joules.

$$\delta = \sum_{j=1}^{m} \varrho_j \tag{2.10}$$

2.2.5 Cost Model

Popular cloud service providers like Amazon Web Service (AWS), Google, and Azure use pay-per-second billing scheme [43], which is the basis of the cost model used in this thesis. The heterogeneity of tasks and VM causes different execution cost for the same task on different VMs. Let, ec_i^j is the execution cost of a task tk_i on VM vm_j per time unit (Seconds) and is calculated as:

$$ec_i^j = vm_j^{ec} \times et_i^j. \tag{2.11}$$

The total execution cost of a cloud system is formulated in Equation 2.12 and measured in the cost unit ($).

$$ec = \sum_{j=1}^{m} \sum_{i=1}^{n} ec_i^j \times \mathcal{X}_i^j \tag{2.12}$$

2.2.6 Reliability Model

The reliability of a cloud system can be assessed through the probability of successfully executing all the incoming tasks at a particular instant. The reliability model assists the cloud scheduler in mapping tasks on the cloud infrastructure and helps to achieve fault-tolerance. Reliability of a task tk_i on vm_j, (rel_i^j) is defined as the product of failure rate vm_j^{fr} of vm_j and execution time tk_i on vm_j, i.e., et_i^j. Mathematically, it can be written as:

$$rel_i^j = e^{-vm_j^{fr} \times et_i^j \times \mathcal{X}_i^j}. \tag{2.13}$$

So, the overall reliability of vm_j is computed as:

$$vm_j^{rel} = rel_i^j \times \prod_{tk_k \ on \ vm_j} rel_k^j. \tag{2.14}$$

With an increase in execution time (i.e., VM use), the chances of VM failure increases. Hence, the reliability of the cloud system decreases gradually.

2.2.7 Analytical Model

Queuing theory is widely used to model and study the performance, and QoS of various ICT systems [16, 17]. An analytical model based on a queuing system is used to study the proposed heterogeneous cloud system's behavior. In this context, Li *et al.* [10] introduced a cost-efficient scheduling technique for the cloud system modeled as $M/GI/1/PS$ to meet the response time and deadline constraints. Kafhali *et al.* [16] presented an analytical queuing model to show the dynamic behavior of fog nodes in the IoT environment. Khazaei *et al.* [44] used a $M/G/m/m+r$ queuing system to model a cloud computing center and find the probability of blocking new request, the probability of immediate service to a request, etc. Further, a queuing theory-based model is employed in [45] to assure QoS in terms of response time.

Table 2.1: Queuing Model Used by Researchers

Research works	**Queuing Model**	**Working Environment**
Li *et al.* [10]	$M/GI/1/PS$	Heterogeneous
Kafhali *et al.* [16]	$M/m/n/K$	Homogeneous
Khazaei *et al.* [44]	$M/G/m/m+r$	Homogeneous
Vilaplana, *et al.* [45]	$M/M/1$ and $M/M/m$	Homogeneous
Bruneo [46]	Task Arrival : Poisson distribution, Service time : Exponential distribution	Homogeneous
Liu *et al.* [47]	$M/G/1$	Heterogeneous
Zhang *et al.* [48]	$M/G/1$	Homogeneous
Mei *et al.* [49]	$M/M/n/n$	Homogeneous
Fan *et al.* [50]	Task Arrival : Poisson distribution, Service time : Exponential distribution	Heterogeneous

Bruneo in [46] presented a stochastic reward net-based analytical model for analyzing an IaaS cloud system's behavior. The queuing model is adopted by Liu *et al.* [47] to study the performance of cloud services with resource sharing among VMs in the cloud system. Zhang *et al.* [48] used the Stackelberg game concept to analyze the pricing and resource allocation problem in a three-tier IoT fog network. Mei *et al.* [49] realized a cloud system as a $M/M/n/n$ multiserver queuing system, and address the profit maximization problem. The authors in [50] designed an application-aware workload allocation scheme for minimizing the response time of IoT application requests on the cloud. Khazaei *et al.* [51] used an analytical model to study the complexity of the cloud computing system. Ghosh *et al.* [52] developed a stochastic analytical model for performance quantification of IaaS cloud. An analytical model is presented in [53] to estimate the performances of deadline-based services in a heterogeneous cloud system. Table 2.1 shows the list of the queuing model and environment considered by researchers for modeling the cloud system. Most researchers modeled a cloud system with Poisson task arrival, exponential service

time, and homogeneous working environment. As the arrival of tasks to the cloud system

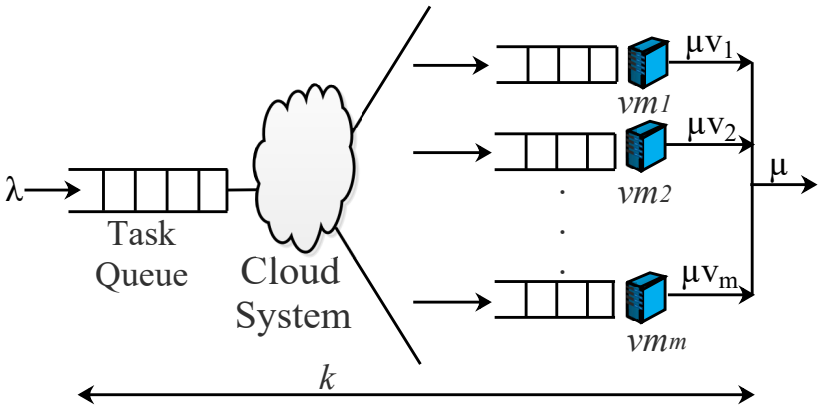


Figure 2.3: Analytical Model of a Cloud System

is independent of one another, the reasonable assumption is that the task arrival follows a Poisson distribution. It is assumed that the arrival rate of an individual task (λ_r) follows a Poisson distribution and the aggregate arrival rate λ is also Poisson distributed (given in Equation 2.15).

$$\lambda = \sum_{r=1}^{n} \lambda_r \tag{2.15}$$

It is assumed that the service rate of tasks on a VM is exponentially distributed with an aggregate value of μv_j (i.e., vm_j^{sp}). So, the service time or execution time (et_i^j) of a task on a VM also follows an exponential distribution. Hence, this research's cloud system is modeled as a $M/M/m/k$ multi-process queuing system, as shown in Figure 2.3. A multi-process system is defined as the system with one arrival entrance, waiting queue, and one exit point but can execute more than one task simultaneously. The third parameter, m of the queuing system, indicates the number of VMs in the cloud system. The maximum number of tasks possible in the multi-process system is denoted by the fourth parameter, k, of the queuing system. It signifies the length (or size) of the queuing system's task queue (or buffer). Each

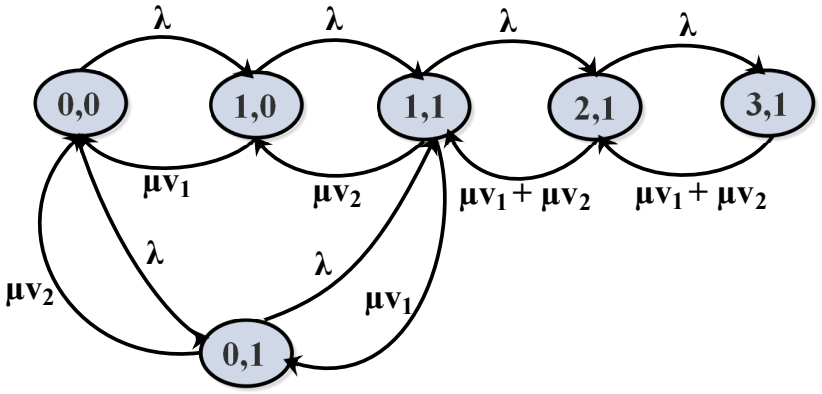


Figure 2.4: CTMC Model of $M/M/2/4$

VM can be modeled by a $M/M/1$ queuing system and the whole cloud system can be

considered as one $M/M/m/k$ queue or m number of $M/M/1$ queues with finite queue length. A Continuous Time Markov Chain (CTMC) model of cloud system with 2 VMs and 4 tasks, i.e., $M/M/2/4$ is shown in Figure 2.4. Let, $\mu v_1 > \mu v_2$. The state of the CTMC model corresponds to the number of tasks to be serviced by two VMs. Let S_0 indicates state $(0, 0)$, i.e., no VM is executing a task. S_1 represents $(0, 1)$ or $(1, 0)$, i.e., either vm_1 or vm_2 is executing a task. S_2 denotes state $(1, 1)$ that shows both VMs are executing a task. S_3 and S_4 represents state $(2, 1)$ and $(3, 1)$ respectively.

By solving time reversibility equations, the limiting or blocking probabilities is generated for the system. Time reversibility equation for S_0 is shown in Equation 2.16.

$$\lambda\pi_0 = (\mu v_1 + \mu v_2)\pi_1 \Longrightarrow \pi_1 = \left(\frac{\lambda}{\mu v_1 + \mu v_2}\right)\pi_0 \tag{2.16}$$

Similarly, Equation 2.17 shows the time reversibility equation for state S_1.

$$\lambda\pi_1 = (\mu v_1 + \mu v_2)\pi_2 \Longrightarrow \pi_2 = \left(\frac{\lambda}{\mu v_1 + \mu v_2}\right)^2\pi_0 \tag{2.17}$$

Equation 2.18 is derived by generalizing Equation 2.17 for m number of VMs and k number of tasks.

$$\pi_i = \left(\frac{\lambda}{\mu v_1 + \mu v_2 + ... + \mu v_m}\right)^i \pi_0 = \left(\frac{\lambda}{\mu_t}\right)^i \pi_0, \tag{2.18}$$

where $i = 0, 1, 2, ..., k$ and $\mu_t = \sum_{j=1}^{c} \mu v_j$.

To preserve the law of probability, the sum of probabilities of having number of tasks starting from 0 to k must be one. This is shown in Equation 2.19 as:

$$\sum_{i=0}^{k} \pi_i = 1. \tag{2.19}$$

Let, $\sigma = \frac{\lambda}{\mu_t}$. Putting σ value in Equation 2.18 and solving Equation 2.19, Equation 2.20 is formulated.

$$\left[1 + \sigma + ... + \sigma^k\right]\pi_0 = 1 \Longrightarrow \left[\frac{1 - \sigma^{k+1}}{1 - \sigma}\right]\pi_0 = 1 \Longrightarrow \pi_0 = \frac{1 - \sigma}{1 - \sigma^{k+1}} \tag{2.20}$$

The blocking probability or limiting probability π_k indicates that, there are k number of tasks being processed in the $M/M/m/k$ queuing system. It is computed in Equation 2.21 as:

$$\pi_k = \sigma^k \left[\frac{1 - \sigma}{1 - \sigma^{k+1}}\right]. \tag{2.21}$$

A new task will have to wait if there are more than m number of tasks in the system. Thus

the number of tasks in the waiting queue (η_q) is calculated as:

$$\eta_q = \sum_{i=m+1}^{k} (i-m)\pi_i = \frac{\sigma^m}{1-\sigma^{k+1}} \left[\frac{\sigma - (k+1)\sigma^{k+1} + k\sigma^{k+2}}{1-\sigma} \right]. \tag{2.22}$$

Similarly, the total number of tasks in the system (queue + service) is computed as:

$$\eta_s = \sum_{i=1}^{k} i\pi_i = \frac{\sigma - (k+1)\sigma^{k+1} + k\sigma^{k+2}}{(1-\sigma)(1-\sigma^{k+1})}. \tag{2.23}$$

Now the average waiting time (ω_{t_r}) and response time (Υ_{t_r}) can be calculated as:

$$\omega_{t_r} = \frac{\eta_q}{\lambda}, \tag{2.24}$$

$$\Upsilon_{t_r} = \frac{\eta_s}{\lambda}. \tag{2.25}$$

The proposed analytical model is used in simulation to analyze the performance of different task scheduling algorithms.

2.3 Real-time Task Scheduling

This section discusses various task scheduling algorithms, specifically focusing on the real-time task.

2.3.1 Energy-aware Scheduling

Energy conservation in cloud computing is receiving a great deal of attention among the research community, and efficient scheduling methods have been overwhelmingly investigated. Zhu *et al.* [1] proposed a rolling horizon optimization policy-based scheduling algorithm, EARH, to conserve energy in the virtualized cloud. Chen *et al.* [2] presented an Energy-efficient Online Scheduling Algorithm (EONS) to achieve energy-efficiency and better resource utilization for real-time workflows in the cloud data center. Authors in [3] have presented a metaheuristic based approach to reduce energy consumption and makespan in a cloud-based fog environment. Gao *et al.* [4] proposed the "Guided Migrate and Pack" (GMaP) scheduling framework based on VM migration to maximize energy-efficiency while reducing Service Level Agreement (SLA) violation due to deadline miss. To save energy, the authors in [14] have presented EEVS, an energy-efficient algorithm using the DVFS technique where first, they find the optimal performance-power ratio of Physical Machine (PM) and then assign VM to PM with the highest ratio. Authors in [15] have used the DVFS technique to develop an energy-efficient algorithm with monetary cost constraints for workflows in the cloud system. Chen *et al.* [54] have proposed an energy-efficient reactive scheduling algorithm, ERECT, using the DVFS technique to minimize energy consumption. Shi and Jang [41] proposed a probabilistic scheduling algorithm to reduce

energy consumption in the mobile cloud.

Juarez *et al.* [9] and Li [55] have proposed a heuristic to reduce energy consumption and execution time for parallel tasks in the cloud environment. Zhang *et al.* [56] have used historical scheduling information for the pre-creation of VMs to improve the performance and execution of a real-time task. Yassa *et al.* [32] used DVFS and PSO techniques to optimize makespan, cost, and energy for workflow scheduling. In [57], a dynamic VM consolidation technique is used to realize energy-efficiency in the cloud data center without committing SLA violations. Xing *et al.* [58] have utilized VM migration and resource fairness concept to optimize energy usage for IoT applications in the cloud environment. A greedy scheduling algorithm named Most Energy-Efficient First (MESF) presented in [59] saves energy by assigning a task to the most efficient server based on energy profile. Koodziej *et al.* [29] have employed the DVFS model and GA for solving energy-aware scheduling problems in the computational grid. In [60], authors have proposed a heterogeneity aware resource management system called HARMONY to decrease the total energy consumption and performance penalty in a cloud environment. DVFS technique is used in [2][61] to find the trade-off between performance and energy-efficiency. The work presented in [62] minimizes energy consumption by turning off the most effective processor in the cloud system from an energy-saving perspective. Researchers in [63] have presented a routing protocol considering hop distance, temperature, and energy for implanted biosensor networks to reduce packet delivery latency. Quang-Hung *et al.* [30] have proposed a GA based VM allocation scheme to minimize the total energy consumption of computing servers. In [64], authors have proposed a VM placement method to minimize power consumption while enhancing resource utilization. Arroba *et al.* [65] have proposed power and thermal-aware strategies for optimizing cooling and computing efficiency in the cloud data center.

2.3.2 Cost-aware Scheduling

The cost factor also plays a vital role in the task schedule to evaluate a cloud system's performance, but a few works have taken this into account. Li *et al.* [10] proposed an algorithm named CEAS using the DVFS technique to reduce execution cost and energy consumption of scientific workflows in the cloud. Pham *et al.* [7] proposed Cost-Makespan aware Scheduling (CMaS) algorithm to handle the trade-off between performance and cost while executing IoT applications. Zhang *et al.* [38] introduced Energy and Deadline Aware with Non-Migration Scheduling (EDA-NMS) algorithm to minimize energy consumption and cost. Hu *et al.* [39] presented time and cost-efficient Flutter scheduling algorithm for big data processing in geo-distributed cloud data centers. Cai *et al.* [66] presented the Delay-based Dynamic Scheduling (DDS) algorithm to reduce resource renting cost of workflow execution in the cloud. Garg *et al.* [67] proposed MaxCTT, SuffCTT, and MinCTT scheduling algorithms to optimize the trade-off between execution time and cost.

An Energy-aware Stochastic Task Scheduling algorithm (ESTS) is presented in [68] to maximize the guaranteed confidence probabilities under deadline and energy consumption budget constraints in a heterogeneous computing system. Poola *et al.* [69] presented a robust scheduling algorithm for workflows, which reduces makespan and cost. A regression model is used by researchers in [70] for provisioning cloud resources to find the trade-off between cost-saving and QoS requirements. Chen *et al.* [34] considered the uncertainty of task execution time and data transfer time in their scheduling algorithm to optimize service renting cost for workflows in the cloud. Guo *et al.* [33] proposed the Cost-Effective Fault-Tolerant scheduling algorithm (CEFT) using the PSO technique for real-time tasks in the cloud system. Alkhanak *et al.* [71] presented an analysis of existing approaches to solve the problem of cost optimization in Scientific WorkFlow Scheduling (SWFS). In [72], authors employed dynamic programming and greedy concept to reduce execution cost under timing constraints. Zheng *et al.* [73] proposed the MapReduce Application Scheduling Algorithm (MASA) to reduce data analysis costs while minimizing SLA violations.

2.3.3 Makespan-aware Scheduling

Zhang *et al.* [38] used Bayes classifier to classify tasks based on historical scheduling data. They proposed a two-stage scheduling scheme and evaluated its performance using makespan, waiting time, and VM utilization. Hu *et al.* [39] designed a task scheduling algorithm to minimize completion time and network cost of big data processing jobs in geo-distributed data centers. Poola *et al.* [69] proposed a scheduling policy to minimize makespan and cost for scientific workflows while achieving fault-tolerance. Panda *et al.* [74] proposed scheduling techniques based on min-max and median-max to reduce makespan while improving average cloud utilization in a multi-cloud environment. Stavrinides *et al.* [75] used the Earliest Deadline First (EDF) and best fit theory to guarantee applications' execution within deadline constraint while reducing makespan and cost charged to the user. Sahoo *et al.* [76] proposed EDF based algorithm using Best Fit, Worst Fit, and First Fit concept to optimize guarantee ratio, VM utilization, and throughput while guaranteeing deadline constraint. Su and Wang [77] used the Pareto dominance concept to select a VM to achieve cost efficiency and makespan minimization. Rehman *et al.* [31] proposed a Multi-Objective Genetic Algorithm (MOGA) approach to minimize makespan under budget and time constraint. The authors in [78] introduced big data analytics framework using the Fuzzy inference system to reduce the processing time of high-frequency data stream. In [79] researchers have provided a solution to the sensor deployment and scheduling problem considering network lifetime in the wireless sensor network.

2.3.4 Use of Learning Automata-based Approach

Learning Automata (LA) theory is appropriate for the environment, which is dynamic, complex, and there is a large number of uncertainties like cloud environment, computer networks, etc. [34]. Sahoo *et al.* [35] used the LA concept for real-time task execution that minimizes energy consumption and makespan in a cloud environment. Narendra *et al.* [80] presented a survey in the area of learning automata. Their study mainly focused on the norms and behavior of LA, reinforcement, or learning schemes, the convergence of learning algorithms, and several automata' interaction. Various applications of LA are also discussed, like parameter optimization and decision making. Authors in [81] discussed how LA behaves with a changing number of actions. Misra *et al.* [82] proposed an LA-based framework to improve the performance of QoS-enabled cloud services concerning response time and speed-up. Rezvanian *et al.* [83] used LA to find the solution of the minimum vertex-covering problem in a stochastic graph. Ranjbari *et al.* [84] proposed an algorithm based on LA to detect the overloaded Physical Machine. The prevention from Physical Machine overload reduces VM migration count and helps consolidate VMs, which minimizes energy consumption. In [85], authors have employed LA theory to develop a prediction model for cloud resource usage. An LA-based ranking algorithm is introduced in [86], where a learning automaton ranks the search documents based on user feedback. Venkataramana *et al.* [87] proposed LA-based task assignment architecture for a heterogeneous computing system to achieve load balancing and minimum total execution time. Authors in [88] used the LA concept to minimize the energy consumption in a heterogeneous cloud environment.

2.3.5 Fault-tolerant Task Scheduling

Cloud experiences failures and performance fluctuations when executing tasks. This generates the need for a high fault-tolerant cloud environment, which can be achieved through an efficient fault-tolerant scheduling algorithm. Han *et al.* [11] presented a scheduling algorithm named ARCHER for hybrid real-time tasks integrating Primary-Backup concept and checkpoint strategies to achieve a balance between fault-tolerance and resource utilization in the cloud. Yan *et al.* [13] employed both resubmission and replication strategies to achieve fault-tolerance and efficient resource utilization. Li *et al.* [55] proposed a failure aware energy-efficient scheduling algorithm for the cloud data center considering computing and cooling energy, and the reliability of servers. Xie *et al.* [89] proposed a QFEC fault-tolerant scheduling algorithm with minimum execution costs while satisfying the reliability constraint of workflows. But this scheme uses multiple replicas of a task and doesn't consider overlapping technique. Zhang *et al.* [38] proposed a Support Vector Machine (SVM)-grid based online fault detection model to improve the system performance. Zhu *et al.* [28] developed a fault-tolerant task scheduling method using the Primary-Backup concept along with overlapping mechanism and VM

migration for real-time workflows in the cloud. Latiff *et al.* [90] introduced a league championship based scheduling mechanism to reduce the task execution failure rate. Zhu *et al.* [25] proposed a Service-Aware fault-tolerant scheduling algorithm using the Overlapping technique, SAO, to improve resource utilization and schedulability. Wang *et al.* [5] proposed an elastic resource provisioning mechanism taking into account Backup-Backup overlapping and VM migration to ensure both fault-tolerance and high resource utilization in clouds.

Ding *et al.* [91] used the Primary-Backup model and resource migration approach to achieve fault-tolerance and high resource utilization. Ghosh *et al.* [92] achieves a high acceptance ratio by the integration of overloading and deallocation techniques. Further, some works have been done in fault detection of components in the cloud system. Smara *et al.* [93] construct a fail-silent cloud module to detect transient fault and response time failure in the cloud system. Gabel *et al.* [94] used proactive approach to identify machines with latent faults. Bui *et al.* [95] proposed a fuzzy logic based algorithm and prediction technique to identify faults occurring in the IaaS system. Nimkar *et al.* [96] proposed security framework for federated IaaS cloud. The authors in [97] proposed a fault-tolerant strategy to ensure the reliability and real-time requirement of a cloud application. Qiu *et al.* [6] used queuing theory and the Bayesian approach to evaluate the correlated metrics: reliability, performance, and power consumption of cloud service. Wen *et al.* [98] used an entropy-based method to quantify the most reliable cloud and then optimize the deployment of workflow in terms of entropy and monetary cost. Xiao *et al.* [99] proposed a heuristic approach that assigns each task to a processor with maximum reliability while ensuring energy constraint. Li *et al.* [100] proposed Back Propagation (BP) neural network-based scheduling method to guarantee system performance and reliability, satisfying cost and deadline constraints. Nik *et al.* [101] presented a scheduling approach that minimizes workflow execution cost while ensuring deadline and reliability constraints. Qin and Jiang *et al.* [26] used Backup-Backup overlapping and Primary-Backup overlapping to achieve fault-tolerance while improving the reliability of the heterogeneous system. Zhu *et al.* [102] proposed a fault-tolerant scheduling algorithm QAFT to improve reliability, schedulability, and resource utilization for real-time tasks in heterogeneous clusters. In [103], the authors have presented a security framework for the Internet of Medical Things (IoMT).

Researchers in [104] employed the Primary-Backup approach to minimize response time and replication cost while ensuring deadline constraint. Qu *et al.* [105] presented a cost-efficient auto-scaling policy according to fault-tolerant semantics using various spot instances. The authors in [106] presented an energy-aware fault-tolerant scheduling algorithm using active replication strategy and utilization of energy consumption for decision making under time and reliability constraint. Sun *et al.* [107] developed optimal request scheduling and resource management strategies considering reliability, performance, and energy consumption simultaneously. Malik *et al.* [108] presented a reliability assessment model that helps with scheduling to achieve fault-tolerance in the cloud system. Most

of the above work discusses techniques to handle a single fault in the system. In this context, Manimaran *et al.* [109] designed an algorithm to handle multiple faults at a time in a multiprocessor system. Further, they used a myopic distance algorithm, flexible backup overloading, and resource claiming technique to improve the system's performance. Al-Omari *et al.* [110] used dynamic grouping and backup overloading to handle multiple faults in a real-time multiprocessor system.

2.3.6 Use of Game Theory

In game theory, a task scheduling game is a game that models a scenario in which multiple users wish to utilize numerous processing machines (e.g., VM). In this context, Ranganathan *et al.* [21] proposed an auction based non-cooperative game to optimize the average power of a circuit during scheduling in behavioral synthesis. Li *et al.* [36] presented a non-cooperative game model for task modeling taking into account reliability value in cloud system . Yang *et al.* [37] used a cooperative game model for task scheduling, considering the reliability of the balanced task. Khan and Ahmad [111] proposed a cooperative game based solution to handle the real-time task allocation problem in the computational grid considering minimization of energy consumption and makespan simultaneously. The authors in [112] used a non-cooperative game to solve the server load balancing problem and VM placement problem in cloud computing. Li *et al.* [113] proposed an non-cooperative game theoretic approach to schedule linear deteriorating jobs in parallel machines. Researchers in [114] modeled the network selection and resource allocation in wireless access networks as a non-cooperative congestion game to minimize the selection cost. Further, a mathematical programming based method is designed to find Nash equilibria and optimize the cost function. Huu *et al.* [115] designed the resource allocation model using a combinatorial auction mechanism to minimize energy consumption in the cloud environment.

Wu *et al.* [116] used Modified Vickery Auction (MVA) mechanism and Continuous Double Auction (CDA) mechanism for resource allocation, taking into account procurement cost. Su *et al.* [117]developed a Stackelberg game-theoretic resource allocation scheme for media cloud to achieve high Quality of Experience (QoE) while maximizing the profit. Ding *et al.* [118] proposed a reverse online auction based resource allocation where the multi-attribute bid value is defined based on price, memory, and speed. Wang *et al.* [119] proposed a decentralized Multi-Agent (MA) based VM allocation scheme to minimize energy usage in a cloud system. Authors in [120] presented a Vickery auction based framework for Data-as-a-Service (DaaS) in cloud computing. Zhou *et al.* [121] designed the job scheduling problem in networked manufacturing as an n-person non-cooperative game. Here Nash Equilibrium point is achieved with the help of GA while optimizing the makespan. Khan and Ahmad [122] compare and analyze non- cooperative, semi-cooperative, and cooperative game based resource allocation in grid computing. Das *et al.* [123] proposed auction theory based solution approach for resource allocation problem in mobile cloud

computing. He *et al.* [124] proposed a game-theoretic approach to provide service to the maximum user with minimum overall system cost in an edge computing environment.

2.3.7 Outcome of the Survey

The observations made from the literature study are listed below.

- VM migration, DVFS technique, and other techniques like VM consolidation, nature-inspired approach, and greedy approach have been well studied separately by the researchers for saving energy. The diversified strategies and need of the energy-efficient system allows one to explore different methods to achieve the desired goal.

- Further, a considerable amount of work has been done considering the benefits of the LA approach. Only a few of them deal with energy-aware scheduling of independent real-time tasks in the heterogeneous cloud environment. In this context, an LA concept is employed to design a scheduling framework for real-time tasks in the heterogeneous cloud environment. Authors in [87] also presented a task scheduling framework; however, the main distinction between their work and the proposed approach is two-fold. First, the absence of the virtualization concept in their work, whereas VM is the basic computation unit in the proposed work. Second, the proposed work takes the task's deadline into account while scheduling, which was not considered in earlier work.

- The works, as mentioned earlier, either focus on maximizing reliability with cost (or energy consumption) constraint or make a trade-off between these two without considering fault-tolerance. Compared with the existing research mentioned above, this work focuses on optimizing the bi-objective function considering energy consumption and reliability, and the fault-tolerance approach. The main distinction between existing researches and the proposed work is two-fold. First, the Primary-Backup (PB) model with overlapping technique is applied, and cloud virtualization is exploited to tolerate VM failure. Second, if the primary copy's execution is successful, resources reserved for a corresponding backup copy are reclaimed (deallocation process).

- Most of the research work using game theory takes time, QoS, or monetary profit as the goal without considering other requirements of cloud environment like energy consumption, reliability, etc. Unlike the above research, this work proposes a non-cooperative game model for the bi-objective task scheduling problem, including the minimization of energy consumption while satisfying reliability constraint.

2.4 Summary

This Chapter reviewed the real-time task scheduling algorithm and the proposed cloud system model. The review is organized based on methods and parameters based measures used for task scheduling. Further, a list is presented to show the study's outcome that point out the issues in existing scheduling approaches and how it is addressed in this thesis. In the rest of this thesis, a discussion of the proposed scheduling algorithms designed to address the real-time task scheduling problem in the cloud system is presented. The next Chapter presents a VM scoring based scheduling algorithm to reduce energy consumption and execution cost simultaneously.

Chapter 3
VM Scoring based Scheduling Algorithm

Abstract:- The development of large scale data centers containing thousands of computing resources consumes an enormous amount of energy, which contributes to the huge cost. Further, the development of computing services and applications demands improved energy consumption and execution cost. In this regard, this Chapter proposes VM scoring based Energy and Cost Aware (ECA) real-time task scheduling algorithm based on the TOPSIS analysis method to minimize the energy consumption and execution cost simultaneously while ensuring deadline constraint.

3.1 Introduction

The cost incurred in the cloud system can be monetary (e.g., execution cost, communication cost, storage cost, etc.) and temporal cost (e.g., earliest finish time, data transfer time, execution time, etc.). The estimated execution cost realizes the cost of processing a task at the VM and can be measured before scheduling. Besides, it assists the algorithm with decision making for scheduling real-time tasks. Execution cost is an essential contributor to operational cost, which can be reduced through cost optimization scheduling methods. The computing environment consisting of VMs is decided based on the task's size and other QoS constraints like a budget, deadline, energy consumption, etc. Each computing environment has a different specification, which ultimately affects the total cost. Moreover, the energy conservation problem is receiving increasing attention due to environmental and financial factors. The rapid development of the computing services and applications demands not only the optimization of cost through scheduling but also other constraints like energy consumption, makespan, etc. [71]. Hence the development of a multi-objective scheduling problem. There are different Multi-Objective Decision Making (MODM) methods to solve the above mentioned multi-objective scheduling problem. In this context, this work focuses on developing a scheduling algorithm that minimizes the bi-objective cost function defined by energy consumption and execution cost. Further, VM selection with the earliest finish time reduces the total execution time, i.e., makespan.

3.1.1 Scheduling Framework

The VM scoring based real-time task scheduling framework is shown in Figure 3.1. The scheduling framework consists of two components, namely *Schedulability Analyzer*, and *Resource Manager*, to accomplish the work. The *Resource Manager* contains two sub-components *Energy Calculator* and *Cost Calculator*. The working of each component

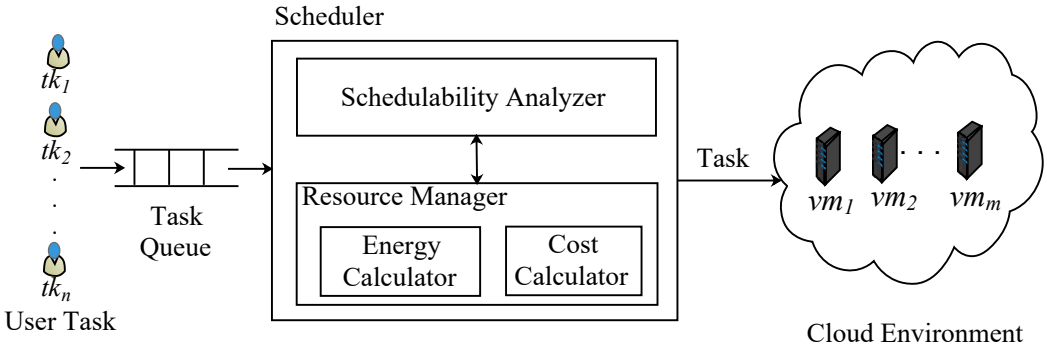


Figure 3.1: VM Scoring based Real-time Task Scheduling Framework

is explained below.

The primary responsibility of *Schedulability Analyzer* is to check whether an incoming task can be finished before its deadline or not. Additionally, it informs the *Resource Manager* if a task's deadline constraint is difficult to achieve so that new VMs can be added.

Resource Manager is used to generate $\langle task, VM \rangle$ pair based on Algorithm 3.2. Besides, it keeps track of provisioned cloud resources and deals with scalability issues. An *Energy Calculator* is used to compute the energy consumed, and a *Cost Calculator* generates the execution cost while executing a task on a VM. Both the calculated values are used to select an appropriate VM for a task.

3.2 Energy and Cost Aware (ECA) Scheduling Algorithm

The flowchart in Figure 3.2 shows the working of the ECA scheduling algorithm. For each task, a score value considering energy consumption and execution cost is computed for all VMs in $\mathcal{V}$. Then the VM list is sorted and stored in $sort\_vm\_score$. Then $y\%$ of VM is chosen. A VM with the earliest finish time and satisfying deadline constraint is selected to execute a task. If none of the VM satisfies the task's deadline constraint, then another $y\%$ of VM is chosen from the sorted list. This process continues until the sorted list is not empty. If a task can't be executed with the existing VMs, then a new VM is added to the cloud environment. The steps for the VM selection process in ECA are discussed below.

Step 1 : In a $m \times 2$ decision matrix $\mathcal{D}\_vm$, first column indicates energy consumption and second column indicates execution cost of a VM while executing a task. Let, $d\_vm(i, j)$ represents an element of $\mathcal{D}\_vm$ matrix, where $i = \{1, 2, ..., m\}$ and

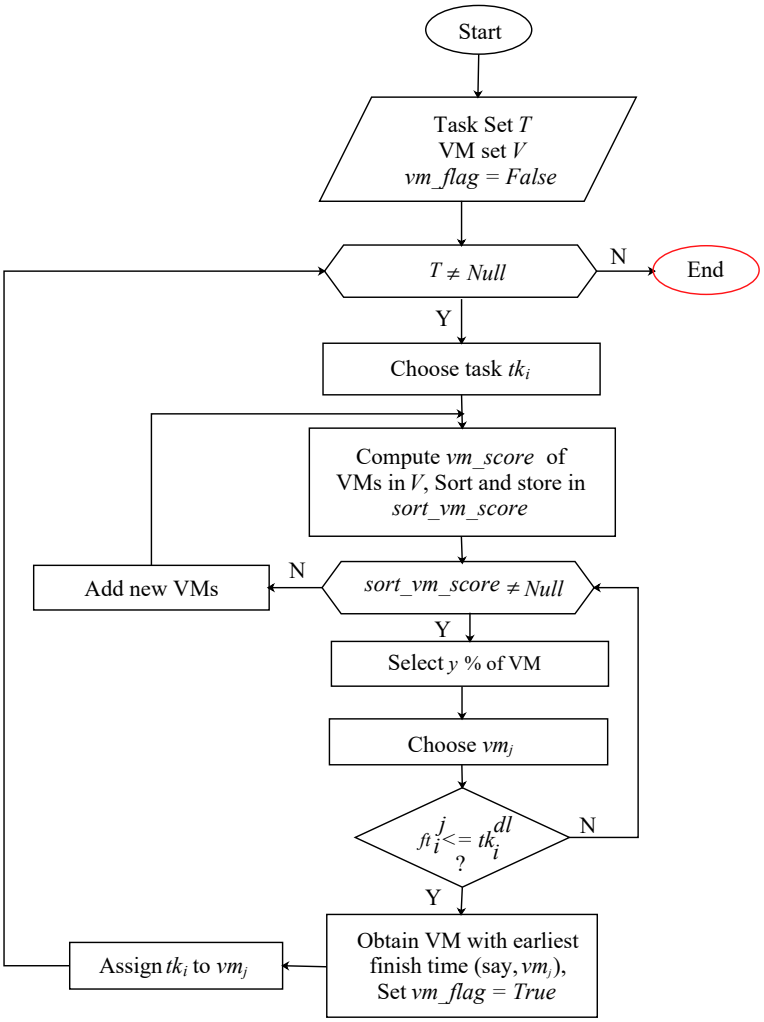


Figure 3.2: Flowchart of ECA Scheduling Algorithm

$j = \{1, 2\}$. An entry $d\_vm(1, 1)$ in $\mathcal{D}\_vm$ indicate the energy consumption of vm_1 whereas $d\_vm(1, 2)$ indicate the execution cost in vm_1.

$Step\ 2:$ As the measuring unit of energy consumption (Joule) and execution cost ($) is different, a normalization process is employed to obtain a comparable scale. The normalized decision matrix $\mathcal{ND}\_vm$ is a $m \times 2$ matrix, where each element (nd_vm) of matrix is computed as follows:

$$nd\_vm(i, j) = \frac{d\_vm(i, j)}{\sum_{i=1}^{n} d\_vm(i, j)} \times wt_j \quad \forall i, j, \tag{3.1}$$

where wt_j is the weight associated with energy consumption and execution cost.

$Step\ 3:$ A VM with the best performance according to energy consumption and execution cost is chosen as a positive ideal VM, vm_{id}^{+}. Similarly, a VM with worst performance is

chosen as a negative ideal VM, vm_{id}^{-}. It is formulated as:

$$vm_{id}^{+} = \{min(nd\_vm(i,1)), min(nd\_vm(i,2))\} = \{en^{+}, co^{+}\}, \tag{3.2}$$

$$vm_{id}^{-} = \{max(nd\_vm(i,1)), max(nd\_vm(i,2))\} = \{en^{-}, co^{-}\}. \tag{3.3}$$

$Step\ 4:$ The Euclidean distance between the potential VM and ideal VMs (vm_{id}^{+}, vm_{id}^{-}) is computed and its calculation is shown in Equation 3.4 and Equation 3.6 respectively.

$$\beta_{vm}^{+} = [(nd\_vm(i,1) - en^{+})^2 + (nd\_vm(i,2) - co^{+})^2]^{1/2} \tag{3.4}$$

$$\beta_{vm}^{-} = [(nd\_vm(i,1) - en^{-})^2 + (nd\_vm(i,2) - co^{-})^2]^{1/2} \tag{3.5}$$

$Step\ 5:$ The overall score of a VM is computed as:

$$vm_{i}^{score} = \frac{\beta_{vm}^{+}}{\beta_{vm}^{+} + \beta_{vm}^{-}}. \tag{3.6}$$

Algorithm 3.1 : Sched_tsk (ft_i^j, tk_i^{dl})

Input: finish time ft_i^j, task deadline tk_i^{dl}
Output: Decision on deadline miss

1: **if** $ft_i^j \leq tk_i^{dl}$ **then**
2: return $True$;
3: **else**
4: return $False$;
5: **end if**

Algorithm 3.2 shows the process of the ECA scheduling algorithm. The working of the ECA algorithm is discussed below. First, the score of a VM to execute a task is calculated. Then VMs are sorted using the scoring value (lines 3 - 6). The top $y\%$ of VMs are selected and checked if the deadline constraint is met or not with the help of Algorithm 3.1. If the condition is true, the VM with the earliest finish time is selected; otherwise, the next $y\%$ is selected. This process continues until the sorted VM list is $Null$ (lines 7 - 23). If tk_i can not be allocated, that means the available VMs cannot satisfy task requirements, so add new VMs (lines 24 - 26). If tk_i can be allocated, then the VM with the earliest finish time (i.e., $v(tk_i)$) is selected to execute the task. The matched pair $\langle tk_i, v(tk_i)\rangle$ is stored in $\mathcal{SCH}$. Then, tk_i is scheduled and executed at $v(tk_i)$ (lines 28 - 32).

Theorem 3.1. *The time complexity of the ECA algorithm is $O(nm + nlogn)$, where n is the number of tasks, and m is the number of VMs.*

Proof. The time complexity to compute a VM's score for a task is $O(m)$. Hence for n task it is $O(nm)$. Sorting of VM based on its score takes $O(mlogm)$ time. The time complexity to find an appropriate VM (i.e., VM with earliest finish time) is $O(nm)$. For other lines, the time complexity is $O(1)$. So, the time complexity of ECA algorithm is $O(nm) + O(mlogm) + O(nm) = O(nm + mlogm)$.

Algorithm 3.2 : Energy and Cost Aware (ECA) Algorithm

Input: Task set $\mathcal{T}$, VM set $\mathcal{V}$
Output: Scheduled pair $\mathcal{SCH}$

1: $vm\_flag \longleftarrow False$, $vm\_alloc \longleftarrow Null$, $\mathcal{SCH} \longleftarrow Null$;
2: **for** each task tk_i in $\mathcal{T}$ **do**
3: **for** each VM vm_j in $\mathcal{V}$ **do**
4: Compute score of vm_j by Equation 3.6 ;
5: **end for**
6: $sort\_vm\_score \longleftarrow sort(vm\_score)$; // vm_score is matrix containing score of a VM
7: **while** $sort\_vm\_score \neq Null$ **do**
8: $eft \longleftarrow \infty$;
9: $vm\_sel \longleftarrow$ top $y\%$ of VM in $sort\_vm\_score$;
10: **for** each vm_j in vm_sel **do**
11: Compute ft_i^j;
12: **if** Sched_tsk $(ft_i^j, tk_i^{dl}) == True$ **then**
13: $vm\_flag \longleftarrow True$;
14: **if** $ft_i^j < eft$ **then**
15: $eft \longleftarrow ft_i^j$;
16: $vm\_alloc \longleftarrow vm_j$;
17: **end if**
18: **end if**
19: **end for**
20: **if** $vm\_flag == False$ **then**
21: $vm\_sel \longleftarrow$ next top $y\%$ of VM in $sort\_vm\_score$;
22: **end if**
23: **end while**
24: **if** $vm\_flag == False$ **then**
25: Add new VMs;
26: goto Step 3;
27: **end if**
28: **if** $vm\_flag == True$ **then**
29: $v(tk_i) \longleftarrow vm\_alloc$;
30: **end if**
31: $\mathcal{SCH} \longleftarrow < tk_i, v(tk_i) >$;
32: Assign task tk_i to $v(tk_i)$;
33: **end for**

□

Lemma 3.1. *The execution cost for executing n tasks in the given cloud system having the m number of VMs is a function of the execution time of tasks on VMs is*

$$\sum_{j=1}^{m}\sum_{i=1}^{n} ec_i^j = \sum_{j=1}^{m} vm_j^{ec} \sum_{i=1}^{n} et_i^j \times \mathcal{X}_i^j. \tag{3.7}$$

Proof. Let et_l^j is the execution time of tk_l on vm_j, Φ_l is the time gap between finish time of tk_l and start time of task tk_{l+1} on vm_j. Let, tk_l is the task executed by vm_j just before the

arrival of task tk_{l+1} on vm_j. Then the number of times vm_j is used ($\mathcal{N}_j$) during scheduling process can be represented as: $(et_1^j + \Phi_1, et_2^j + \Phi_2, ...)$. So,

$$\mathcal{N}_j = \sum (et_i^j + \Phi_j) \times \mathcal{X}_i^j. \tag{3.8}$$

Assuming Φ_j has negligible value, Equation 3.8 can be rewritten as:

$$\mathcal{N}_j = \sum (et_i^j) \times \mathcal{X}_i^j. \tag{3.9}$$

So, the total execution cost of tasks on vm_j can be computed as:

$$ec_i^j = vm_j^{ec} \times \sum (et_i^j) \times \mathcal{X}_i^j. \tag{3.10}$$

This is true for all the VMs in the cloud system. So, the total execution cost of n tasks on m VMs can be estimated as:

$$\sum_{j=1}^{m} \sum_{i=1}^{n} ec_i^j = \sum_{j=1}^{m} vm_j^{ec} \sum_{i=1}^{n} et_i^j \times \mathcal{X}_i^j. \tag{3.11}$$

From the Equation 3.11 it can be inferred that the execution cost of a VM mostly depends on how much time it is being used, i.e., the execution time of tasks on it. □

3.3 Performance Evaluation

The simulation approach is preferred as it facilitates the cloud environment for different scenarios [67]. To show the effect of the ECA algorithm on system performance, it is compared with EneRgy-Efficient reaCTive (ERECT) [54], and Energy and Deadline Aware with Non-Migration Scheduling (EDA-NMS) [38] algorithms. The two algorithms used for comparisons are summarized below.

- ERECT: A task is assigned to an energy efficient VM and operate in optimal frequency for energy conservation. But, execution cost is not considered for scheduling a task.
- EDA-NMS: This algorithm is designed to reduce energy consumption while improving tasks' completion time without VM migration. Here, a task is selected based on its criticality (i.e., cost of failure). A VM with sufficient capacity is then selected to complete the task before its deadline, and for energy conservation and operation cost minimization. They have considered different workload types but presented limited experimental work (e.g., performance analysis based on task arrival rate, deadline variation, etc. are missing).

The metrics used to evaluate the performance of algorithms are Success Ratio (SR), total energy consumption (δ), and makespan (τ).

Table 3.1: Parameters for Simulation Studies

Parameter	Value (Fixed) -(Varied)
Task Count	$1000 - (200, 400, 600, 800, 1000)$
VM Count	$80 - (40, 60, 80, 100)$
ts_{var}	$1500 - (500, 1000, 1500, 2000, 2500)$
sp_{var}	$400 - (200, 400, 600, 800, 1000)$
task arrival rate (λ)	0.05 - $(0.01, 0.03, 0.05, 0.08, 0.1)$
$baseD$	4 - $(2, 4, 6, 8)$

3.3.1 Simulation Setting

The detailed settings and parameters for simulation setup are given as follows.

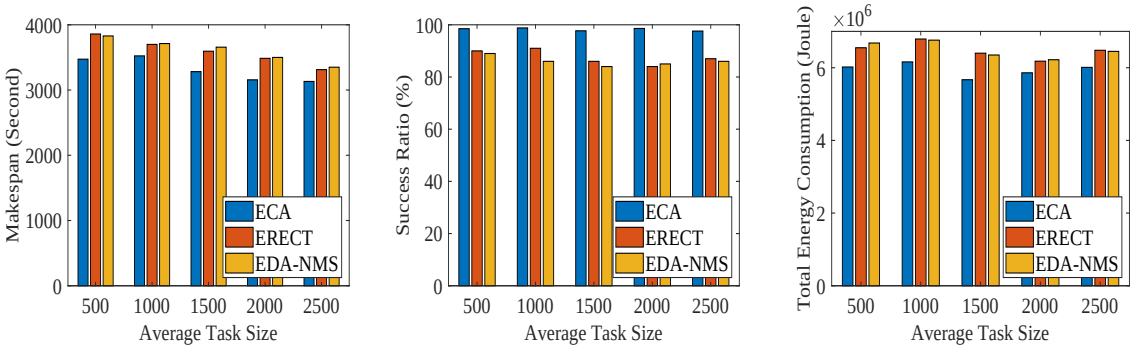
- Task deadline is set as $tk_i^{dl} = tk_i^{ar} + baseD$ where $baseD$ is in Uniform distribution $U(4, 8)$.
- Let task size represents the task heterogeneity. It is uniformly distributed in the range $[ts_{avg} - ts_{var}]$ and $[ts_{avg} + ts_{var}]$ where ts_{avg} is average task size and ts_{var} is variable scope taking $ts_{avg} = 5000$ MI as center respectively.
- Let VM speed indicates VM heterogeneity. It is uniformly distributed in the range $[sp_{avg} - sp_{var}]$ and $[sp_{avg} + sp_{var}]$ where sp_{avg} is average VM speed and sp_{var} is variable scope taking $sp_{avg} = 1500$ MIPS as center respectively. MIPS and MI are assumed based on the study found in [125] and [40] respectively.
- The execution cost vm_j^{ec} of VM vm_j is generated using Uniform distribution $U(0.14, 4.5)\$$ per time unit as referred in [43].

The values of parameters are listed in Table 3.1. The development of IT infrastructure and applications make task and VM heterogeneity obvious in the cloud system. In this context, a set of experiments was conducted to study the effect of task and VM heterogeneity on a cloud system's performance.

3.3.2 Impact of Task Heterogeneity on System Performance

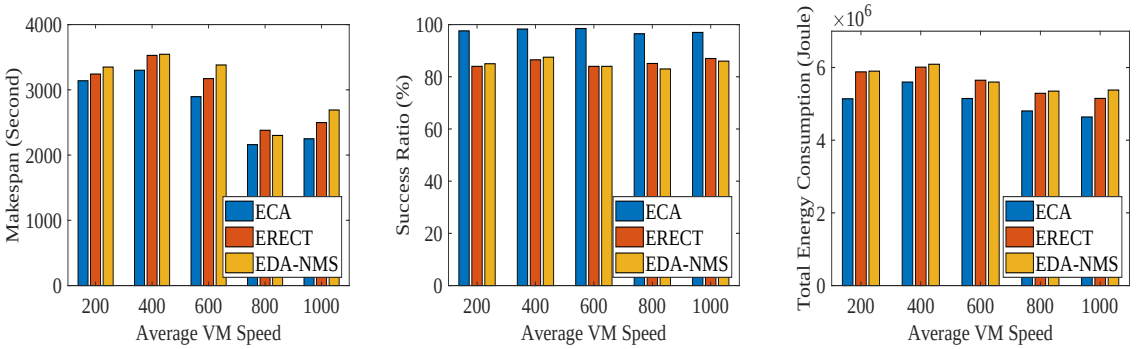
Figure 3.3 shows the effect of task heterogeneity on system performance. The task set with task size near the minimum limit results in short task execution time. Whereas if the task set contains tasks with task size near the maximum limit, there is long task execution time. The short execution time and long execution time due to heterogeneous task set contribute to the makespan. So the makespan, as shown in Figure 3.3a is neither increasing nor decreasing as it depends on the task heterogeneity. Further, the short execution time allows many tasks to execute before the deadline, but the long execution time increases the chance of deadline miss for tasks. Hence the task heterogeneity induces a stable Success Ratio with constant

VM count, as given in Figure 3.3b. The execution of a large number of tasks due to short execution time contributes to total energy consumption. Besides, the long task execution time increases VM's active time and, hence, the total energy consumption. So the simulation outcome is as shown in Figure 3.3c for total energy consumption. However, the simulation results confirm that VM selection, with the help of scoring function, helps ECA perform better than ERECT and EDA-NMS.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 3.3: Impact of Task Heterogeneity on System Performance



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 3.4: Impact of VM Heterogeneity on System Performance

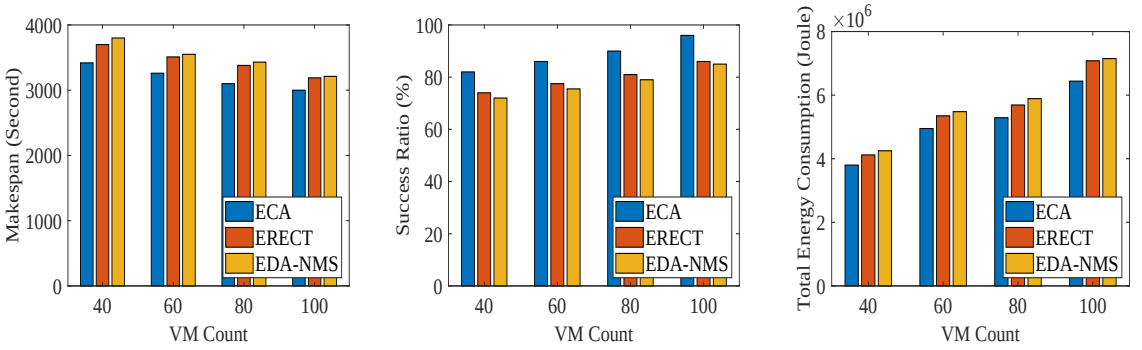
3.3.3 Impact of VM Heterogeneity on System Performance

The simulation results obtained by varying VM speed is depicted in Figure 3.4. A VM set with high-speed VMs produces short task execution time, and if the VM set contains low-speed VMs, then there is long task execution time. The short execution time causes reduced makespan, but the long task execution time results in higher makespan. Hence the simulation outcome in Figure 3.4a for makespan due to heterogeneous VM set has variations, but nature is neither increasing nor decreasing for parametric value. The Success Ratio increases for short execution time and decreases for long execution time. This is because the short execution time allows a large number of task execution before the

deadline, but the long execution time cause deadline misses for tasks. The stable Success Ratio, as shown in Figure 3.4b, confirms the effect of VM heterogeneity on it. The presence of high-speed VM consumes more energy, whereas low-speed VM has less energy consumption. Therefore small variation in total energy consumption, as shown in Figure 3.4c, is due to the heterogeneous VM set. Moreover, the outcome shows that the VM scoring method helps ECA perform better than ERECT and EDA-NMS.

3.3.4 Impact of VM Count on System Performance

Figure 3.5 shows the effect of VM count on system performance. An addition in VM count reduces the total execution time as many tasks get executed in different VMs simultaneously. Hence the simulation result shown in Figure 3.5a for makespan decreases with an increase in VM count. Further, the large VM set allows many tasks to complete execution before the deadline causing a rise in Success Ratio as given in Figure 3.5b. The successful execution of many tasks due to an increase in VM count results in a higher active time of VM. Further, idle VM's presence due to a large VM set contributes to the idle energy consumption. Hence the total energy consumption, as shown in Figure 3.5c, is increasing. The simulation results show that the use of scoring value for VM selection for a task helps ECA perform better than others.



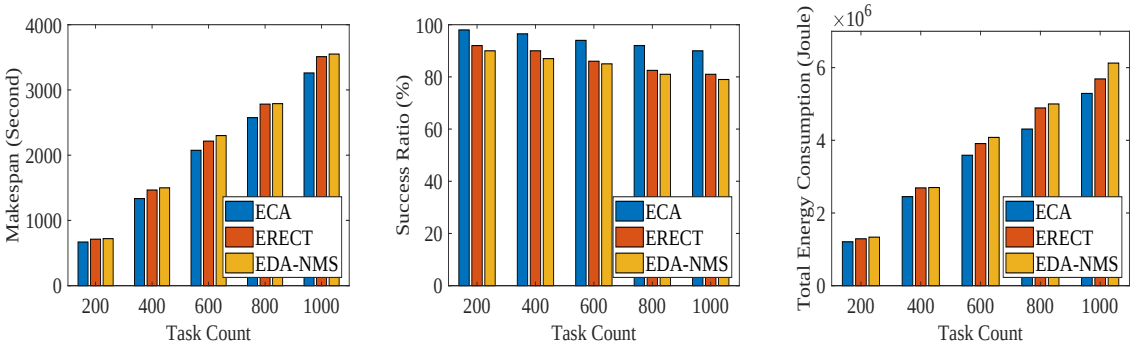
(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 3.5: Impact of VM Count on System Performance

3.3.5 Impact of Task Count on System Performance

Figure 3.6 shows the effect of task count on system performance. An increase in task count with fixed VM count increases the total execution time, which increases the makespan as given in Figure 3.6a. As the task count increases, there are chances that some tasks will not be executed before the deadline. Hence the simulation outcome as given in Figure 3.6b for Success Ratio decreases with an increase in task count. Further, the addition in task count with constant VM count increases VM's active time, which in turn contributes to higher

total energy consumption as given in Figure 3.6c. Figure 3.6, it can be observed that the VM scoring function helps ECA perform better than ERECT and EDA-NMS.

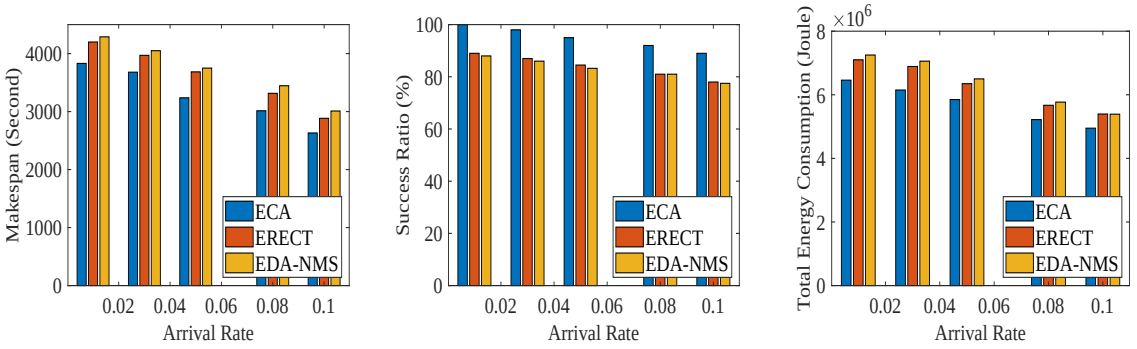


(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 3.6: Impact of Task Count on System Performance

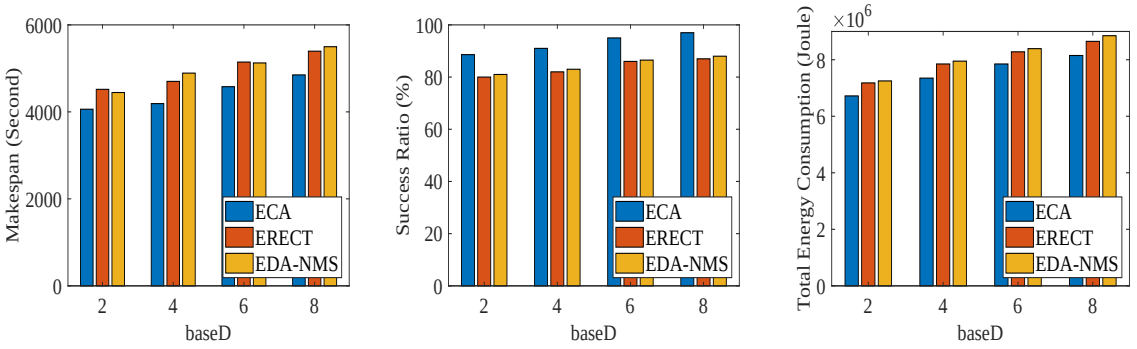
3.3.6 Impact of Arrival Rate on System Performance

The impact of task arrival rate on system performance is shown in Figure 3.7. The increase in task arrival rate causes a large number of tasks in a short time. Thus, fewer tasks complete their execution, causing a decrease in makespan, as shown in Figure 3.7a. Moreover, the large number of tasks in a short time interval causes fewer tasks to complete their execution before the deadline, affecting the Success Ratio. Hence, there is a decrease in the Success Ratio, as given in Figure 3.7b. Furthermore, the less number of task execution causes less VM active time. Hence a decrease in total energy consumption, as shown in Figure 3.7c. However, the VM selection, with the help of scoring value, helps ECA to perform better compared with ERECT and EDA-NMS.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 3.7: Impact of Arrival Rate on System Performance



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 3.8: Impact of Deadline Variation on System Performance

3.3.7 Impact of Deadline Variation on System Performance

This subsection discusses the impact of deadline variation on system performance. Figure 3.8 shows the effect of $baseD$ on system performance. As the $baseD$ increases, a large set of tasks is executed before the deadline, increasing the total execution time. Hence the simulation outcome as shown in Figure 3.8a for makespan increases with an increase in $baseD$. Further, a task gets more time to finish before its deadline due to a rise in $baseD$. So, the Success Ratio increases as given in Figure 3.8b. Besides, VM's active time increases due to the execution of many tasks with an increase of $baseD$. So, there is an increase in total energy consumption, as shown in Figure 3.8c. Moreover, the use of a scoring function to select an appropriate $\langle task, VM \rangle$ pair, helps ECA perform better compared with ERECT and EDA-NMS.

3.4 Summary

This Chapter discusses a VM scoring based scheduling approach to minimize energy consumption and execution cost. It is based on the TOPSIS analysis method. Here, a VM is assigned a score value for a task, and VM with the best score value is selected for that task execution. ECA's performance improvement over ERECT for the Success Ratio is 11.8%, for makespan is 8.3%, and for total energy consumption is 8.1%. Similarly, ECA has an improvement of 12.5% in Success Ratio, 9.7% in makespan and 9.7% in total energy consumption.

In the next Chapter, a Learning Automata (LA)-based scheduling algorithm is proposed for real-time tasks to minimize energy consumption and execution cost.

Chapter 4

Learning Automata-based Scheduling Algorithm

Abstract:- A cloud data center consisting of thousands of computing resources consumes a lot of energy. Usually, cloud users anticipate the completion of their tasks as early as possible. A reinforcement-based method helps to find the best decision by gradually learning the best action from the working environment. This Chapter discusses the reinforcement-based learning method known as Learning Automata (LA) in detail, followed by the proposed LA model and LA-based scheduling framework. The proposed Learning Automata-based Scheduling (LAS) used to find the best ⟨ task, VM ⟩ pair that reduces energy consumption and makespan simultaneously in a cloud system.

4.1 Introduction

Cloud computing is a revolutionary paradigm, which uses data centers to provide services over the Internet. These data centers consume large amounts of energy, whereas a cloud user demands their tasks to be completed as soon as possible. Makespan is defined as the maximum completion time or as a finishing time of the latest task in the system. Usually, less makespan denotes better efficiency of an algorithm. For real-time applications, the finishing time should not exceed its temporal deadline requirements. The reason for the deadline miss may be due to prolonged waiting time or insufficient resource allocation. It is necessary to employ some means to manage multiple and conflicting objectives while minimizing the overall deadline miss event. In this context, reinforcement learning based techniques like Learning Automata (LA) is used to generate the best schedule, where the system gradually learns to perform good actions in response to different environment status. LA's primary goal is to select the optimal action from a set of possible actions, similar to it, a real-time task scheduling problem in the cloud corresponds to choose the best mapping of the task to VM from a possible mapping set. Further, the reinforcement-based learning scheme with very little historical information adds LA's suitability for solving the real-time task scheduling problem. The LA is best suitable to design real-time task scheduling algorithm due to the following feature:

- A reinforcement-based learning scheme penalizes the wrong selection and rewards the good selection. This learning process helps to select an appropriate VM for a real-time task.
- The responses from the environment is modeled in the action probability matrix. This history information is used to make the best decision possible at any instant of time.
- The automaton decides without performing any high time-consuming calculations, making it suitable for the real-time system.

4.1.1 Learning Automata

Learning automata acts as an adaptive decision-making unit that learns to select the best action from a set of allowed actions through repeated interaction with an uncertain environment [80]-[88]. The environment generates a reinforcement signal for each action. Based on the signal value, the probability distribution over the action set is updated. This process continues until a stopping criterion (i.e., maximum iteration count or probability value attained a threshold limit) is reached. The relationship between the environment and learning automata is shown in Figure 4.1. LA can be classified either as a fixed structure or variable structure. In Fixed structure LA (FLA), the number of actions is constant over time, but in Variable structure LA (VLA), the number of actions available changes with time. As in the cloud, the number of service requests changes over time; a VLA will be appropriate to model the cloud system for tasks with a deadline.

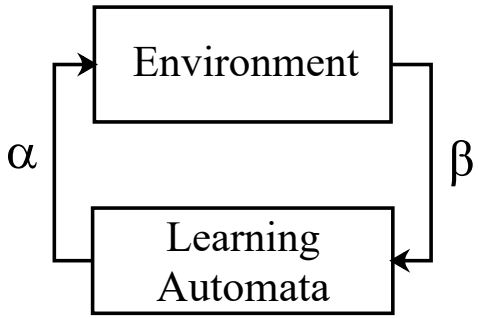


Figure 4.1: Relationship Between Learning Automata and its Environment

A VLA can be defined by quadruple $\langle P, \alpha, \beta, \mathcal{L} \rangle$, where α is the set of actions, β is the set of inputs (or reinforcement signals), P is the set of action probabilities, and $\mathcal{L}$ is the learning or reinforcement algorithm. The learning algorithm is a recurrence relation adopted to revise the action probability vector. Let $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ and $\beta = \{\beta_1, \beta_2, ..., \beta_r\}$, where r is the number of actions possible by an automaton. Let, $\alpha_i(k_1) \in \alpha$ is the action taken by automaton A_i and $p_i(k_1) \in P$ is the probability value for action $\alpha_i(k_1)$ at iteration k_1. The learning algorithm uses reward (ϕ) and penalty (φ) constants to update the action probabilities. The action probability value is updated using the Equation 4.1 and Equation

4.2.

$$p_j(k_1+1) = \begin{cases} p_j(k_1) + \phi \times (1 - p_j(k_1)) & j = i, \\ (1-\phi) \times p_j(k_1) & \forall j, j \neq i. \end{cases} \tag{4.1}$$

$$p_j(k_1+1) = \begin{cases} (1-\varphi) \times p_j(k_1) & j = i, \\ \frac{\varphi}{r-1} + (1-\varphi) \times p_j(k_1) & \forall j, j \neq i. \end{cases} \tag{4.2}$$

Equation 4.1 is used to reward action $\alpha_i(k_1)$ if it is favorable to the environment and reinforcement signal β_i is set to 0. First part of Equation 4.1 says that increase the probability of $\alpha_i(k_1)$ for next iteration if it is favorable whereas decrease the probability of other actions using second part. Similarly Equation 4.2 is used to penalize action $\alpha_i(k_1)$ if it is unfavorable to the environment and β_i is set to 1. Equation 4.2 states that decrease the probability of $\alpha_i(k_1)$ (first part) while increasing the probability of other actions (second part) for next iteration. Based on the values of ϕ and φ learning algorithm can be classified as follows: if $\phi = \varphi$, then it is called linear reward penalty (L_{R-P}) algorithm, if $\phi \gg \varphi$ it is reward-ϵ-penalty ($L_{R\epsilon P}$), and if $\varphi = 0$, it is called reward-inaction (L_{R-I}) algorithm.

4.2 Learning Automata-based Scheduling (LAS) Framework

The scheduler's primary purpose is to map a task set to a VM set from the VM pool, such that energy consumption and makespan is minimized. The scheduling framework is shown in Figure 4.2. It consists of the *LA Model* and *LA Table*. *LA Model* is employed to generate the best scheduling decision possible through the reinforcement learning process and is executed for a fixed number of iteration. The outcome of each iteration is stored in the *LA Table*. In each iteration, the scheduler selects a VM based on its goodness value. The scheduler uses the outcome of the *LA Model* after the final iteration for actual execution on the cloud system.

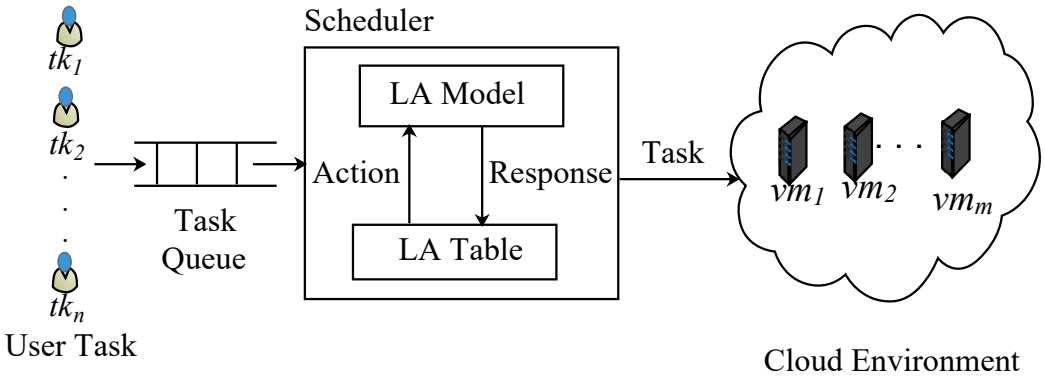


Figure 4.2: LA-based Real-time Task Scheduling Framework

4.2.1 Learning Automata Model

The proposed VLA model is shown in Figure 4.3. It is assumed that each incoming task is associated with an automaton. The VM pool realizes the random environment. The VLA is represented by six-tuple $\langle tk_i, A_i, \alpha_i, p_i, \beta_i, \mathcal{L} \rangle$, where tk_i indicates i^{th} task, A_i is the learning automaton associated with tk_i, α_i is the action set of A_i, p_i is the action probability vector corresponding to α_i, β_i is the reinforcement signal from the environment for action α_i and $\mathcal{L}$ is the learning algorithm. The action set α_i is represented by the probable set of VMs assignment for a task. Hence, $\alpha_i = \{\alpha_i^j | 1 \leq j \leq m\}$ is the action set of A_i, where α_i^1 means task tk_i is assigned to VM vm_1, α_i^2 means tk_i is assigned to vm_2 and so on. Similarly, action probability vector $p_i = \{p_i^j | \alpha_i^j \in \alpha_i\}$, where p_i^1 is the probability value of action α_i^1, p_i^2 is the probability of action α_i^2 and so on. At any instant, action with highest probability is chosen, i.e, $\alpha_i = \{\alpha_i^j | p_i^j = max(p_i)\}$. The reinforcement signal β_i can have binary values 0 and 1. Let, $\omega(k_1)$ represents value of the bi-objective minimization equation (Equation 4.5) at iteration k_1.

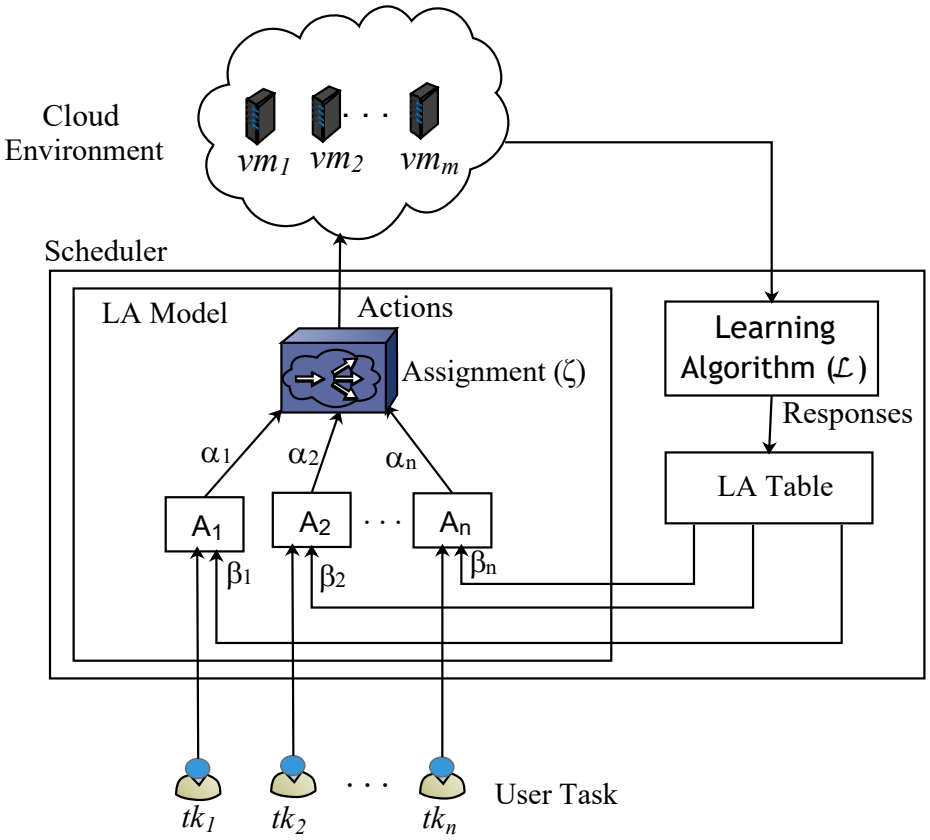


Figure 4.3: Proposed VLA Model

The learning algorithm $\mathcal{L}$ is defined as follows: If $\omega(k_1) \leq \omega(k_1 - 1)$, then check whether action α_i taken by automaton A_i meet the deadline constraint or not. If α_i completes within deadline, set $\beta_i = 0$, otherwise set $\beta_i = 1$. This process avoids the possibility of misinterpretation of favorable output response from the environment as the favorable input

to a particular automaton. The action probability update equation defined in Equation 4.1 and Equation 4.2 are rewritten for the proposed VLA model as:

$$p_i^j(k_1+1) = \begin{cases} p_i^j(k_1) + \phi \times (1 - p_i^j(k_1)) \;\; j = i, \\ (1-\phi) \times p_i^j(k_1) \;\; \forall j, j \neq i. \end{cases} \tag{4.3}$$

$$p_i^j(k_1+1) = \begin{cases} (1-\varphi) \times p_i^j(k_1) \;\; j = i, \\ \frac{\varphi}{r-1} + (1-\varphi) \times p_i^j(k_1) \;\; \forall j, j \neq i. \end{cases} \tag{4.4}$$

If $\beta_i = 0$, action taken by A_i is rewarded using Equation 4.3 and if $\beta_i = 1$, action of A_i is penalized using Equation 4.4. Let p_i^j indicates the goodness value of a VM, and a higher number for it is preferable.

4.3 Learning Automata-based Scheduling (LAS) Algorithm

Figure 4.4 shows the overall working of the LAS scheduling algorithm. The algorithm generates ETC and P matrix and then selects the action with the highest probability and calculates cost metric value. If the cost metric value of iteration k_1 is less than iteration $k_1 - 1$, check whether the deadline constraint is met or not. If the condition is true, set $\beta_i = 0$, else set $\beta_i = 1$. If $\beta_i = 0$ action taken by A_i is rewarded, whereas the action of A_i is penalized. Accordingly, the action probability value is updated. The above process continues until the maximum iteration count is reached. After that, the task assignment is done based on action with the highest probability.

The steps of the proposed LAS algorithm are discussed below.

- *Initialization:* The probability value for every action $\alpha_i^j \in \alpha_i$ for automaton A_i is set to $\frac{1}{m}$. Let, assignment vector ζ with size $1 \times n$, represents the action chosen by each automaton at a particular iteration. The initial value of ω is set to INF, i.e., $\omega(0) = INF$.

Algorithm 4.1 : Mat_Gen $(\mathcal{T}, \mathcal{V}, k_1)$

Input: Task set $\mathcal{T}$, VM set $\mathcal{V}$, iteration number k_1
Output: Matrices ETC, P

```
for each task tk_i in T do
    for each VM vm_j in V do
        Calculate et_i^j using Equation 2.1;
        Set p_i^j = 1/m;
    end for
end for
```

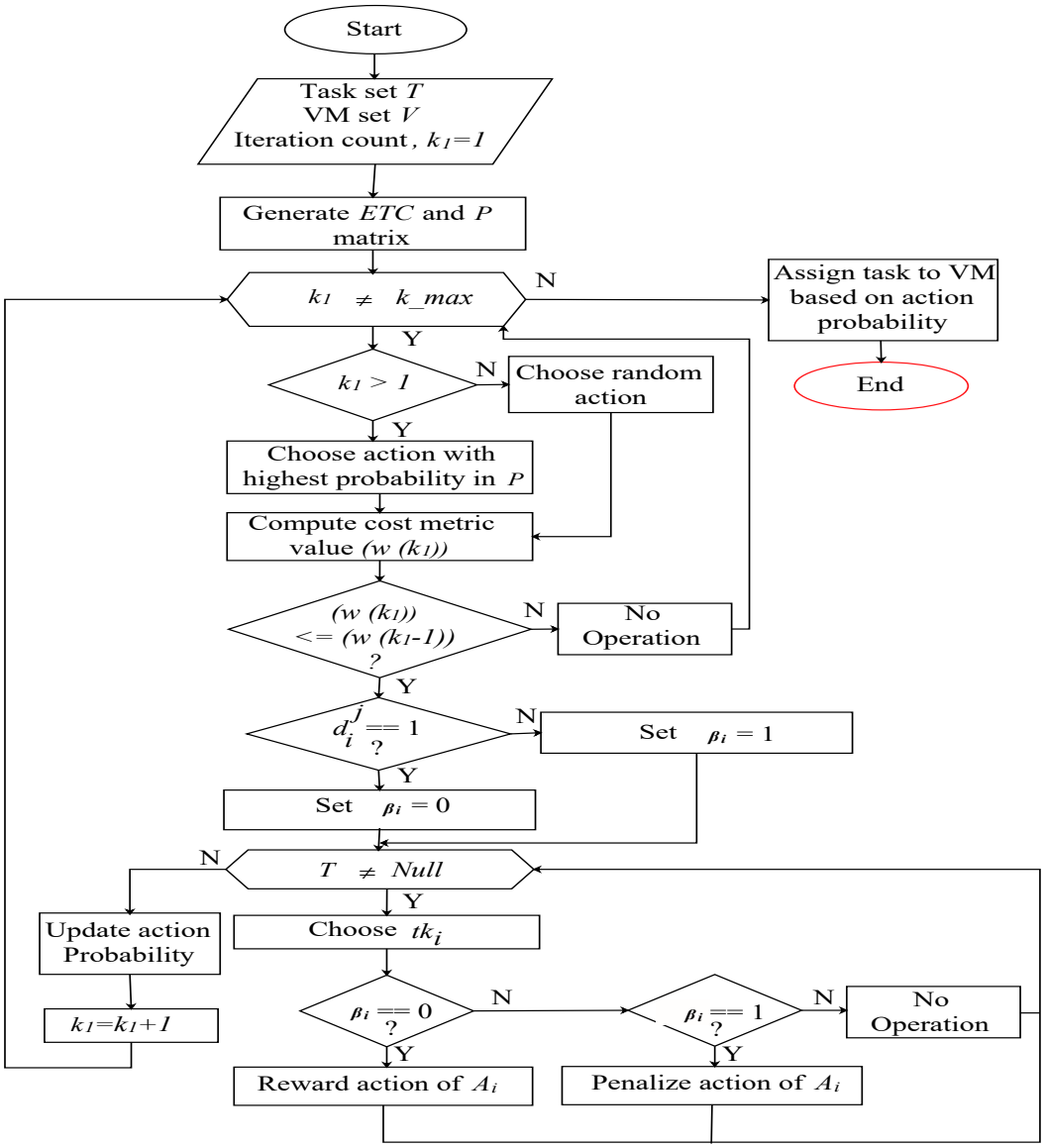


Figure 4.4: Flowchart for LAS

- *Matrix Generation:* Each row of ETC matrix imitates the action set of automaton A_i, $i = \{1, 2, ..., n\}$. For instance, row 1 shows the set of actions (i.e., α_1^j, $j = \{1, 2, ..., m\}$) possible for automaton A_1. Similarly, let p_i^j is contained within $n \times m$ action probability matrix, P. In the first iteration, ζ is formed by randomly selecting an element from each row of the ETC matrix. In subsequent iterations, an element having the highest value in each row of P is marked, and ζ is constructed by selecting the corresponding action in ETC. Pseudo-code to generate ETC and P matrices is shown in Algorithm 4.1.

- *Cost Metric (ω) Calculation:* Let, τ_{max} be the maximum allowable makespan, and δ_{max} be the maximum possible energy consumption of the system. Then, the

Algorithm 4.2 : Cal_Cost_Metric (ζ)

Input: Assignment vector (ζ)
Output: Cost metric (ω)

1: **for** each action $\alpha_i^j \in \zeta$ **do**
2: Compute start time st_i^j and finish time ft_i^j by Equation 2.2 and Equation 2.3 respectively;
3: **if** Sched_tsk $(ft_i^j, tk_i^{dl}) == True$ **then**
4: Set $d_i^j = 1$;
5: Calculate δ and τ using Equation 2.10 and Equation 2.7 respectively.
6: Obtain ω using Equation 4.5;
7: **else**
8: Set $d_i^j = 0$;
9: **end if**
10: **end for**

bi-objective minimization problem can be represented as:

$$\omega = \frac{\delta}{\delta_{max}} \times x + \frac{\tau}{\tau_{max}} \times y, \tag{4.5}$$

where x, and y are weight associated with δ and τ respectively. Since the elements of the proposed bi-objective minimization problem have distinct measurement units, the normalization process is employed to make them comparable ones. The algorithm for

Algorithm 4.3 : Sched_tsk(ft_i^j, tk_i^{dl})

Input: finish time ft_i^j, task deadline tk_i^{dl}
Output: Decision on deadline miss

1: **if** $ft_i^j \leq tk_i^{dl}$ **then**
2: return $True$;
3: **else**
4: return $False$;
5: **end if**

cost metric calculation is shown in Algorithm 4.2. The working of Algorithm 4.2 is as follows: first, the start time and finish time of each action in ζ is computed in line 2. If finish time of task tk_i on vm_j met its deadline (Algorithm 4.3) then set $d_i^j = 1$ and estimate cost metric ω (lines 3 - 6). If condition doesn't hold then set $d_i^j = 0$. This also indicates that there is no VM vm_j, which can execute tk_i within its deadline, so add a new VM.

- *Decision based on Action Probabilities:* Due to the heterogeneity of tasks and VM, the action performed by an automaton can be rewarded sometimes, while other times, the very same action will be penalized. A decision on the action is made based on action probability value using Algorithm 4.4. This process continues until the stopping criterion is not satisfied. Let θ_{mn} and θ_{mx} indicates minimum and maximum value

Algorithm 4.4 : Action_Decision $(\mathcal{T}, \mathcal{V}, k_1)$

Input: $\mathcal{T}, \mathcal{V}, k_1, \theta_{mn}, \theta_{mx}$
Output: Decision on action

```
1: for each task tk_i in T do
2:     for each VM vm_j in V do
3:         if p_i^j(k_1) ≤ θ_mn then
4:             Deactivate α_i^j;
5:         end if
6:         if p_i^j(k_1) == θ_mx then
7:             Assign tk_i to vm_j;
8:         end if
9:     end for
10: end for
```

for action probability p_i^j. This algorithm checks if $p_i^j(k_1)$ of action α_i^j is less than θ_{mn} or not. If condition is true then deactivate α_i^j for automaton A_i (lines 3 - 5). If $p_i^j(k_1) = \theta_{mx}$, all the actions except α_i^j are deactivated and tk_i is assigned to vm_j.

- *Calculation of Constants:* The weight constants x and y are set to 0.5, as equal importance is given to both the metrics. The reward (ϕ) and penalty (φ) parameters are given the same value.

- *Stopping Criteria:* The learning process stops when the number of iteration k_1 exceeds the maximum iteration count k_{max} or p_i of all automaton reaches its threshold value, whichever is earlier.

The overall working of the LAS is shown in Algorithm 4.5. The working of LAS is explained below. First, the ETC and probability matrix P is generated for a given set of VMs and tasks. Then, for each iteration, the assignment vector ζ is generated (lines 4 - 8). For each assignment, the cost metric is calculated using Algorithm 4.2. If the cost metric at iteration k_1 is better than the value at iteration $k_1 - 1$, then check whether task execution is completed within deadline constraint or not. The reinforcement signal for task's meeting the deadline constraint is set to "0", i.e., $\beta_i = 0$, whereas for other tasks $\beta_i = 1$ (lines 12 - 16). An automaton is either rewarded or penalized based on the reinforcement signal (lines 20 - 24). This process continues until the stopping criterion is not satisfied.

Theorem 4.1. *The time complexity of the LAS algorithm is* $O(nm)$.

Proof. The time complexity of generating ETC and P matrix is $O(nm)$ (Algorithm 4.1). Generation of assignment vector is $O(nm)$. In Algorithm 4.1, the time complexity for calculation of task's start time and finish time is $O(nm)$. Checking if a task meet it's deadline or not takes $O(nm)$ time. In Algorithm 4.5, the complexity of reward and penalty calculation is $O(n)$ and for setting reinforcement signal value the time complexity is $O(nm)$. In Algorithm 4.4 the time complexity of making decision on action is $O(nm)$. For other lines, the time complexity is $O(1)$. Hence, the time complexity of LAS algorithm is $O(nm) + k_{max}(O(nm) + O(nm) + O(nm) + O(n) + O(nm) + O(nm)) = O(nm)$. □

Algorithm 4.5 : Learning Automata-based Scheduling (LAS) Algorithm

Input: $\mathcal{T}, \mathcal{V}, k_1, k_{max}$
Output: Scheduled pairs $\langle tk_i, vm_j \rangle$

Set $k_1 = 1$;
Generate ETC matrix and probability matrix P using Mat_Gen $(\mathcal{T}, \mathcal{V}, k_1)$;
while $k_1 \leq k_{max}$ **do**
    if $k_1 == 1$ **then**
        Construct ζ by random selection;
    else
        Construct ζ by choosing actions with highest probabilities in $\mathcal{P}$;
    end if
    Calculate cost metric using Cal_Cost_Metric (ζ);
    if $\omega(k_1) \leq \omega(k_1 - 1)$ **then**
        for each (tk_i, vm_j) pair in the system **do**
            if $d_i^j == 1$ **then**
                Set $\beta_i = 0$;
            else
                Set $\beta_i = 1$;
            end if
        end for
    end if
    for each task tk_i in the system **do**
        if $\beta_i == 0$ **then**
            Reward the action taken by A_i using Equation 4.3;
        else if $\beta_i == 1$ **then**
            Penalize the action chosen by A_i using Equation 4.4;
        end if
    end for
    Make decision on action taken by A_i using Action_Decision $(\mathcal{T}, \mathcal{V}, k_1)$;
    Increment k_1;
end while

4.3.1 Example

The proposed scheduling scheme is explained with an example as shown in Figure 4.5. Let, there are 5 tasks ($n = 5$) and 3 numbers of VMs ($m = 3$) in the cloud system. An automaton $A_i,\ i = \{1, 2, 3, 4, 5\}$ can choose three actions $\alpha_i^j,\ j = \{1, 2, 3\}$. Let, $k_{max} = 3$.

- *Iteration 1:* Initial configuration of matrices P and ζ are shown in Figure 4.5(a). Let, the action set $\alpha = \{\alpha_1^2, \alpha_2^3, \alpha_3^1, \alpha_4^2, \alpha_5^2\}$. Let, actions α_1^2, α_2^3, and α_4^2 meet the deadline constraint of respective tasks. But, actions α_3^1 and α_5^2 fail to do so. Let, the calculated cost metric value, $\omega(1) = 20$. As $\omega(1) \leq \omega(0)$, the reinforcement signal for each action is $\{\beta_1 = 0, \beta_2 = 0, \beta_3 = 1, \beta_4 = 0, \beta_5 = 1\}$. The actions α_1^2, α_2^3, and α_4^2 are

$P =$

Task \VM	vm_1	vm_2	vm_3
tk_1	0.33	0.33	0.33
tk_2	0.33	0.33	0.33
tk_3	0.33	0.33	0.33
tk_4	0.33	0.33	0.33
tk_5	0.33	0.33	0.33

$\zeta =$

1	2	3	4	5
vm_2	vm_3	vm_1	vm_2	vm_2

(a) Initial Configuration

$P =$

Task \VM	vm_1	vm_2	vm_3
tk_1	0.30	0.40	0.30
tk_2	0.30	0.30	0.40
tk_3	0.30	0.35	0.35
tk_4	0.30	0.40	0.30
tk_5	0.35	0.30	0.35

$\zeta =$

1	2	3	4	5
vm_2	vm_3	vm_2	vm_2	vm_1

(b) Matrices after Iteration 1

$P =$

Task \VM	vm_1	vm_2	vm_3
tk_1	0.27	0.46	0.27
tk_2	0.27	0.27	0.46
tk_3	0.27	0.415	0.315
tk_4	0.27	0.46	0.27
tk_5	0.315	0.32	0.365

$\zeta =$

1	2	3	4	5
vm_2	vm_3	vm_2	vm_2	vm_3

(c) Matrices after Iteration 2

$P =$

Task \VM	vm_1	vm_2	vm_3
tk_1	0.243	0.514	0.243
tk_2	0.243	0.243	0.514
tk_3	0.243	0.4735	0.2835
tk_4	0.243	0.514	0.243
tk_5	0.2835	0.288	0.4285

$\zeta =$

1	2	3	4	5
vm_2	vm_3	vm_2	vm_2	vm_3

(d) Final Scheduling Decision (after Iteration 3)

Figure 4.5: An Example for LAS

rewarded. The action probability of A_1 with $\phi = \varphi = 0.1$, is calculated as follows:

$$\begin{cases} p_1^1(2) = (1-\phi) \times p_1^1(1) = 0.9 \times 0.33 = 0.297 \simeq 0.3, \\ p_1^2(2) = 0.33 + .1 \times 0.67 = 0.397 \simeq 0.4, \\ p_1^3(2) = (1-\phi) \times p_1^3(1) = 0.3. \end{cases} \tag{4.6}$$

Similarly, probability value for A_2 and A_4 are computed. The action probability of A_3 is updated as follows:

$$\begin{cases} p_3^1(2) = (1-\varphi) \times p_3^1(1) = 0.9 \times 0.33 = 0.297 \simeq 0.3, \\ p_3^2(2) = \frac{0.1}{2} + .9 \times p_3^2(1) = 0.347 \simeq 0.35, \\ p_3^3(2) = 0.35. \end{cases} \tag{4.7}$$

Likewise, the probability of A_5 is computed. The updated values are shown in Figure

4.5(b).

- *Iteration 2:* Let, the action set $\alpha = \{\alpha_1^2, \alpha_2^3, \alpha_3^2, \alpha_4^2, \alpha_5^1\}$. Let, actions α_1^2, α_2^3, α_3^2 and α_4^2 meet the deadline constraint of respective tasks. But, action α_5^1 fail to do so. Let, the calculated cost metric value, $\omega(2) = 18$. As $\omega(2) \leq \omega(1)$, the reinforcement signal for each action is $\{\beta_1 = 0, \beta_2 = 0, \beta_3 = 0, \beta_4 = 0, \beta_5 = 1\}$. The action with the reward is updated as follows:

$$\begin{cases} p_1^1(3) = (1-\phi) \times p_1^1(2) = 0.27, \\ p_1^2(3) = 0.4 + .1 \times 0.6 = 0.46, \\ p_1^3(3) = 0.27. \end{cases} \tag{4.8}$$

  Similarly, probability value for A_2, A_3 and A_4 are computed. The action probability of A_3 is updated as follows:

$$\begin{cases} p_5^1(3) = 0.9 \times 0.35 = 0.315, \\ p_5^2(3) = \frac{0.1}{2} + .9 \times p_3^2(2) = 0.32, \\ p_5^3(3) = \frac{0.1}{2} + .9 \times 0.35 = 0.365. \end{cases} \tag{4.9}$$

  The updated values are shown in Figure 4.5(c).

- *Iteration 3:* Let, actions $\alpha = \{\alpha_1^2, \alpha_2^3, \alpha_3^2, \alpha_4^2, \alpha_5^3\}$ meet both cost metric and deadline constraint. The reinforcement signal for each action is "0", i.e., $\{\beta_1 = 0, \beta_2 = 0, \beta_3 = 0, \beta_4 = 0, \beta_5 = 0\}$. The updated P matrix is shown in Figure 4.5(d). Hence, the final scheduling decision is $\zeta = \{\alpha_1^2, \alpha_2^3, \alpha_3^2, \alpha_4^2, \alpha_5^3\}$.

Lemma 4.1. *To preserve the law of probability measure, the sum of probabilities of actions possible by an automaton A_i is one. Mathematically, it is expressed as:*

$$\sum_{j=1}^{m} p_i^j = 1, \tag{4.10}$$

where m is the number of actions possible by A_i.

Proof. Let, A_1 is the automaton for task tk_1 and can choose five actions $\alpha_1^1, \alpha_1^2, \alpha_1^3, \alpha_1^4, \alpha_1^5$ with probabilities $p_1^1, p_1^2, p_1^3, p_1^4, p_1^5$ respectively. α_1^1 means task tk_1 is assigned to VM vm_1, α_1^2 means task tk_1 is assigned to VM vm_2 and so on. p_1^1 is the probability of choosing action α_1^1, p_1^2 is the probability of choosing action α_1^2 and so on. Let, task tk_1 is assigned to VM vm_1. If the action is rewarded, then the action probability is updated according to Equations 4.11, and 4.12.

$$p_1^1(k_1 + 1) = p_1^1(k_1) + \phi \times (1 - p_1^1(k_1)) \tag{4.11}$$

$$p_1^{2,3,4,5}(k_1 + 1) = (1 - \phi) \times p_1^{2,3,4,5}(k_1) \tag{4.12}$$

$p_1^{2,3,4,5}$ is used instead of individual probability $p_1^2, p_1^3, p_1^4, p_1^5$. Similarly, Equations 4.13, and

4.14 are used to update action probability for penalty.

$$p_1^1(k_1+1) = p_1^1(k_1) - \varphi \times (1 - p_1^1(k_1)) \tag{4.13}$$

$$p_1^{2,3,4,5}(k_1+1) = (1-\varphi) \times p_1^{2,3,4,5}(k_1) + (\frac{\varphi}{4}) \tag{4.14}$$

Equation 4.15 is generated by reframing Equation 4.11.

$$p_1^1(k_1+1) = (1-\phi) \times p_1^1(k_1) + \phi \tag{4.15}$$

Equation 4.16 shows the results of different k_1 values in Equation 4.15.

$$\begin{cases} k_1 = 1, \;\; p_1^1(2) = (1-\phi) \times p_1^1(1) + \phi \\ k_1 = 2, \;\; p_1^1(3) = (1-\phi) \times p_1^1(2) + \phi \\ k_1 = 3, \;\; p_1^1(4) = (1-\phi) \times p_1^1(3) + \phi \end{cases} \tag{4.16}$$

The solution of Equation 4.16 with substitution method is written in Equation 4.17.

$$p_1^1(4) = (1-\phi)^3 \times p_1^1(1) + \phi \times [1 + (1-\phi) + (1-\phi)^2] \tag{4.17}$$

Equation 4.18 is formulated by generalizing Equation 4.17.

$$\begin{aligned} p_1^1(k_1+1) &= (1-\phi)^{k_1+1} \times p_1^1(0) + \phi \times [1 + (1-\phi) + ... + (1-\phi)^{k_1}] \\ &= (1-\phi)^{k_1+1} \times p_1^1(0) + \phi \times \left[\frac{1-(1-\phi)^{k_1}}{1-(1-\phi)}\right] \end{aligned} \tag{4.18}$$

For large value of k_1, $(1-\phi)^{k_1}$ and $(1-\phi)^{k_1+1}$ tends to zero. So, the Equation 4.18 is rewritten as:

$$p_1^1(k_1+1) = \phi \times \left[\frac{1}{1-(1-\phi)}\right] = 1. \tag{4.19}$$

The results of different k_1 values in Equation 4.12 is written in Equation 4.20.

$$\begin{cases} k_1 = 1, \;\; p_1^{2,3,4,5}(2) = (1-\phi) \times p_1^{2,3,4,5}(1) \\ k_1 = 2, \;\; p_1^{2,3,4,5}(3) = (1-\phi) \times p_1^{2,3,4,5}(2) \\ k_1 = 3, \;\; p_1^{2,3,4,5}(4) = (1-\phi) \times p_1^{2,3,4,5}(3) \end{cases} \tag{4.20}$$

By solving Equation 4.20 with substitution method Equation 4.21 is generated.

$$p_1^{2,3,4,5}(4) = (1-\phi)^3 \times p_1^{2,3,4,5}(1) \tag{4.21}$$

Equtaion 4.22 is formulated by generalizing Equation 4.21.

$$p_1^{2,3,4,5}(k_1+1) = (1-\phi)^{k_1+1} \times p_1^{2,3,4,5}(1) = 0 \tag{4.22}$$

For large value of k_1, $(1-\phi)^{k_1+1}$ tends to 0. Sum of all probabilities is:

$$\sum_{j=1}^{5} p_1^j = p_1^1(k_1+1) + p_1^{2,3,4,5}(k_1+1) = 1 + 0 = 1. \tag{4.23}$$

Similarly, the penalty Equation 4.13 and Equation 4.14 can be used to prove "sum of all probabilities is one". □

4.4 Performance Evaluation

To show the effectiveness of Learning Automata-based Scheduling (LAS), it is compared with two different algorithms, Multi-Heuristic Resource Allocation (MHRA) [9], and Dynamic Task Scheduling (DTS) [56]. The algorithms for comparisons are summarized as follows:

MHRA: The two-phase MHRA algorithm is designed to minimize a bi-objective cost function that includes energy consumption and total execution time. In the first phase, tasks are ranked based on heuristic rules like the longest processing time first, shortest processing time first, etc. In the next phase, the best cloud resource is selected that minimizes the bi-objective cost function.

DTS: It is a two-step process. At first, tasks are classified according to historical scheduling information. In the next step, the VM of various types is accordingly created. Then mapping tasks with a VM is performed dynamically. This algorithm considers makespan but not total energy consumption.

The metrics used to evaluate the performance of the scheduling algorithms LAS, MHRA, and DTS are the Success Ratio (SR), makespan (τ), and total energy consumption (δ). The values of parameters are listed in Table 4.1.

Table 4.1: Parameters for Simulation Studies

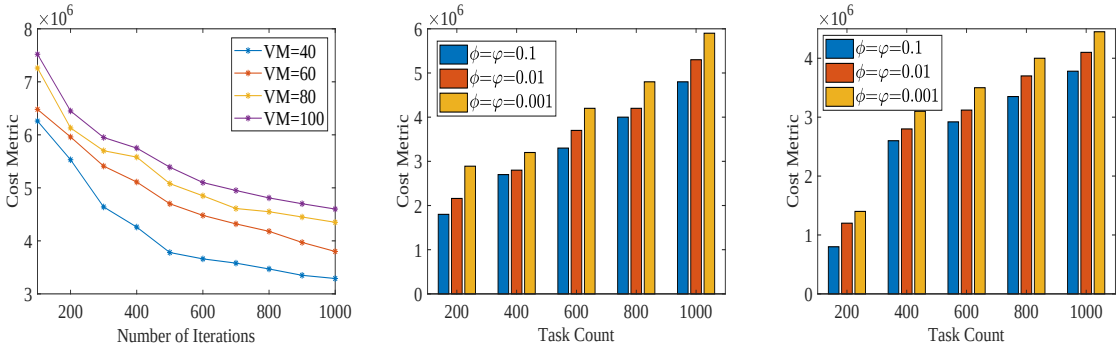
Parameter	**Value (Fixed) -(Varied)**
Task Count	$1000 - (200, 400, 600, 800, 1000)$
VM Count	$80 - (40, 60, 80, 100)$
ts_{var}	$1500 - (500, 1000, 1500, 2000, 2500)$
sp_{var}	$400 - (200, 400, 600, 800, 1000)$
task arrival rate (λ)	0.05 - $(0.01, 0.03, 0.05, 0.08, 0.1)$
$baseD$	4 - $(2, 4, 6, 8)$

4.4.1 Simulation Settings

The detailed settings and parameters for simulation are given as follows:

- Task deadline is set as $tk_i^{dl} = tk_i^{ar} + baseD$ where $baseD$ is in Uniform distribution $U(4, 8)$.
- Let task size represents the task heterogeneity. It is uniformly distributed in the range $[ts_{avg} - ts_{var}]$ and $[ts_{avg} + ts_{var}]$ where ts_{avg} is average task size and ts_{var} is variable scope taking $ts_{avg} = 5000$ MI as center respectively.

- Let VM speed indicates VM heterogeneity. It is uniformly distributed in the range $[sp_{avg} - sp_{var}]$ and $[sp_{avg} + sp_{var}]$ where sp_{avg} is average VM speed and $sp_{avg} = 1500$ is variable scope taking sp_{avg} as center respectively.
- The reward and penalty constant is set to $\phi = \varphi = 0.1$.



(a) Cost Metric (b) Cost Metric (VM Count=40) (c) Cost Metric (VM Count=60)

Figure 4.6: Cost Metric variation

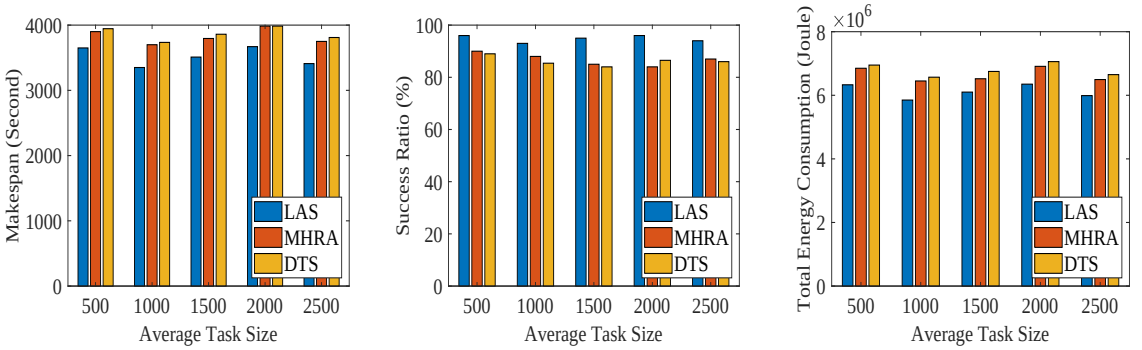
A set of experiments were conducted with initial configuration to examine the effect of the number of iterations on the cost metric. But VM count is varied between $[40 - 100]$ in the step of 20. The outcome of the experiment is shown in Figure 4.6(a), and it was observed that after 500 iterations, there is minimal variation in the cost metric value even if the number of iteration increases significantly. Another experiment was carried out with different values of reward and penalty constant, and setting the maximum iteration count k_{max} to 500. From Figure 4.6(b) and Figure 4.6(c), it can be inferred that the nature of the graph is similar for different values of ϕ and φ (i.e., ϕ, $\varphi = 0.01, 0.001$) even if there is variation in the task count and VM count. Thus the ϕ and φ values are set to 0.1.

The development of IT infrastructure and applications cause heterogeneous task and VM to operate in the cloud system. In this context, another set of experiments was performed to study the effect of task and VM heterogeneity on LAS, MHRA, and DTS. Further, the system performance is analyzed by varying task count, VM count, arrival rate, and deadline.

4.4.2 Impact of Task Heterogeneity on System Performance

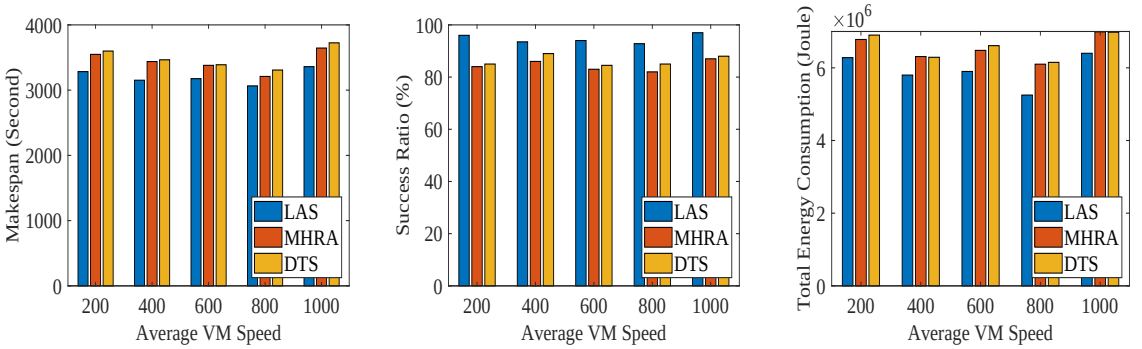
The impact of task heterogeneity on Success Ratio, makespan, and the total energy consumption is given in Figure 4.7. In all three cases, the nature of the simulation result is neither increasing nor decreasing. This is because if the task set contains tasks whose task size is near the minimum limit, the total execution time will be less. Whereas if the task set contains tasks with task size near the maximum limit, the task has a long execution time. The short task execution time decreases the makespan; similarly, the long task execution time gives rise to a higher makespan. So, the task set with task size within the maximum and

minimum limit has both possibilities, which cause a stable makespan, as shown in Figure 4.7a. The short execution time allows many tasks to complete before the deadline and hence increase the Success Ratio. But the Success Ratio decreases for a long task execution time with constant VM count as it increases the chance of deadline miss for a task. Hence there is a little variation in Figure 4.7b for Success Ratio. Consequently, there is an increase in energy consumption due to the rise in the number of tasks getting executed. Further, the high execution time increases VM's active time, causing an increase in total energy consumption. Hence the heterogeneous task set gives the simulation outcome, as shown in Figure 4.7c for total energy consumption. Moreover, it can be seen that the LAS algorithm performs better over MHRA and DTS.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 4.7: Impact of Task Heterogeneity on System Performance



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 4.8: Impact of VM Heterogeneity on System Performance

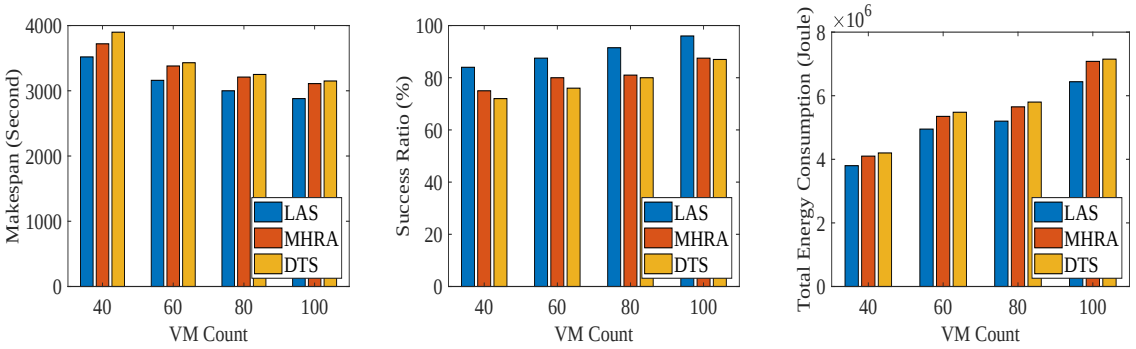
4.4.3 Impact of VM Heterogeneity on System Performance

Figure 4.8 shows the effect of VM heterogeneity on Success Ratio, makespan, and total energy consumption. In all three cases, the nature of the simulation result is neither increasing nor decreasing. This is because the execution time of the task increases if the VM set contains low-speed VMs. Whereas the VM set with high-speed VMs allows many

tasks to execute before the deadline due to short execution time. Thus the makespan, as shown in Figure 4.8a, has little variation due to VM heterogeneity. The short task execution time results in several numbers of tasks getting executed before the deadline. Hence there is a rise in the Success Ratio. However, some tasks miss their deadline due to the long task execution time, which results in a decrease in the Success Ratio. The VM heterogeneity confirms the outcome, as shown in 4.8b for the Success Ratio. The low-speed VM has less energy consumption. Further, the use of high-speed VM contributes to an increase in total energy consumption. Hence the variation in total energy consumption in Figure 4.8c due to VM heterogeneity is less. Moreover, the outcome shows that the learning process from the environment helps LAS perform better than MHRA and DTS.

4.4.4 Impact of VM Count on System Performance

The VM count is varied in the range $[40 - 100]$ in the interval of 20. Figure 4.9 shows the effect of VM count on system performance. The increase in VM count reduces the total execution time as several tasks can be executed simultaneously on various VMs. Hence the makespan increases, and the simulation result is given in Figure 4.9a. It is observed from Figure 4.9b that with an increase in VM count, there is an increase in the Success Ratio. This is because an increase in VM count allows many tasks to execute within the deadline constraint, giving rise to a higher Success Ratio. A large number of task execution due to an addition in VM count increases VM's active time. Besides, there is an increase in idle energy consumption due to idle VM in large VM set. So the total energy consumption increases as given in Figure 4.9c. Besides, it can be found that LA theory helps LAS to improve task schedulability and performs better compared to MHRA and DTS



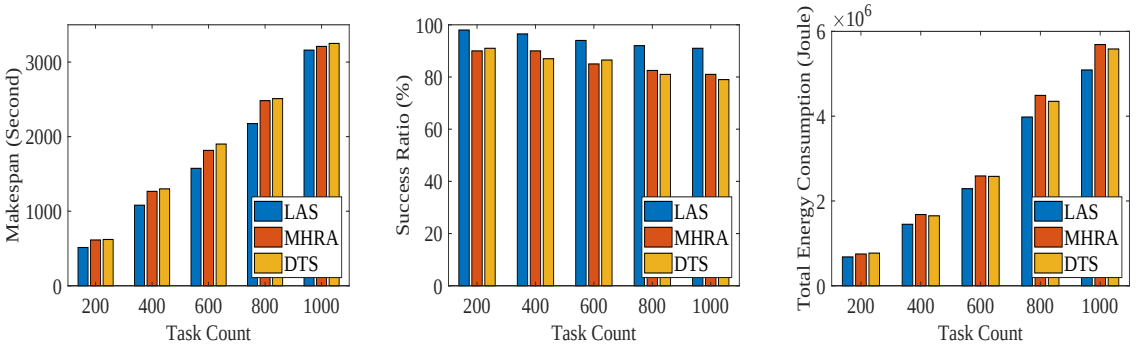
(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 4.9: Impact of VM Count on System Performance

4.4.5 Impact of Task Count on System Performance

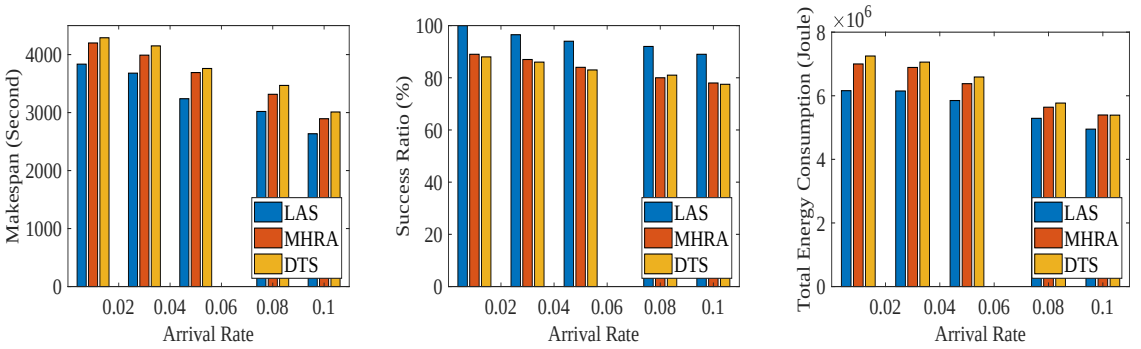
The task count is varied in the range $[200 - 1000]$ in the interval of 200. The impact of task count on system performance is shown in Figure 4.10. The addition in task count increases

the makespan, as given in Figure 4.10a. This is because an increase in task count with constant VM count causes a rise in total execution time. Figure 4.10b shows that as the task count increases, the Success Ratio decreases. The rise in task count increases the chance that some tasks will not be executed before the deadline and hence a low Success Ratio. Further, the increase in task count increases the total energy consumption, as shown in Figure 4.10c. This is because the rise in task count with fixed VM count raises VM's active time and, hence, the total energy consumption. However, VM selection with the learning process helps LAS perform better than MHRA and DTS.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 4.10: Impact of Task Count on System Performance



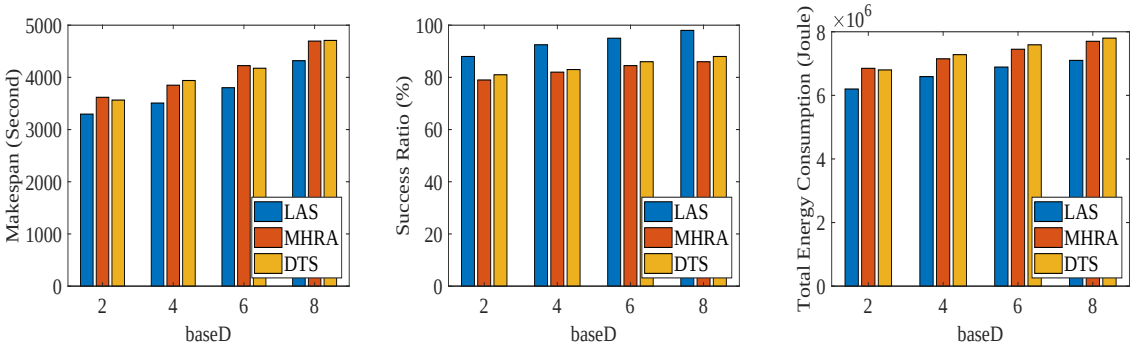
(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 4.11: Impact of Arrival Rate on System Performance

4.4.6 Impact of Arrival Rate on System Performance

Figure 4.12 shows the effect of task arrival rate on system performance. The (λ) value is varied from 0.01 to 0.1 with step 0.02 to analyze the system performance by task arrival rate. There is a large set of tasks in a short time due to the increase in task arrival rate. Figure 4.11a, it can be observed that the large task set in a small time interval allows fewer tasks to get executed, resulting in shorter total execution time and hence a decrease in makespan. Further, the increase in the task arrival rate causes a decrease in the Success Ratio, as given

in Figure 4.11b. This is because fewer tasks completed execution before the deadline due to high task count in a short time. Furthermore, the less number of task execution causes a reduction in total energy consumption, as shown in Figure 4.11c due to less VM active time. Besides, the simulation outcomes confirm that the selection of $\langle task, VM \rangle$ pair with the help of the learning process, helps LAS perform better than MHRA and DTS.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 4.12: Impact of Deadline Variation on System Performance

4.4.7 Impact of Deadline Variation on System Performance

The impact of deadline variation on system performance, as shown in Figure 4.12, is studied by varying the $baseD$ value in the range $[2-8]$. It is observed from Figure 4.12a and Figure 4.12b that the increase in $baseD$ causes an increase in the makespan and Success Ratio. This is because of a large number of tasks execution before the deadline due to an increase of $baseD$. The total energy consumption also increases as given in Figure 4.12c because of the large set of tasks execution raises the active time of VM. However, the learning process from the environment allows LAS to perform better in comparison with MHRA and DTS.

Table 4.2: Comparison with ECA (VM Count)

	ECA	LAS
SR	90	91.5
Makespan	3100	3000
Total Energy Consumption	5.29×10^6	5.20×10^6

4.4.8 Comparison

Table 4.2 to Table 4.5 shows the comparison of LAS with ECA (Chapter 3) algorithm considering different scenarios. The comparison shows that LAS performs better in most of the scenarios but in some cases ECA shows better result.

Table 4.3: Comparison with ECA (Task Count)

	ECA	LAS
SR	90	91
Makespan	3260.4	3160.4
Total Energy Consumption	5.29×10^6	5.09×10^6

Table 4.4: Comparison with ECA (Arrival Rate)

	ECA	LAS
SR	95	94
Makespan	3237.9	3239
Total Energy Consumption	5.85×10^6	5.85×10^6

4.5 Summary

In this Chapter, a bi-objective scheduling algorithm for real-time tasks is proposed. The objective of proposed scheduling is to minimize energy consumption and makespan simultaneously. To realize the goals, an LA-based method that works on the principle of reinforcement learning is employed. First, an LA-based scheduling framework is proposed, and then an enabling LAS scheduling algorithm is presented. The experimental outcomes implied that the learning process from the environment helps LAS achieve better performance than MHRA and DTS. Compared to MHRA, the proposed LAS has an improvement of 11.3% in Success Ratio, 8.6% in makespan, and 9.0% in total energy consumption. Similarly, LAS has an improvement of 11.1% in Success Ratio, 10.0% in makespan, and 10.0% in total energy consumption over DTS.

In the next Chapter, a game theory based scheduling algorithm is presented.

Table 4.5: Comparison with ECA (Deadline)

	ECA	LAS
SR	91	92.5
Makespan	4190.1	3508
Total Energy Consumption	7.35×10^6	6.59×10^6

Chapter 5

Game Theory based Scheduling Approach

Abstract:- The development of a large number of data centers increases the cloud system's energy consumption and affects system reliability. In this Chapter, a Real-time Task Scheduling Game (RTSG) framework is developed to solve allocating a real-time task to a self-aware and non-cooperative VM environment. At first, a non-cooperative task scheduling game model is designed to address the conflicting scheduling objectives,i.e., minimum energy consumption while satisfying reliability and deadline constraint. After that, a Vickery auction-based framework is introduced to find the Nash Equilibrium (NE) point representing the best $\langle task, VM \rangle$ combination of the RTSG game.

5.1 Introduction

The evolution of cloud computing facilitates the development of many data centers, which allows applications with varying demands like no deadline miss, minimum execution cost and energy consumption, etc. to operate in a virtualized environment. But, rapid development increases the energy consumption of the cloud system. Besides, the reliability of the system is affected by high energy consumption [1, 5]. The availability of a system is usually considered to be a factor in its reliability. It plays a vital role in the migration of critical data and applications to the cloud, maintaining user satisfaction, and avoiding revenue loss due to SLA violation penalties. With a heterogeneous and uncertain working environment, a cloud scheduler faces a challenge in deciding the best operational conditions for energy usage and reliability. The decision problem becomes even complicated for real-time tasks due to timing constraints. Further, multiple VMs present in the cloud environment compete for real-time tasks and decide based on their self-interest without sharing knowledge with other VMs so that the scheduling objective is optimized. In this context, game theory and auction theory help find the best $\langle task, VM \rangle$ combination, that meets the scheduling objective.

5.1.1 Generalized Game model

The game theory based scheduling algorithm is designed considering a generalized game theory model discussed in [21, 22, 36, 37, 112, 121]. The game model that is adopted has the following characteristics:

- a set of participants called players.
- each player has set of strategies which, specifies how player will behave or act.
- for each strategies, each player receives a utility value. The value indicates the willingness of a player to perform an action.
- a strategy s_i^j is a best response by a player pl_j to a choice of strategies $(s_1^j, s_2^j, ..., s_{i-1}^j, s_{i+1}^j, ..., s_n^j)$ by all the other players if

$$u_i^j(s_1^j, s_2^j, ..., s_{i-1}^j, s_i^j, s_{i+1}^j, ..., s_n^j) \geq u_i^j(s_1^j, s_2^j, ..., s_{i-1}^j, s_i^{j'}, s_{i+1}^j, ..., s_n^j) \tag{5.1}$$

  for all other possible strategies $s_i^{j'}$ available to player pl_j. Here, u_i^j is the utility value received by pl_j for strategy s_i^j.
- a Nash Equilibrium (NE) point, consists of each players best response to all the others is considered as a solution of a game.

5.2 Game Theory based Scheduling Framework

The main objective is to execute more tasks within timing constraints while optimizing reliability and energy consumption. A utility function u_i^j defined in terms of reliability and energy consumption of a task tk_i on a VM vm_j is used to represent the scheduling objective. It is formulated in Equation 5.2.

$$u_i^j = w1 \times \frac{vm_j^{rel}}{rel\_max} + w2 \times \frac{\varrho_j}{e\_max} \tag{5.2}$$

Besides, u_i^j is the estimated weighted sum value for executing task tk_i on VM vm_j. Let, rel_max and e_max are the maximum reliability and energy consumption possible in a cloud system. Both the values are used in the expression to normalize the respective quantities. Let $\mathcal{U}$ is a $m \times n$ matrix which contains u_i^j. The scheduler wants to optimize the sum of utility values of all the VMs in the system. Mathematically, it is expressed in Equation 5.3.

$$optimize \sum_{j=1}^{m} \sum_{i=1}^{n} u_i^j \times \mathcal{X}_i^j \tag{5.3}$$

The game theory based real-time task scheduling framework shown in Figure 5.1 includes the *Schedulability Analyzer*, *Decision Maker*, and *Resource Manager* component.

The *Schedulability Analyzer* handles the task queue and places the task with the earliest deadline at the head of the queue. Further, this component is used to check whether a task's deadline can be met or not. If the *Schedulability Analyzer* fails to find a suitable VM for a task, it asks the *Resource Manager* to add more VMs. Despite scaling up the VMs, if the *Schedulability Analyzer* fails to find an appropriate VM; then, the task gets rejected. The

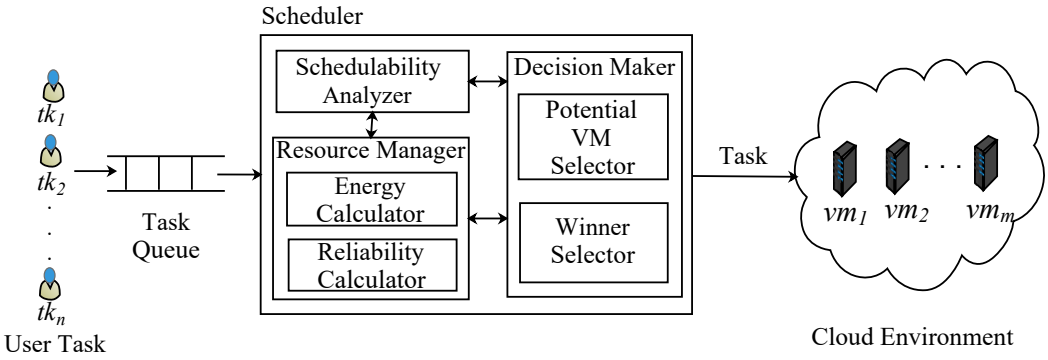


Figure 5.1: Game Theory based Real-time Task Scheduling Framework

Decision Maker uses the auction based approach to find a suitable VM for a task so that the scheduling objective is optimized. It consists of a *Potential agent/player/VM Selector* and *Winner Selector* sub-components. The *Potential VM Selector* unit uses the task's deadline, estimated finish time of all the tasks, and status of VM in the cloud environment to select VMs that can ensure the deadline requirement of a task. These potential VMs submit bids for a task defined by a utility function. The bids of the VMs are given as input to the *Winner Selector* unit. The *Winner Selector* unit decides a VM as a winner among potential candidates based on auction rules, i.e., the second-price sealed-bid. In the second-price sealed-bid, a VM will have to pay the amount equal to the second-best bid value.

5.3 Real-time Task Scheduling Game (RTSG) Model

The proposed bi-objective real-time task scheduling problem is formulated as a non-cooperative game. The outcome of the game is considered as a solution to the scheduling problem. The Real-time Task Scheduling Game (RTSG) model is described by 7-tuple $\langle sche, \mathcal{V}, \mathcal{T}, \psi, \mathcal{U}, sw, \mathcal{R} \rangle$ where,

- $sche$ is the game host or scheduler.
- Each VM $vm_j \in \mathcal{V}$ is modeled as a player for executing a task, where $\mathcal{V}$ represents set of m players.
- $\mathcal{T}$ represents set of n tasks.
- ψ is the task allocation strategy space for all players, i.e., $\psi = \psi_1 \times \psi_2 \times ... \times \psi_m$, where each element ψ_j is the strategy space of player vm_j. $\psi_j = \{s_i^j | i = 1, 2, ...n\}$, where s_i^j denotes that task tk_i is allocated to VM vm_j.

- Each element u_i^j of $\mathcal{U}$ is used to compute VM specific utility value $u_j^{'}$. It's calculation is represented in Equation 5.4.

$$u_j^{'} = \sum_{i=1}^{n} u_i^j \times \mathcal{X}_i^j \tag{5.4}$$

  where binary indicator $\mathcal{X}_i^j = 1$ if tk_i is assigned to vm_j, or $\mathcal{X}_i^j = 0$ otherwise. The value indicates the benefit, vm_j can gain if strategy s_i^j is chosen by it to execute tk_i.

- The overall goal of the scheduler is represented as social welfare (sw) and is estimated in Equation 5.5.

$$sw = \sum_{j=1}^{m} u_j^{'} \tag{5.5}$$

- $\mathcal{R} = \{R_1, R_2, ..., R_n\}$ be the set of rewards. The rewards can be non-monetary (e.g. score, ranking) or monetary (e.g. profit) quantity. Here, a non-monetary quantity is used for reward. $R_i^j \in R_i$ indicates reward or score for executing tk_i task on VM vm_j. It is computed in Equation 5.6.

$$R_i^j = \frac{1}{\gamma^{tk_i^{dl} - ft_i^j}}, \tag{5.6}$$

  where $\gamma \leq 1$ but $\gamma \neq 0$. The VM vm_j with large R_i^j value is preferable.

Figure 5.2 shows the working of the RTSG model. For each task, a utility value (u_i^j) is computed for all VMs in $\mathcal{V}$. Energy consumption and reliability are taken into account while calculating the utility value. Further, finish time ft_i^j is also calculated. If the finish time is less than task deadline tk_i^{dl}, then the bid (b_i^j) by VM vm_j is set as u_i^j otherwise it is set to -1. After that sel_vm_j is computed. If b_i^j is not equal to -1 then winner VM vm_j is selected and task tk_i is assigned to vm_j. The second-best VM in the winner list is selected, and the first winner pays an amount equal to $alp^{u_i^k}$.

A scheduler aims to find a task scheduling strategy that optimizes the overall utility value. The game is played in the following way: the scheduler initializes a set of tasks. Each VM (or player) simultaneously selects a task that has the best utility for itself. If multiple players choose the same task, assign the task to one of the competing players, which has the best utility value. Each VM executes the selected tasks. This process continues until all the tasks are selected from the task set. Algorithm 5.1 shows the initial step of RTSG. A Nash Equilibrium of the game is found to confirm that each VM made its best decision towards the scheduling objective. The challenges faced to find NE point in scheduling game is the incomplete information (i.e., a VM has incomplete or no information on the individual utility function of other VMs). Thus, a VM cannot precisely derive other VM's best strategies to make its own best response. This is similar to an auction scenario where each bidder would not share its valuation beforehand. In the process of scheduling decisions, the utility value

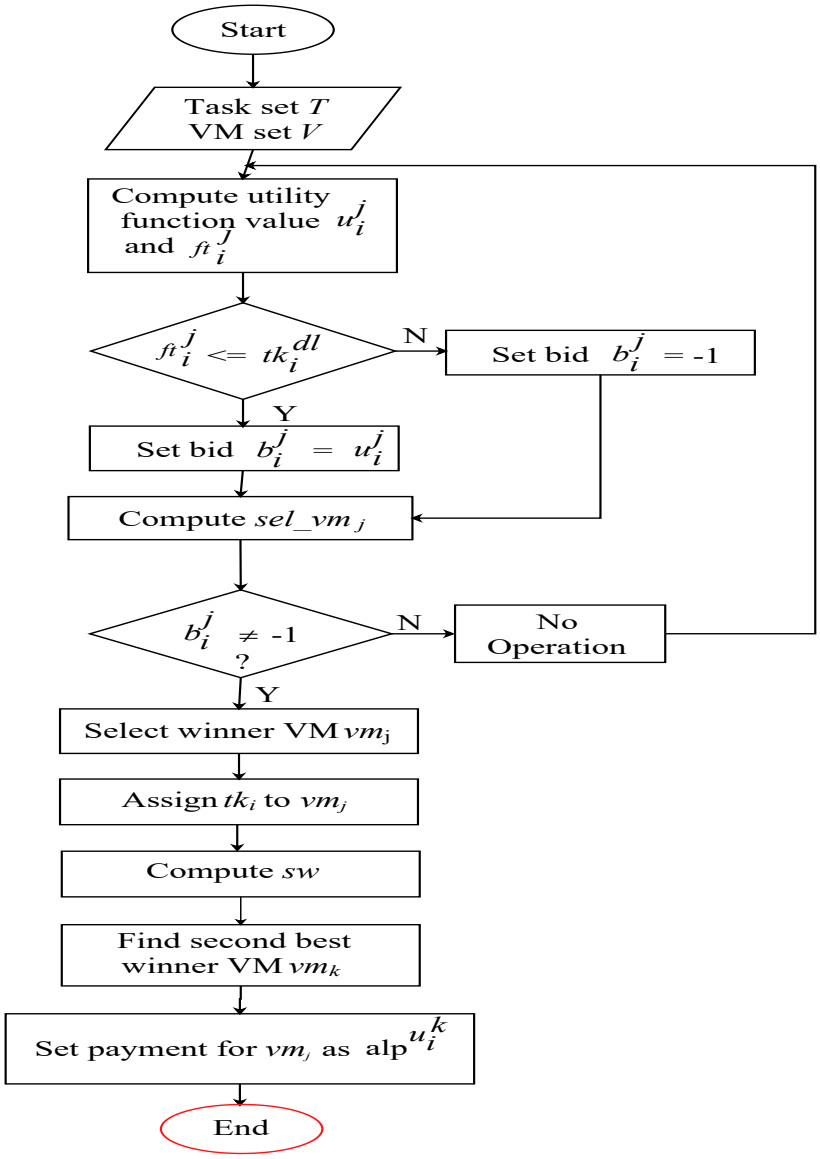


Figure 5.2: Flowchart of RTSG Model based Scheduling

of all participating VMs is not known to each other. Thus a Vickery auction or second-price sealed-bid based scheduling algorithm is preferred to address this challenge. One of Vickery auction properties says that "allocate an item to the bidder who values it the most." Hence, Vickery auction for the execution of real-time tasks can be justified, as it is employed to find the best VM for a task with incomplete information.

5.3.1 Nash Equilibrium based on Auction Mechanism

Auctions are a common and simple way of performing resource allocation in a multi-agent system. The basic rules governing an auction are; bidding for items, determining winning agents based on certain attributes, and payment made. In an auction, agents can express how much they want a particular item via their bid, and a central auctioneer (here it is, scheduler) can make the allocation based on these bids. In this regard, here, each VM is modeled as a

Algorithm 5.1 : Real-time Task Scheduling Game (RTSG)

Input Task set $\mathcal{T}$, VM set $\mathcal{V}$
Output NE_sol

1: **for** each $tk_i \in \mathcal{T}$ **do**
2: **for** each $vm_j \in \mathcal{V}$ **do**
3: Compute utility function value using Equation 5.2;
4: **end for**
5: **end for**
6: $NE\_sol \longleftarrow$ Nash_sol ($\mathcal{T}$, $\mathcal{V}, \mathcal{U}$);

player bidding for executing the task. Let a VM deploy a process, also known as an agent, to bid on its behalf. For instance, a process that provides service to VM vm_j is called agent $AG(vm_j)$, while the VM vm_j is busy executing the task, $AG(vm_j)$ may take part in the bidding process to gather more task for execution. Each agent submits a bid for a task that

Algorithm 5.2 : Nash_sol ($\mathcal{T}$, $\mathcal{V}, \mathcal{U}$)

Input: $\mathcal{T}$, $\mathcal{V}, \mathcal{U}, \mathcal{R}$
Output: Scheduled list sch

1: $b \longleftarrow$ Pot_vm_sel ($\mathcal{U}$);
2: $sch \longleftarrow$ Winner_det ($b, \mathcal{R}$);

is at the head of the task queue. Let the task set $\mathcal{T}$ act as the buyers of a bid. An assumption is made that a task can only be assigned to a single VM at a time. Further, the preference of a task $tk_i \in \mathcal{T}$ for some VMs over others can be specified by a utility value u_i^j. The utility value (or bid value b_i^j) that agent $AG(vm_j)$ derives from task tk_i can be considered as the valuation that vm_j assigns to tk_i. Besides, the valuation function reflects the agent's utility of owning the given task. The best Nash is the equilibrium with the minimum utility value, and the worst Nash is the equilibrium with the maximum utility value.

Algorithm 5.3 : Sche_tsk (ft_i^j, tk_i^{dl})

Input: finish time ft_i^j, task's deadline tk_i^{dl}
Output: Decision on deadline miss

1: **if** $ft_i^j \leq tk_i^{dl}$ **then**
2: return $True$;
3: **else**
4: return $False$;
5: **end if**

To select a winner from a potential set of VMs a selection criterion sel_vm_j is computed using Equation 5.7.

$$sel\_vm_j = \frac{R_i^j}{b_i^j} \tag{5.7}$$

The selection criterion defined in this way makes the winning chance of a VM sensitive with the bid value and reward value. Further, Equation 5.7 says that, to increase sel_vm_j

value, VM vm_j needs to either increase the reward value or decrease the bid value. The payment pay_j, to be paid by a VM who won, is computed as $alp^{(u_i^k)}$, where u_i^k is the VM's bid value which would have won if VM vm_j would have lost. The process of reaching

Algorithm 5.4 : Pot_vm_sel $(\mathcal{U}, \mathcal{T}, \mathcal{V})$

Input: $\mathcal{T}, \mathcal{V}, \mathcal{U}$
Output: bid matrix b

1: **for** each $tk_i \in \mathcal{T}$ **do**
2: **for** each $vm_j \in \mathcal{V}$ **do**
3: Compute ft_i^j using Equation 2.3.
4: **if** Sche_tsk $(ft_i^j, tk_i^{dl}) == True$ **then**
5: $b_i^j = u_i^j$;
6: **else**
7: $b_i^j = -1$;
8: **end if**
9: **end for**
10: **end for**

the Nash Equilibrium is shown in Algorithm 5.2. The process is discussed with the help of Algorithm 5.4 and Algorithm 5.5. The working of Algorithm 5.4 is explained below. Each VM compute the finish time of task tk_i, and schedulability is checked using Algorithm 5.3. If the condition holds, each VM bid its utility value as the bid value; otherwise, it is set to zero. After receiving the VM's bid value, a winner is selected based on the predefined criterion using Algorithm 5.5.

The working of Algorithm 5.5 is explained below. The overall process has a two-part, allocation rule and payment rule. In allocation rule (lines 2 - 17), a task is allocated to VM with the highest sel_vm value. The payment rule (lines 18 - 28) decides the payment of pay_j to be paid by the winner. The $\langle task, VM \rangle$ pair present in sch, shows the optimal strategy of each player resulting in Nash solution of the problem.

Theorem 5.1. *The time complexity of RTSG is* $O(nm)$.

Proof. The time complexity to compute utility value is $O(nm)$ (see Algorithm 5.1). Similarly, potential VM selection process need $O(nm)$ time (see Algorithm 5.4). The time complexity to compute selection criterion value is $O(nm)$. Further, the selection of winner (best) and second best winner needs $O(m)$ and $O(m-1)$ time respectively (see Algorithm 5.5). Rest of the lines time complexity is $O(1)$. So, the time complexity of RTSG is $O(nm)$+$O(nm)$+$O(nm)$+$O(m)$+$O(m-1)$=$O(nm)$. □

Theorem 5.2. *The RTSG model has a Nash Equilibrium.*

Proof. Using the method of contradiction, it is proved that RTSG has a Nash Equilibrium. Let, u_i^j and $u_i^{j'}$ is the utility value for strategy s_i^j and $s_i^{j'}$ of tk_i respectively. Let, strategy $s_i^{j'}$ is in Nash equilibrium. Let, strategy s_i^j is a best response by a VM vm_j that earns it more utility than using the strategy $s_i^{j'}$ causing violation in Nash equilibrium. But, $s_i^{j'}$ gives the maximum utility value making it best response for tk_i. As $s_i^{j'}$ makes the best response of

Algorithm 5.5 : Winner_det $(b, \mathcal{R})$

Input bid matrix b, reward $\mathcal{R}$, alp
Output Scheduled list sch, payment pay

$winner\_vm \longleftarrow Null$, $sch \longleftarrow Null$, $sch^{'} \longleftarrow Null$;
for each $tk_i \in \mathcal{T}$ **do**
  for each $vm_j \in \mathcal{V}$ **do**
    Compute sel_vm_j using Equation 5.7;
  end for
  $max\_sel \longleftarrow -\infty$;
  for each vm_j in $\mathcal{V}$ **do**
    if $b_i^j \neq -1$ **then**
      if $sel\_vm_j > max\_sel$ **then**
        $max\_sel \longleftarrow sel\_vm_j$;
        $winner\_vm \longleftarrow vm_j$;
      end if
    end if
  end for
  Assign tk_i to $winner\_vm$;
  Set $sch \longleftarrow sch \cup \langle tk_i, winner\_vm \rangle$;
end for
Compute social welfare value sw using Equation 5.5;
for each $vm_j \in sch$ **do**
  Find new winner $winner\_vm^{'}$ from set $\mathcal{V} \setminus \{vm_j\}$;
  $sch^{'} \longleftarrow sch^{'} \cup winner\_vm'$;
  if $sch^{'} \neq \phi$ **then**
    $vm_k \longleftarrow$ VM with highest sel_vm_k value;
    $pay_j \longleftarrow alp^{(u_i^k)}$;
  else
    $pay_j \longleftarrow 0$;
  end if
end for

vm_j, there couldn't exist any strategy whose utility value will be more than it. Hence, a contradiction occurs. □

Theorem 5.3. *There exist at least one Nash equilibrium in RTSG that is a solution in optimal solution set (Opt_Sol).*

Proof. Let, sol^* is a solution of Opt_Sol and $sol^0 = sol^*$. A solution sol is obtained by applying RTSG having $sw(sol) \leq sw(sol^*)$. But, sol^* has been assumed to be an optimal solution, $sw(sol^*)$ cannot be further reduced. Hence, $sw(sol) = sw(sol^*)$ and consequently sol is also an optimal solution. Therefore, it can be concluded that there is at least one Nash equilibrium that is also a solution in Opt_Sol. □

5.4 Performance Evaluation

The proposed approach is compared with the algorithms presented by Yang *et al.* [37], and Wang *et al.* [119]. The algorithms for comparisons are summarized as follows:

- Yang *et al.* [37]: Authors have used a cooperative game model considering the reliability of the balanced task for task scheduling in the cloud environment.
- Wang *et al.* [119]: The authors have designed an auction-based VM allocation mechanism to enhance energy saving in the cloud system.

The metrics used to evaluate the performance of the algorithm are Success Ratio (SR), makespan (τ), and total energy consumption (δ). The values of parameters are listed in Table 5.1.

Table 5.1: Parameters for Simulation Studies

Parameter	**Value (Fixed) -(Varied)**
Task Count	$1000 - (200, 400, 600, 800, 1000)$
VM Count	$80 - (40, 60, 80, 100)$
ts_{var}	$1500 - (500, 1000, 1500, 2000, 2500)$
sp_{var}	$400 - (200, 400, 600, 800, 1000)$
task arrival rate (λ)	0.05 - $(0.01, 0.03, 0.05, 0.08, 0.1)$
$baseD$	4 - $(2, 4, 6, 8)$

5.4.1 Simulation Setting

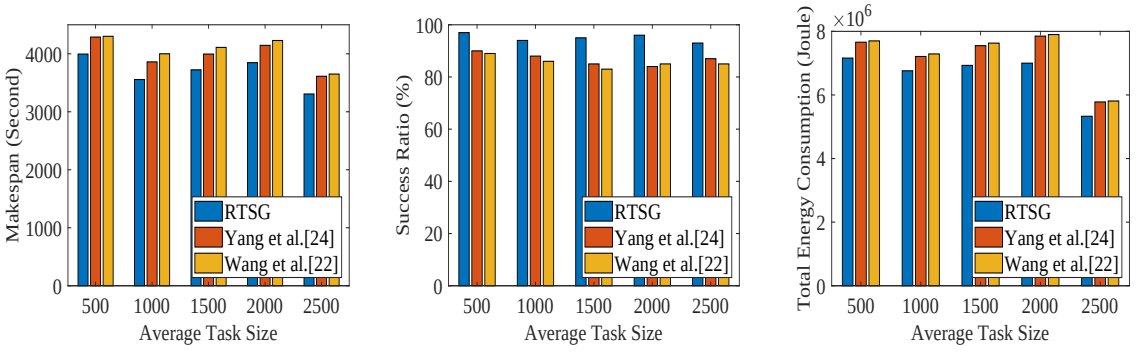
The detailed settings and parameters for evaluating the performance of RTSG approach are as follows.

- Task deadline is set as $tk_i^{dl} = tk_i^{ar} + baseD$ where $baseD$ is in Uniform distribution $U(4, 8)$.
- Let task size represents the task heterogeneity. It is uniformly distributed in the range $[ts_{avg} - ts_{var}]$ and $[ts_{avg} + ts_{var}]$ where ts_{avg} is average task size and ts_{var} is variable scope taking $ts_{avg} = 5000$ MI as center respectively.
- Let VM speed indicates VM heterogeneity. It is uniformly distributed in the range $[sp_{avg} - sp_{var}]$ and $[sp_{avg} + sp_{var}]$ where sp_{avg} is average VM speed and sp_{var} is variable scope taking $sp_{avg} = 1500$ MIPS as center respectively.
- The failure rate of VM vm_j follows Uniform distribution $U(1.0 \times 10^{-6}, 3.5 \times 10^{-6})$ per hour [26].

The rapid growth in IT infrastructure and applications allows heterogeneous task and VM in the cloud system. Hence, a set of simulations is conducted to analyze the effect of task and VM heterogeneity on system performance. Besides, the simulation results are discussed to find the impact of task and VM count, arrival rate, and deadline variation in the cloud system.

5.4.2 Impact of Task Heterogeneity on System Performance

The performance of the cloud system for task heterogeneity is shown in Figure 5.3. The task set with task size near the minimum limit has a short execution time, whereas task set with task size near the maximum limit has long task execution time. The short task execution time reduces the total execution time and hence the makespan. The long task execution time increases the total execution time and thus the makespan. So, the makespan as given in Figure 5.3a is stable. Further, the short execution time causes many tasks to complete their execution before the deadline. This increases the Success Ratio. The long task execution time causes deadline miss for a task that contributes to the reduced Success Ratio. Hence the heterogeneous task set causes neither increasing nor decreasing Success Ratio as shown in Figure 5.3b. The short task execution time reduces VM's active time, which contributes to low total energy consumption. Besides, the long task execution time increases VM's active time and, hence, the total energy consumption. So, the simulation outcome for total energy consumption is as shown in Figure 5.3c. However, the use of VM selection criteria considering reward and bid value helps scheduling with the RTSG model performs better against approaches presented in Yang *et al.* and Wang *et al.*.



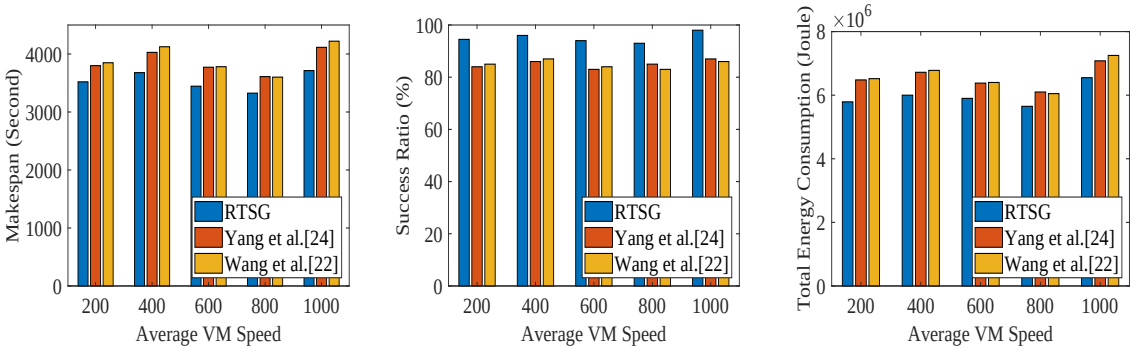
(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 5.3: Impact of Task Heterogeneity on System Performance

5.4.3 Impact of VM Heterogeneity on System Performance

The effect of VM heterogeneity on system performance is shown in Figure 5.4. The content of the VM set plays a vital role in quantifying system performance. If the VM set has low-speed VMs, then there is the short task execution time, and if the VM set has high-speed VMs, the task execution time will be high. Thus makespan decreases for short task execution time and increases for long task execution time. The simulation result, as shown in Figure 5.4a, confirms the effect of VM heterogeneity on makespan. Besides, the short task execution time contributes to a rise in the Success Ratio, as it allows many tasks to complete execution before the deadline. The long task execution time increases the chance of a task to miss its deadline, reducing the Success Ratio. So, the Success Ratio due to VM

heterogeneity is as given in Figure 5.4b. The use of high-speed VM causes a rise in total energy consumption. Further, the total energy consumption is small for low-speed VM. Hence the total energy consumption is stable for heterogeneous VM set as given in Figure 5.4c. Besides, the simulation outcome shows that the use of reward and bid value in VM selection causes performance improvement of RTSG compared to approaches presented in Yang *et al.* and Wang *et al.*.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

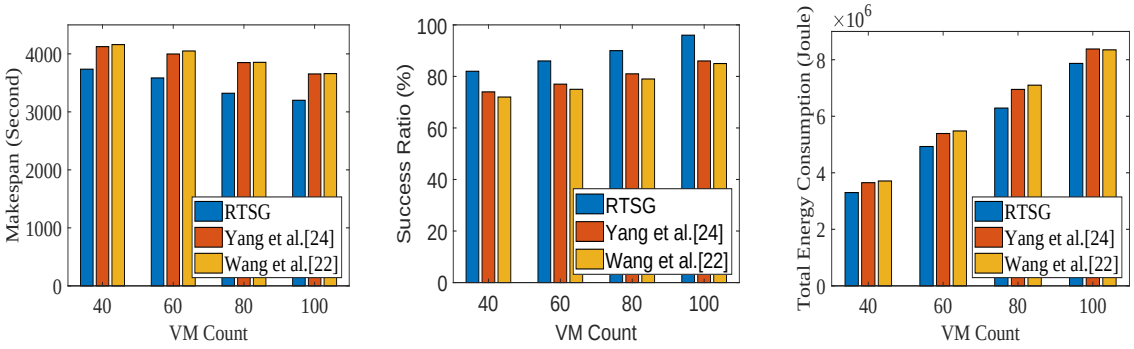
Figure 5.4: Impact of VM Heterogeneity on System Performance

5.4.4 Impact of VM Count on System Performance

The outcome of simulation varying VM count in the range $[40 - 100]$ with a step of 20 is shown in Figure 5.5. A large number of tasks get executed on different VMs within deadline constraints simultaneously because of VM count addition. The total execution time is small and hence a decrease in makespan, as shown in Figure 5.5a. As many tasks getting executed before the deadline, the Success Ratio increases with an increase in VM count. The simulation outcome for the Success Ratio is given in Figure 5.5b. Moreover, the increase in VM's active time due to the large task count increases the total energy consumption. Besides, the idle energy consumption due to the presence of idle VM in large VM set adds to the total energy consumption. Thus there is an increase in total energy consumption as given in Figure 5.5c. From Figure 5.5, it can be inferred that RTSG has better energy saving, less makespan, and better Success Ratio as compared to approaches presented in Yang *et al.* and Wang *et al.*.

5.4.5 Impact of Task Count on System Performance

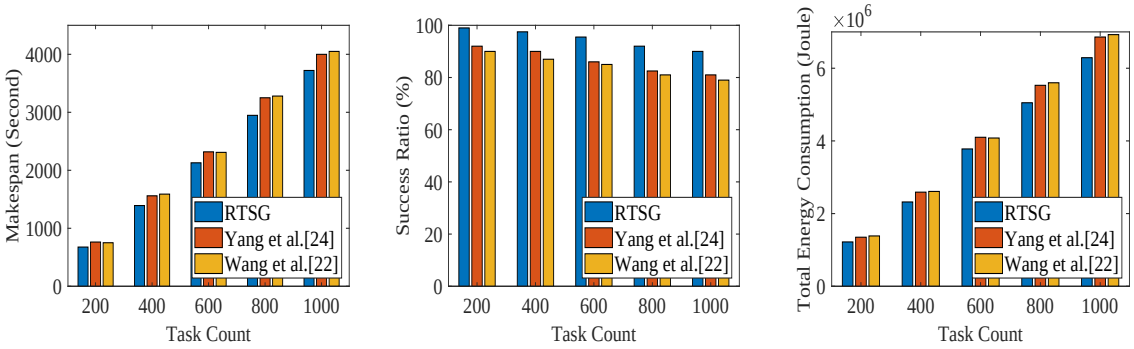
Task count's effect on system performance is analyzed by varying the task count in the range $[200 - 1000]$ in the interval of 200. From Figure 5.6a, it can be observed that the increase in task count increases makespan. This is because the rise to task count with fixed VM count increases the execution time of tasks. Further, the chance of deadline miss of a task increases with an increase in task count and constant VM count. Hence the Success Ratio decreases,



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 5.5: Impact of VM Count on System Performance

as given in Figure 5.6b. The increase in task count increases VM's active time, and thus there is a rise in total energy consumption. The simulation outcome, as given in Figure 5.6c for total energy consumption, confirms the claim. It can be seen from Figure 5.6 that the VM selection considering reward and bid value in RTSG allows it to have improved performance in comparison with approaches presented in Yang *et al.* and Wang *et al.*.



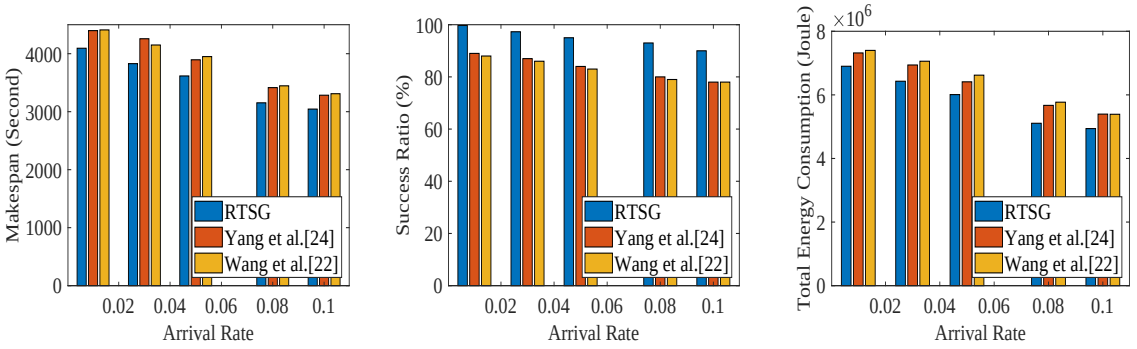
(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 5.6: Impact of Task Count on System Performance

5.4.6 Impact of Arrival Rate on System Performance

The arrival rate (λ) value is varied from 0.01 to 0.1 with step 0.02. As given in Figure 5.7, the simulation outcomes show the effect of arrival rate on system performance. There are many tasks within a short time due to an increase in the arrival rate. So, very few tasks complete its execution, which in turn decreases the makespan. Figure 5.7a shows the effect of the arrival rate on the makespan. The presence of many tasks in a short time duration allows fewer tasks to complete execution before the deadline. Hence there is a decrease in Success Ratio, as given in Figure 5.7b. Further, there is a decrease in total energy consumption as given in Figure 5.7c because only a small set of tasks complete their execution due to high task count in a short time. However, RTSG has improved system performance in terms of

makespan, total energy consumption, and Success Ratio compared to approaches presented in Yang *et al.* and Wang *et al.*.

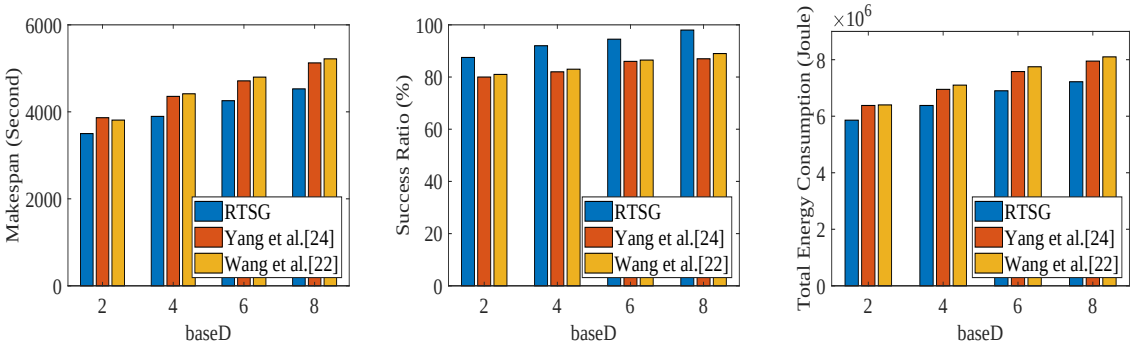


(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 5.7: Impact of Arrival Rate on System Performance

5.4.7 Impact of Deadline Variation on System Performance

The $baseD$ value is set in the range $[2-8]$. The makespan and Success Ratio, as given in Figure 5.8a and Figure 5.8b, increases, as a large number of tasks complete execution before the deadline due to a rise in $baseD$. Further, the increase in $baseD$ allows many tasks to complete execution, which increases the active time of VM. Thus, as shown in Figure 5.8c for total energy consumption, the simulation outcome increases with an increase of $baseD$. Figure 5.8, it can be observed that reward and bid value-based VM selection in RTSG help achieve better performance in terms of makespan, Success Ratio, and total energy consumption compared to others.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 5.8: Impact of Deadline Variation on System Performance

5.5 Summary

This Chapter proposed bi-objective scheduling for real-time tasks in the cloud system. The objectives that are considered for optimization are energy consumption and reliability. Game theory and auction theory are considered to optimize the stated objective. The proposed scheduling is first represented as a non-cooperative game, and then the Vickery auction theory is applied to find the Nash Equilibrium. The result is the best scheduling possible for a set of tasks. The experimental outcomes implied that the proposed game theory based approach performs better than the existing algorithms by Yang *et al.* and Wang *et al.*. The performance improvement of RTSG compared to work presented by Yang *et al.* is 11.2% in Success Ratio, 9.1% in makespan, and 8.5% in total energy consumption. Compared to the work of Wang *et al.*, the proposed RTSG has an improvement of 11.7% in Success Ratio, 9.8% in makespan, and 9.4% in total energy consumption.

In the next Chapter, a Primary-Backup (PB) based fault-tolerant scheduling algorithm is presented.

Chapter 6

Primary-Backup based Fault-tolerant Scheduling Algorithm

Abstract:- Some applications that are moving to the cloud, demand both timing and functional correctness despite the presence of failure. It becomes more cumbersome for real-time applications that need fulfillment of deadline requirements, even in the presence of VM failure. In this regard, a fault-tolerant strategy helps to achieve scheduling objectives while satisfying deadline constraints. Further, the detection of a fault in a VM is an essential process in a fault-tolerant mechanism. This Chapter presents an acceptance test-based method to detect VM failure. Besides, a fault-tolerant scheduling algorithm is proposed using the Primary-Backup and BB overlapping method to optimize energy consumption while satisfying deadline and reliability constraints.

6.1 Introduction

The cloud supported by virtualized data centers creates an illusion of unlimited cloud resources to users and allows elastic provisioning based on user's demand. But, an unreliable cloud system with a higher probability of resource failures inevitably results in more interruption of running VMs, which implies a reduction in the performance of the cloud service [6]. In this context, a fault-tolerant strategy helps guarantee the reliability of the cloud system [5, 7]. The two copy replication Primary-Backup (PB) is widely used by researchers to design fault-tolerant strategy [5, 25–28]. Further, Primary-Backup (PB) overlapping and Backup-Backup (BB) overlapping is used to improve resource utilization. An essential aspect of the fault-tolerant strategy is detecting a fault in the components (e.g., VM, PM, etc.) of the cloud system. The fault detection also plays a significant role in maintaining the reliability requirements of the cloud system. The most popular failure detection strategies are acceptance tests, intrusion detection, and heartbeat/pinging [38, 93]. Intrusion detection is based on behavior observation of the component, and an alarm is raised whenever a component shows abnormal behavior. Heartbeat/pinging is based on keep-alive message strategies. Any unusual behavior due to software faults or transient hardware faults or resource time failure is detected in the acceptance test strategy, then a failure alarm/ message is raised. Acceptance criteria are defined based on functional correctness and completeness,

timeliness, and performance. The acceptance test validates VM's final results' correctness and ensures that they do not lead to disastrous consequences. The failure detection accuracy depends on monitoring process architecture, i.e., centralized Intrusion Detection System (IDS), partially distributed (heartbeat), or distributed (acceptance test). In the acceptance test strategy, the monitor and monitored components are present in the same unit, thus reduces the false alarm rate. Additionally, the accuracy of failure alarm in acceptance strategy is high compared to IDS and heartbeat [93]. Another advantage of the acceptance test is cloud architecture-independent implementation making the approach scalable.

6.2 Primary-Backup based Scheduling Framework

This work aims to execute more tasks within timing constraints (deadline) and achieve fault-tolerance while optimizing reliability and energy consumption. The representation and calculation of the scheduling objective are similar to one that is defined in Section 5.2 but with added fault-tolerance features.

The fault-tolerant scheduling framework shown in Figure 6.1 consists of a *Backup Controller, Schedulability Analyzer*, and a *Resource Manager*. Along with the generation of a mapping between task and VM, the *Resource Manager* gather status information of VM that includes the execution status of the scheduled task, availability of VM, etc. The *Resource Manager* decides whether to run a backup copy or not.

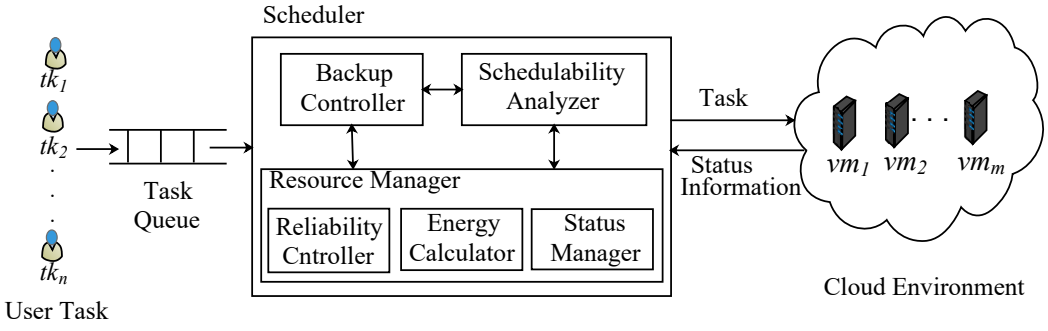


Figure 6.1: Primary-Backup based Fault-tolerant Real-time Task Scheduling Framework

The *Backup Controller* first produces a backup copy of a task and then delivers both primary and backup copies to the *Schedulability Analyzer*.

The *Schedulability Analyzer* is used to check whether a task can be finished before the deadline or not. This decision helps to map a real-time task to an appropriate VM. If *Schedulability Analyzer* fails to find a suitable VM for a task, it informs the *Resource Manager* to add new VMs. Despite the addition of new VMs, if a mapping can't be found to ensure the task's deadline, the task gets rejected.

Resource Manager consists of three sub-components: *Reliability Controller, Energy calculator*, and *Status Manager*. The *Reliability Controller* is used to compute the reliability, and the *Energy Calculator* unit calculates the energy usage of a VM while executing a

task. Further, the values obtained are used in the scheduling process while guaranteeing fault-tolerance. The *Status Manager* computes and stores the VM's state information in the VM Status Table (VST). The information in VST is used for VM fault detection and management. The fault detection is carried out after the execution of the primary copy. If the execution of the primary copy of a task is successful, then the information is sent to the *Backup Controller*, which in turn deallocates the resources from VM, running the backup copy of the task. On the other hand, if execution fails, then the *Backup Controller* is informed to execute a backup copy of the task to achieve fault-tolerance.

6.2.1 Task Model

As Primary-Backup concept is used for fault-tolerant scheduling scheme, it is assumed that each task tk_i has two copies, i.e., P_tk_i (primary) and B_tk_i (backup). Both primary and backup copies have same task attributes, but executed on different VMs. Let, $P\_st_i^j$ and $P\_ft_i^j$ are the start time and finish time of P_tk_i on VM vm_j. For a backup copy, $B\_st_i^j$ and $B\_ft_i^j$ indicates the start time and finish time, respectively. Let, P_est_i and P_eft_i are the earliest start time and finish time of primary copy whereas, B_lst_i and B_lft_i are the latest start time and finish time of the backup copy. $v(P\_tk_i)$ and $v(B\_tk_i)$ represent the VMs where P_tk_i and B_tk_i are allocated respectively. Let, $P\_et_i^j$ and $B\_et_i^j$ are the expected execution time of primary and backup copy of task tk_i on VM vm_j respectively.

Definition 6.1. *The earliest start time P_est_i for P_tk_i is computed as:*

$$P\_st_i^j = max(P\_ft_i^j, tk_i^{ar}), \tag{6.1}$$

$$P\_est_i = min(P\_st_i^j) \ \ 1 \leq j \leq m. \tag{6.2}$$

Definition 6.2. *The earliest finish time P_eft_i for P_tk_i is computed as:*

$$P\_eft_i = P\_est_i + P\_et_i^j. \tag{6.3}$$

Definition 6.3. *The latest start time B_lst_i for B_tk_i is obtained as follows:*

$$B\_st_i^j = tk_i^{dl} - B\_et_i^j, \tag{6.4}$$

$$B\_lst_i = max(B\_st_i^j) \ \ 1 \leq j \leq m. \tag{6.5}$$

Equation 6.4 states that the start time of B_tk_i can be delayed by at most $(tk_i^{dl} - B\_et_i^j)$.

Definition 6.4. *The latest finish time B_lft_i of B_tk_i is at most task's deadline tk_i^{dl}. It is represented as:*

$$B\_lft_i = tk_i^{dl}. \tag{6.6}$$

Definition 6.5. *The relative interval between tk_i^{dl} and $P\_ft_i^j$ determines whether scheduling of task tk_i on vm_j is non fault-tolerant or fault-tolerant. It is assumed that deadline miss indicates schedulability failure. Let,*

$$SL_i^j = tk_i^{dl} - P\_ft_i^j. \tag{6.7}$$

Case 1: *If $SL_i^j < 0$ then $P\_ft_i^j$ will exceed the deadline tk_i^{dl} making the schedule non fault-tolerant.*
Case 2: *If $SL_i^j \geq 0$ then task can be completed before its deadline making the schedule fault-tolerant.*
If all VMs in cloud system are non fault-tolerant for task tk_i then the scheduler should add new VM.

Theorem 6.1. *Let, $v(P\_tk_i) = vm_j$ and $v(B\_tk_i) = vm_k$. The overall reliability of a task tk_i with primary (P_tk_i) and backup (B_tk_i) replica is*

$$R(tk_i) = e^{-vm_j^{fr} \times P\_X_i^j \times P\_et_i^j} + e^{-vm_k^{fr} \times B\_X_i^j \times B\_et_i^j} - e^{-vm_j^{fr} \times P\_X_i^j \times P\_et_i^j - vm_k^{fr} \times B\_X_i^j \times B\_et_i^j}, \tag{6.8}$$

where $P\_X_i^j$ and $B\_X_i^j$ are binary indicators showing if P_tk_i or B_tk_i is assigned to vm_j or not respectively.

Proof. The reliability of P_tk_i executed on VM vm_j without failure is computed as:

$$r_p = e^{-vm_j^{fr} \times P\_X_i^j \times P\_et_i^j}. \tag{6.9}$$

If execution of P_tk_i is not successful then backup copy B_tk_i is executed. The reliability of B_tk_i executed on vm_k is

$$r_b = e^{-vm_k^{fr} \times B\_X_i^j \times B\_et_i^j}. \tag{6.10}$$

In the presence of failure the overall reliability of task tk_i is

$$R(tk_i) = r_p + r_b(1 - r_p) = e^{-vm_j^{fr} \times P\_X_i^j \times P\_et_i^j} + e^{-vm_k^{fr} \times B\_X_i^j \times B\_et_i^j} - e^{-vm_j^{fr} \times P\_X_i^j \times P\_et_i^j - vm_k^{fr} \times B\_X_i^j \times B\_et_i^j}. \tag{6.11}$$

□

6.2.2 Fault Model

The fault model used is similar to the model presented in [5, 28, 91], and is summarized as below:

- The faults can be transient or permanent and assumed to be independent and only affect a single VM at any time instant,
- If the execution of the primary copy fails, then backup copy always finishes successfully,
- If the execution of the primary copy is successful, the backup copy is terminated, and reserved resources are deallocated from the VM running it,
- A new task will not be allocated to any known failed VM,
- An acceptance test is available to detect the failure of a VM.

The proposed fault model can be extended to tolerate multiple VM failures by first dividing the VM set into several small groups. Then, in each group, the proposed fault model is applied as reported in [109, 110] to process multiple VM failure.

6.3 Fault-tolerant Scheduling Algorithm

This section discusses the Reliability and Energy (REO) aware fault-tolerant scheduling algorithm. The proposed approach uses the PB concept, BB overlapping, and backup deallocation techniques. Before discussing the scheduling algorithm, an analysis of the BB overlapping technique is presented in Section 6.3.1.

6.3.1 Backup-Backup Overlapping

In Backup-Backup (BB) overlapping, backups of multiple primaries are scheduled on the same or overlapping time interval on a VM. The following conditions must be satisfied for the overlapping of backups on a VM.

C1: The primary (P_tk_i) and backup (B_tk_i) copy of a task tk_i must be mutually exclusive in space. It can be formulated as:

$$v(P\_tk_i) \neq v(B\_tk_i). \tag{6.12}$$

This condition helps to overcome permanent failure as backup copy always completes task execution in the presence of execution failure of the primary copy.

C2: For an active backup copy of a task condition in Equation 6.13 must hold.

$$P\_st_i^j \leq B\_st_i^j \leq P\_ft_i^j \tag{6.13}$$

C3: For passive backup copy of a task (say t_i^{bk}) condition in Equation 6.14 must hold.

$$P\_ft_i^j \leq B\_st_i^j \tag{6.14}$$

VM vm_j in both **C2** and **C3** must satisfy **C1**.

C4: Even though multiple backups can have overlapping time interval on a VM but only one backup is allowed to execute at any time instant. Let $slot(.)$ shows the time interval between start time and finish time. For instance, for $tk_i, tk_k \in \mathcal{T}$ and $v(P\_tk_i) = v(B\_tk_i)$, if $slot(B\_tk_i) \cap slot(B\_tk_k) \neq \phi$ (overlapping time interval) then at any time either B_tk_i or B_tk_k will be executed.

C5: At most, one primary copy is expected to encounter fault to ensure only one backup will be executed among overlapped backups.

The overlapping technique for active and backup copies is formalized in the following Theorems.

Theorem 6.2. *Given two tasks $tk_i, tk_k \in \mathcal{T}$, passive backup copies B_tk_i and B_tk_k can be overlap only if P_tk_i and B_tk_k are on different VMs. It is represented as:*

$$(v(B\_tk_i) = v(B\_tk_k)) \wedge ((slot(B\_tk_i) \cap slot(B\_tk_k)) \neq \phi) \Rightarrow v(P\_tk_i) \neq v(P\_tk_k). \tag{6.15}$$

Proof. By contradiction, let P_tk_i and P_tk_k are scheduled on same VM, i.e., $v(P\_tk_i) = v(P\_tk_k) = vm_1$. Further B_tk_i overlap B_tk_k, i.e., $(slot(B\_tk_i) \cap slot(B\_tk_k)) \neq \phi$ as shown in Figure 6.2. If vm_1 fails, both backup copies need to be executed at same time on same VM. This situation cause timing conflict due to condition **C4**. A contradiction occurs. Hence, $v(P\_tk_i) \neq v(P\_tk_k)$. □

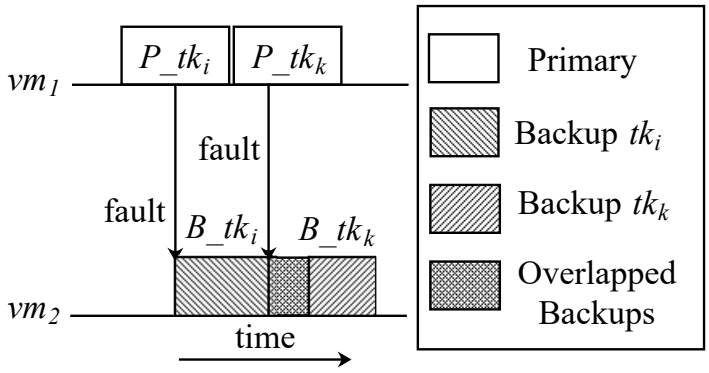


Figure 6.2: Overlapping of Two Passive Backup Copies

Theorem 6.3. *Given two tasks $tk_i, tk_k \in \mathcal{T}$, if at least one of the backup copy (B_tk_i or B_tk_k) is active then B_tk_i can not overlap B_tk_k.*

Proof. Let, B_tk_i overlap with B_tk_k.

Case 1: Both B_tk_i and B_tk_k are active backup copies. The two copies need to be executed simultaneously on the same VM due to the property of active backup copy, i.e., $(slot(B\_tk_i) \cap slot(B\_tk_k)) \neq \phi$ if P_tk_i and P_tk_k fails. This cause a contradiction according to conditions **C4** and **C5**. Hence, B_tk_i cannot overlap B_tk_k.

Case 2: Let B_tk_i is passive and B_tk_k is active backup copy. If P_tk_i fails at time t, where $t \leq P\_ft_i^j \leq B\_st_i^j$, B_tk_i is invoked. Due to property of active backup copy, B_tk_k is executed along with P_tk_k. So, invocation of B_tk_i causes B_tk_k and B_tk_i to execute simultaneously. This is a contradiction according to condition **C4**. Hence, B_tk_i cannot overlap B_tk_k. An example of this situation is shown in Figure 6.3.

Case 3: B_tk_i is active and B_tk_k is passive backup copy. If P_tk_k fails at time t where, $t \leq B\_st_k^j \leq B\_ft_i^j$, B_tk_k is invoked. Based on property of active backup copy B_tk_i is executed along with P_tk_i. The invocation of B_tk_k causes B_tk_i and B_tk_k to execute simultaneously. Hence, B_tk_k cannot overlap B_tk_i according to condition **C4**. □

Theorem 6.4. *Given two tasks $tk_i, tk_k \in \mathcal{T}$, if active backup copy B_tk_i overlaps with passive backup copy B_tk_k then the earliest start time of B_tk_k is*

$$B\_est_k \geq max(P\_eft_i, P\_eft_k). \tag{6.16}$$

Proof. Let, P_tk_k fails at time t where $t \leq B\_st_k^j < B\_ft_i^j$, B_tk_k is invoked. As B_tk_i overlap B_tk_k, i.e., $(slot(B\_tk_i) \cap slot(B\_tk_k)) \neq \phi$. This situation as shown in Figure

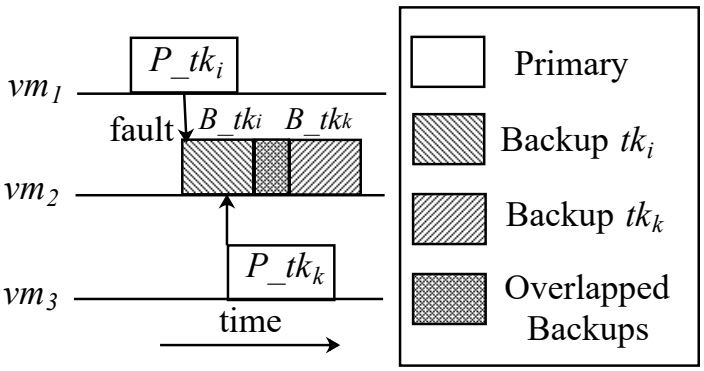


Figure 6.3: Overlapping of Passive and Active Backup Copies (**Case 2**)

6.4 makes both copies to execute simultaneously at same VM, but a conflict occurs due to condition **C2**. So, earliest start time of B_tk_k must be later than the finish time of P_tk_i and P_tk_k. Hence,

$$B\_est_k \geq max(P\_eft_i, P\_eft_k). \tag{6.17}$$

□

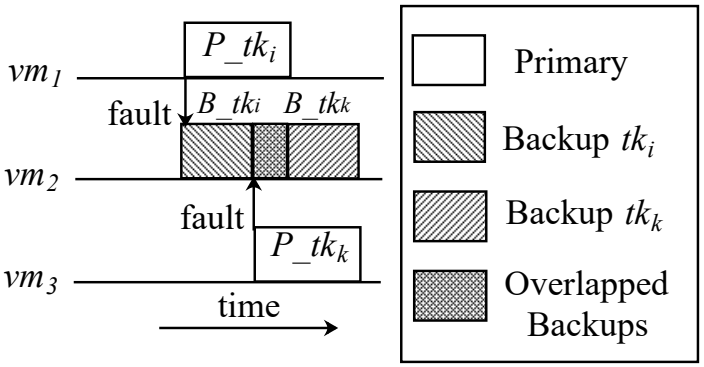


Figure 6.4: Overlapping of Active and Passive Backup Copies with Earliest Start Time

6.3.2 Scheduling Strategy

An acceptance test-based framework is designed to detect VM failure. The failure detection is based on schedulability failure (deadline miss) (p_1) and response time failure (p_2) of a VM. The response time of a task P_tk_i on VM vm_j is defined as the interval between the arrival of a task, and it's finish time, i.e., $P\_rt_i^j = P\_eft_i - tk_i^{ar}$. Let $H(P\_rt_i^j)$ and $L(P\_rt_i^j)$ is the higher and lower limit of response time possible for a task on vm_j. If $P\_rt_i^j < L(P\_rt_i^j)$ or $P\_rt_i^j > H(P\_rt_i^j)$, it is assumed that there is response time failure for vm_j. The response time calculation for a task P_tk_i is shown in Algorithm 6.1.

The schedulability failure is signified by whether a task meets its deadline or not. It is assumed that the primary copy of a task should be assigned to a VM with the earliest finish time, whereas the backup copy should be assigned to a VM with the latest start time to achieve fault-tolerance. Algorithm 6.2 and Algorithm 6.3 gives the detail of schedulability failure of the primary and backup copy of a task, respectively.

Algorithm 6.1 : RT (P_tk_i, vm_j)

Input: P_tk_i, vm_j
Output: Decision regarding response time failure

Compute $P\_rt_i^j = P\_eft_i - tk_i^{ar}$;
if $L(P\_rt_i^j) \leq P\_rt_i^j \leq H(P\_rt_i^j)$ **then**
    Return $True$;
else
    Return $False$;
end if

Algorithm 6.2 : P_Sched_tsk $(P\_ft_i^j, tk_i^{dl})$

Input: $P\_ft_i^j$, tk_i^{dl}
Output: Decision regarding primary copy deadline miss

if $P\_ft_i^j \leq tk_i^{dl}$ **then**
    Return $True$;
else
    Return $False$;
end if

In this work, the timeliness (response time failure) and performance (schedulability failure) criteria are used to study a VM's behavior and check whether its $Faulty$ or not. Let, $np\_1(vm_j)$ and $np\_2(vm_j)$ count the number of times p_1 and p_2 occurs on VM vm_j respectively. mp_1 and mp_2 are the median of number of reports of p_1 and p_2 on all the VMs respectively. Compute $pc\_a(vm_j) = \mid \frac{np\_a(vm_j) - mp\_a}{mp\_a} \mid$, where $a = 1, 2$. Let, Th_p_a be the threshold value for p_a and $stat(vm_j)$ represents the status of vm_j. The detailed of acceptance test framework is shown in Algorithm 6.4. Usually, an acceptance test for the real-time task considers the only schedulability failure [109]. In this context, a two-step fault detection method is employed in this work. In the first step it is checking whether the schedulability failure of VM reaches a threshold or not. If VM passes this test, then the response time failure count is checked. Based on whether the counter reaches the threshold value or not, VM's status is set to $Faulty$ or OK.

The primary copies in REO are assigned according to As Early As Possible (AEAP) mechanism. Figure 6.5 shows the working of the primary copy schedule. At first, a primary

Algorithm 6.3 : B_Sched_tsk $(B\_st_i^k, tk_i^{dl})$

Input: $B\_st_i^k$, tk_i^{dl}
Output: Decision regarding backup copy deadline miss

if $B\_st_i^k \leq tk_i^{dl}$ **then**
    Return $True$;
else
    Return $False$;
end if

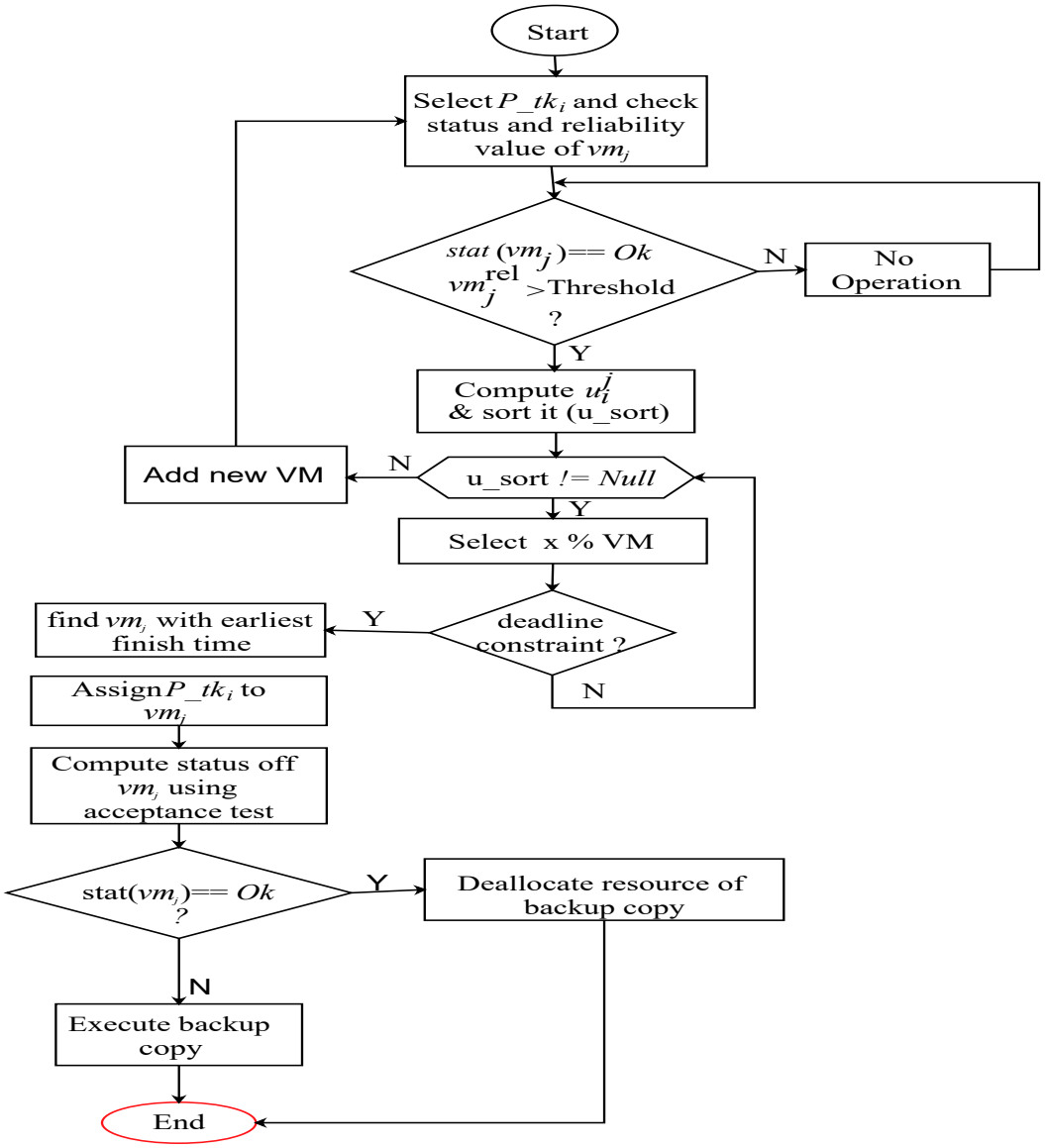


Figure 6.5: Flowchart of Primary Copy Scheduling

copy of a task is selected, and then the status and reliability value of each VM is checked. If the condition holds, the utility value is computed. The VMs are sorted based on utility value (u_sort). Then top $x\%$ VMs are selected from the sorted list. A VM (vm_j), which satisfies deadline constraint and has the earliest finish time, is selected to execute P_tk_i. The status of vm_j is computed using the acceptance test. If the status of vm_j is Ok, then deallocate resources for the backup copy; otherwise, execute the backup copy of the task.

The objective of primary copy scheduling in REO presented in Algorithm 6.5 is to allocate primary copies to the VM with the earliest finish time and satisfy scheduling objectives. The working of Algorithm 6.5 is as follows: top $x\%$ VMs, which satisfies the scheduling objective, is chosen as candidate VMs (lines 3 - 10). Then, the VM with the earliest finish time is chosen to execute the task's primary copy (lines 12 - 23). If there is no VM in candidate VM set V_cand, then the next $x\%$ VMs in u_sort will be

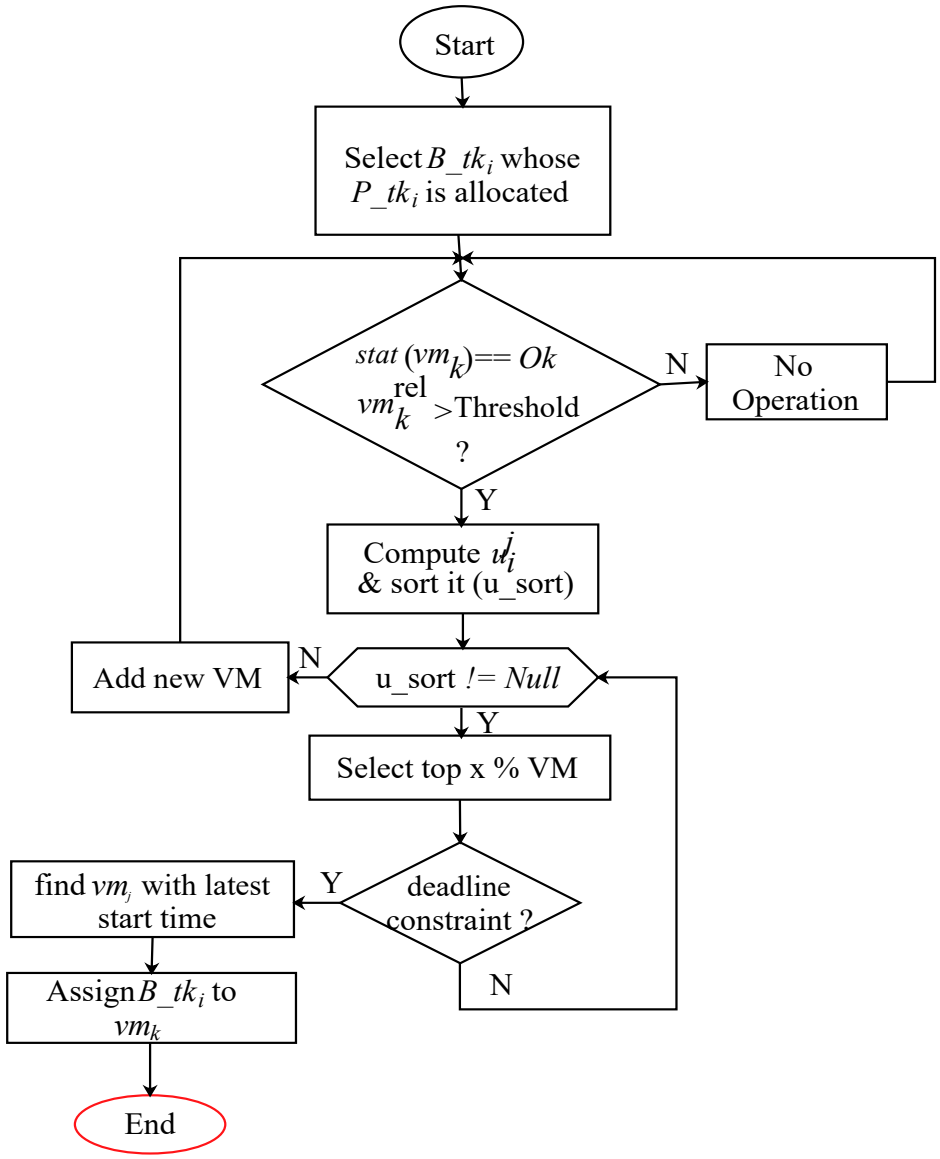


Figure 6.6: Flowchart of Backup Copy Scheduling

used to find the appropriate VM (lines 24 - 26). The overall process helps to map primary copy to a VM that can complete its execution as early as possible while optimizing the scheduling objective. Despite this, if there is no VM in the current cloud system that can execute P_tk_i, then add new VMs (lines 28 - 30). After the VM performing its intended computation, an acceptance test $A\_Test(v(P\_tk_i))$ is carried out. If $A\_Test(v(P\_tk_i))$ returns status of VM,i.e., $stat(v(P\_tk_i))$ as Ok then it is considered that execution of P_tk_i is successful and Algorithm 6.6 is called to deallocate resources from VM where backup is assigned. If $A\_Test(v(P\_tk_i))$ returns $stat(v(P\_tk_i))$ as $Faulty$, backup copy is scheduled by $B\_sch(B\_tk_i, vm_k)$ (lines 32 - 39). If for any task tk_i, $task\_completed(tk_i) \neq 1$ then it is rejected (lines 41 - 45).

Theorem 6.5. *The time complexity for scheduling primary copy in REO as discussed in Algorithm 6.5 is $O(m^2n + nmlogm)$ where m is number of VM in the system, and n is number of task in the system.*

Algorithm 6.4 : A_Test (vm_j)

Input: vm_j
Output: VM status

1: **if** P_Sched_tsk $(P\_ft_i^j, vm_j)) == False$ **then**
2: $np\_1(vm_j) = np\_1(vm_j) + 1$;
3: **end if**
4: **if** RT $(P\_tk_i, vm_j) == False$ **then**
5: $np\_2(vm_j) = np\_2(vm_j) + 1$;
6: **end if**
7: **for** $a = 1, 2$ **do**
8: Compute $pc\_a(vm_j)$;
9: **end for**
10: **if** $pc\_1(v_j) \geq Th\_p\_1$ **then**
11: Return $Faulty$;
12: **else if** $pc\_2(vm_j) \geq Th\_p\_2$ **then**
13: Return $Faulty$;
14: **else**
15: Return Ok;
16: **end if**

Proof. The worst case time complexity to compute u_i^j is $O(m)$ (lines 4 - 8). The sorting process takes $O(mlogm)$ times. The worst case time complexity to determine the VM with earliest finish time for P_tk_i is $O(m^2)$ (lines 11 - 27). The rest of the lines has time complexity $O(1)$. The time complexity of the algorithm is $n(O(m)+O(mlogm)+O(m^2)) = O(mn) + O(nmlogm) + O(m^2n) = O(m^2n + nmlogm)$. □

The objective of backup copy scheduling is to allocate backup copies to a VM with the latest start time so that the scheduling objective is optimized. The backup copy scheduling employs the As Late As Possible (ALAP) mechanism and is discussed in Algorithm 6.7. Figure 6.6 shows the working of the backup copy schedule. At first, backup copy (B_tk_i) of a task is selected whose primary copy is scheduled, and then the status and reliability value of each VM is checked. If the condition holds, the utility value is computed. The VMs are sorted based on utility value (u_sort). Then top $x\%$ VMs are selected from the sorted list. A VM (vm_k), which satisfies deadline constraint and has the latest start time, is selected to execute B_tk_i.

The working of scheduling backup copy in REO is as follows: first, it finds VM on which primary copy is not scheduled and calculates u_i^k on these VMs (lines 4 - 8). Then VM with the latest start time is found among the candidate VM set V_cand (lines 11 - 23). If the selected $x\%$ VM fails to execute B_tk_i then select the next $x\%$ VMs until $u\_sort \neq Null$ (lines 24 - 26). If scheduler fails to find an appropriate VM for B_tk_i then add new VMs (lines 28 - 30); otherwise, B_tk_i is assigned to selected VM in V_sel. If for any task tk_i, $task\_completed(tk_i) \neq 1$ then reject both P_tk_i and B_tk_i (lines 35 - 39).

Theorem 6.6. *The time complexity for scheduling backup copy in REO as discussed in Algorithm 6.7 is* $O(m^2n + nmlogm)$ *where* m *is number of VM in the system, and* n *is*

Algorithm 6.5 : P_sch (P_tk_i, vm_j)

Input: P_tk_i, vm_j
Output: Schedule for P_tk_i

$task\_competed(P\_tk_i) \leftarrow 0$;
for each task tk_i, schedule the primary copy P_tk_i **do**
  $find \leftarrow False$;
  for each VM vm_j in the system **do**
    if $stat(vm_j) == Ok \wedge vm_j^{rel} > R\_Th$ **then**
      Calculate u_i^j using Equation 5.2.
    end if
  end for
  $u\_sort \leftarrow sort(u)$; /* sort() function arrange the elements in ascending order */
  $V\_cand \leftarrow$ top $x\%$ of VM in u_{sort};
  while $u\_sort \neq Null$ **do**
    $P\_eft_i \leftarrow \infty$, $V_{sel} \leftarrow Null$;
    for each VM vm_j in V_cand **do**
      Compute $P\_ft_i^j$;
      if P_sched_tsk $(P\_ft_i^j, tk_i^{dl}) == True$ **then**
        $find \leftarrow True$;
        if $P\_ft_i^j < P\_eft_i$ **then**
          $P\_eft_i \leftarrow P\_ft_i^j$;
          $v_{sel} \leftarrow vm_j$;
          $task\_completed(P\_tk_i) = 1$;
        end if
      end if
    end for
    if $find == False$ **then**
      $V_{cand} \leftarrow$ next top $x\%$ of VM in u_sort;
    end if
  end while
  if $find == False$ **then**
    Add new VM;
  end if
  $v(P\_tk_i) \leftarrow v_{sel}$;
  $stat(v(P\_tk_i)) \leftarrow$ A_Test $(v(P\_tk_i))$;
  if $find == True$ **then**
    if $stat(v(P\_tk_i)) == Ok$ **then**
      Synch (P_tk_i);
    else
      B_Sch (B_tk_i, vm_k);
    end if
  end if
end for
for each task tk_i **do**
  if $task\_completed(P\_tk_i) \neq 1$ **then**
    Reject P_tk_i;
  end if
end for

Algorithm 6.6 : Synch $(P\_tk_i, stat(v(P\_tk_i)))$

Input: P_tk_i, $stat(v(P\_tk_i))$
Output: Schedule for P_tk_i

if $stat(v(P\_tk_i)) == Ok$ **then**
    Store the result;
    Deallocate B_tk_i;
end if

Algorithm 6.7 : B_Sch (B_tk_i, vm_k)

Input: B_tk_i, vm_k
Output: Schedule for B_tk_i

$task\_competed(B\_tk_i) \leftarrow 0$;
for each task tk_i whose P_tk_i has been allocated, schedule B_tk_i **do**
    $find \leftarrow False$;
    for each VM vm_k in $\mathcal{V} \setminus v(P\_tk_i)$ **do**
        if $stat(vm_k) == Ok \wedge vm_k^{rel} > R\_Th$ **then**
            Calculate u_i^k using Equation 5.2;
        end if
    end for
    $u\_sort \leftarrow sort(u)$;
    $V\_cand \leftarrow$ top $x\%$ of VM in u_sort;
    while $u\_sort \neq Null$ **do**
        $B\_lst_i \leftarrow -\infty, V\_sel \leftarrow Null$;
        for each VM vm_k in V_cand **do**
            Compute $B\_st_i^k$;
            if B_sched_tsk $(B\_st_i^k, tk_i^{dl}) == True$ **then**
                $find \leftarrow True$;
                if $B\_st_i^k > B\_lst_i$ **then**
                    $b\_lst_i \leftarrow B\_st_i^k$;
                    $V\_sel \leftarrow vm_k$;
                    $task\_completed(B\_tk_i) = 1$;
                end if
            end if
        end for
        if $find == False$ **then**
            $V\_cand \leftarrow$ next top $x\%$ of VM in u_sort;
        end if
    end while
    if $find == False$ **then**
        Add new VM;
    end if
    if $find == True$ **then**
        $v(B\_tk_i) \leftarrow v\_sel$;
    end if
end for
for each task tk_i **do**
    if $task\_completed(B\_tk_i) \neq 1$ **then**
        Reject P_tk_i and B_tk_i;
    end if
end for

number of task in the system.

Proof. The worst case time complexity to compute u_i^k is $O(m)$ (lines 4 - 8). The sorting process of VM based on utility value takes $O(mlogm)$ times. The worst case time complexity to determine the VM with earliest finish time for B_tk_i is $O(m^2)$ (lines 11 - 27). The rest of the lines has time complexity $O(1)$. The time complexity of the algorithm is $n(O(m) + O(mlogm) + O(m^2)) = O(mn) + O(nmlogm) + O(m^2n) = O(m^2n + nmlogm)$ □

6.4 Performance Evaluation

The proposed REO fault-tolerant scheduling algorithm is compared with the algorithms FESTAL [5] and FTESW [91], which are briefly summarized below.

- FESTAL [5]: The fault-tolerant scheduling algorithm is designed considering the PB model and BB overlapping. FESTAL considers only AEAP and ALAP mechanism, whereas, in REO, both energy consumption and reliability are considered along with AEAP and ALAP mechanism for VM selection.
- FTESW [91]: It uses the PB model but without overlapping concepts along with AEAP and ALAP mechanism to achieve fault-tolerance.

The metrics Success Ratio (SR), total energy consumption (δ), and makespan (τ) are used to evaluate the performance of REO, FTESW and FESTAL algorithms.

6.4.1 Simulation Framework

Figure 6.7 shows the simulation framework used for this work for a fault-tolerant cloud system. The task generated by the task generator is sent to the task scheduler. The task scheduler uses scheduling mechanisms and constraints to map a task to an appropriate VM in the VM pool. To realize the fault-tolerance fault injector and fault detector modules are used. The fault injector randomly selects a VM from the VM pool and generates fault instances according to a specified fault rate. The fault detector runs along with the task scheduler to detect whether a primary copy is $Faulty$ or not. It starts at the end of the execution of the primary copy. The values of parameters are listed in Table 6.1.

6.4.2 Simulation Setting

The detailed settings and parameters for simulation are given below.

- Task deadline is set as $tk_i^{dl} = tk_i^{ar} + baseD$ where $baseD$ is in Uniform distribution $U(4, 8)$.

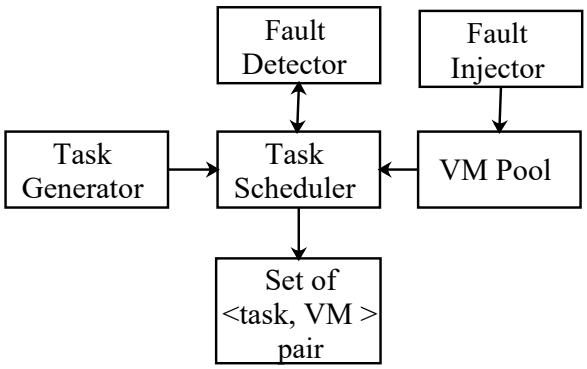


Figure 6.7: Block Diagram of Fault-tolerant Simulation Framework

Table 6.1: Parameters for Simulation Studies

Parameter	**Value (Fixed) -(Varied)**
Task Count	$1000 - (200, 400, 600, 800, 1000)$
VM Count	$80 - (40, 60, 80, 100)$
ts_{var}	$1500 - (500, 1000, 1500, 2000, 2500)$
sp_{var}	$400 - (200, 400, 600, 800, 1000)$
task arrival rate (λ)	0.05 - $(0.01, 0.03, 0.05, 0.08, 0.1)$
$baseD$	4 - $(2, 4, 6, 8)$

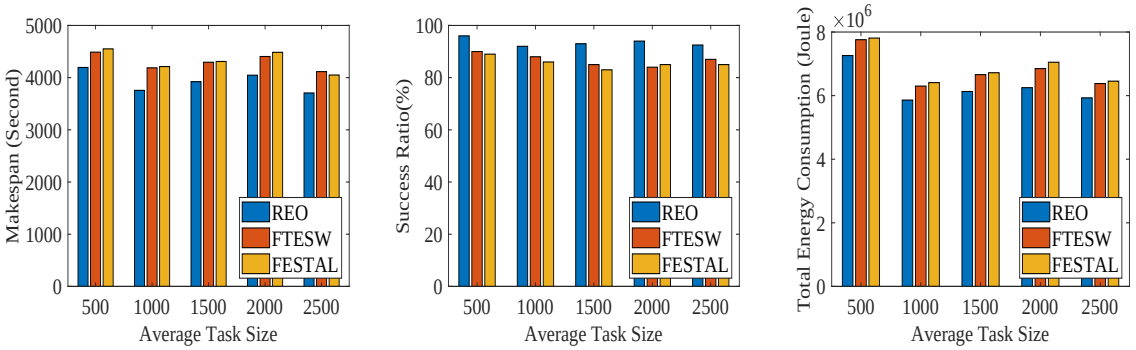
- Let task size represents the task heterogeneity. It is uniformly distributed in the range $[ts_{avg} - ts_{var}]$ and $[ts_{avg} + ts_{var}]$ where ts_{avg} is average task size and ts_{var} is variable scope taking $ts_{avg} = 5000$ MI as center respectively.
- Let VM speed indicates VM heterogeneity. It is uniformly distributed in the range $[sp_{avg} - sp_{var}]$ and $[sp_{avg} + sp_{var}]$ where sp_{avg} is average VM speed and sp_{var} is variable scope taking $sp_{avg} = 1500$ MIPS as center respectively.
- The failure rate of VM vm_j follows Uniform distribution $U(1.0 \times 10^{-6}, 3.5 \times 10^{-6})$ per hour [26].

The development of IT infrastructure and applications makes heterogeneous tasks and VM obvious in the cloud system. In this context, a set of experiments were conducted to analyze the impact of task and VM heterogeneity on system performance. Besides the effect of task and VM count, arrival rate and deadline variation on system performance are also discussed in the next section.

6.4.3 Impact of Task Heterogeneity on System Performance

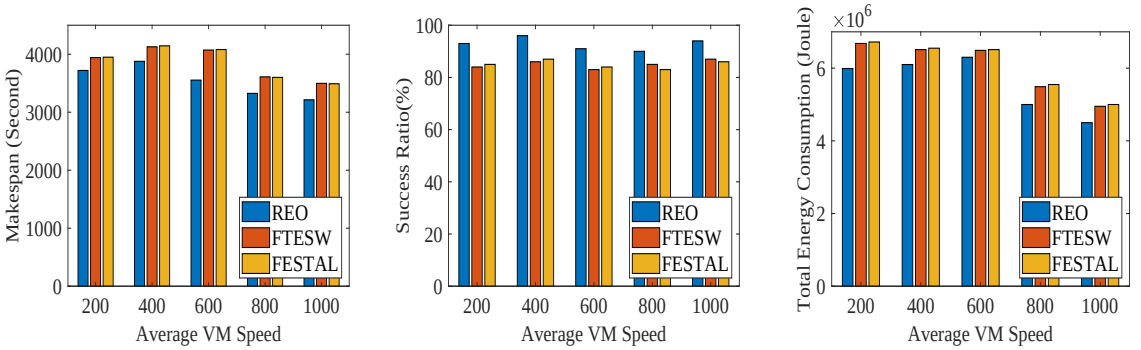
The effect of task heterogeneity on system performance is shown in Figure 6.8. If the task set has tasks whose task size is near the minimum limit, tasks have a short task execution time. Similarly, a task set with task size near the maximum limit has prolonged task execution time. The short execution time contributes to low makespan, and the long execution time increases

the makespan. Hence the simulation outcome, as shown in Figure 6.8a for makespan, has little variation. Further, the chance of a task misses its deadline is more for long execution time, reducing the Success Ratio. The Success Ratio is high because of short task execution time as it allows the task to finish execution before the deadline. So the Success Ratio due to task heterogeneity is, as given in Figure 6.8b. The short execution time contributes to low total energy consumption, whereas the long execution time increases VM's active time, which raises the total energy consumption. The effect of the heterogeneous task set on total energy consumption is shown in Figure 6.8c. However, the proposed acceptance test for VM fault detection and backup deallocation technique helps REO perform better against FTESW and FESTAL. The simulation outcome confirms that REO is adaptive to different task sizes.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 6.8: Impact of Task Heterogeneity on System Performance



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 6.9: Impact of VM Heterogeneity on System Performance

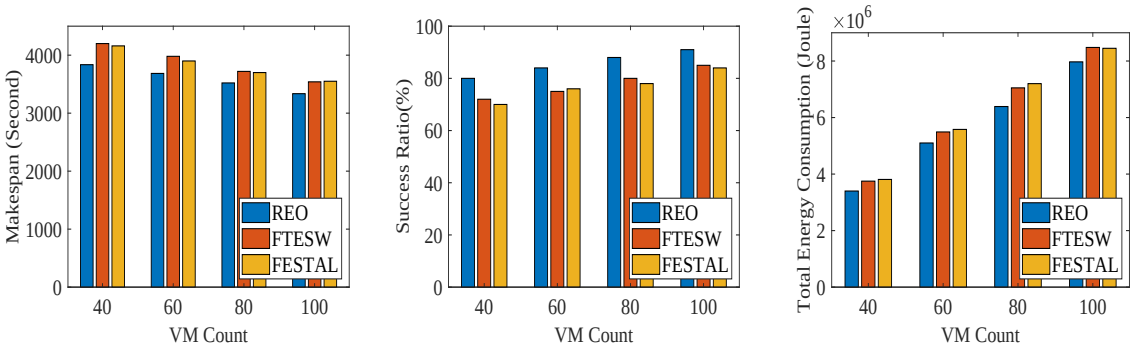
6.4.4 Impact of VM Heterogeneity on System Performance

The performance of the cloud system considering VM heterogeneity is shown in Figure 6.9. If the VM set contains low-speed VMs, tasks have long execution time, and the VM set with high-speed VMs short execution time. Therefore the makespan is stable in Figure 6.9a. The Success Ratio decreases as the prolonged execution time cause many tasks to miss their

deadline. Besides, the short task execution time results increase the Success Ratio. This is because many tasks complete their execution before the deadline due to the short execution time. The small variation in Success Ratio, as given in Figure 6.9b, confirms the effect of VM heterogeneity on it. The low-speed VM consumes less energy compared to high-speed VM. So the total energy consumption is stable for heterogeneous VM set as given in Figure 6.9c. However, the proposed acceptance test for VM fault detection and backup deallocation technique causes REO performance improvement against FTESW and FESTAL.

6.4.5 Impact of VM Count on System Performance

The impact of VM count on system performance, as given in Figure 6.10 is obtained by varying it in the range $[40 - 100]$ with a step of 20. The increase in VM count results execution of a large number of tasks on different VMs simultaneously. This reduces the total execution time and, thus, the makespan, as given in Figure 6.10a. The addition in VM count allows many tasks to execute before the deadline. Hence, there is an increase in the Success Ratio, as shown in Figure 6.10b. The total energy consumption increase as many tasks are executed with constant VM count in the system. Further, idle VM's presence due to large VM count contributes to idle energy consumption and, hence, the total energy consumption, as shown in Figure 6.10c. Besides, the simulation outcomes also show that REO performs better compared to FTESW and FESTAL.



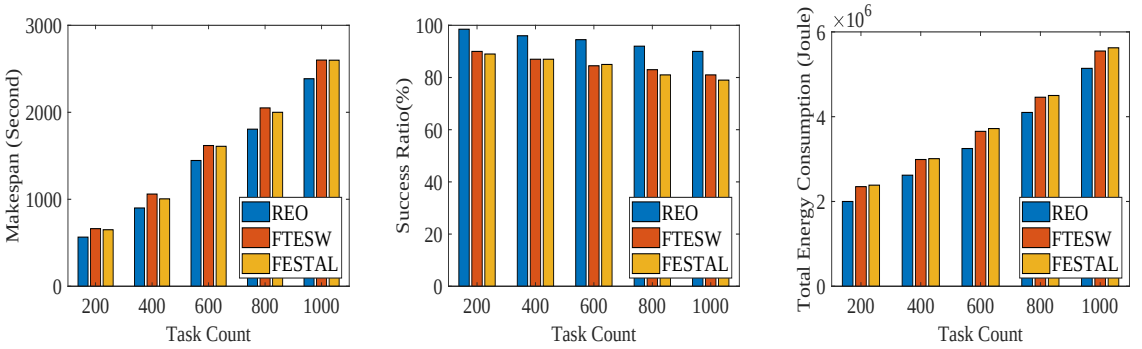
(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 6.10: Impact of VM Count on System Performance

6.4.6 Impact of Task Count on System Performance

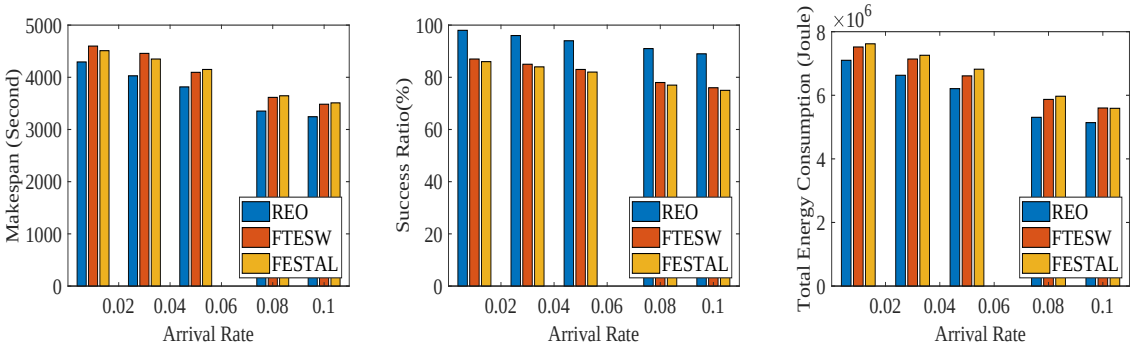
The task count is varied in the range $[200 - 1000]$ with an interval of 200. Its impact on system performance is shown in Figure 6.11. The rise in task count increases the makespan. This can be attributed to the fact that the increase in the number of tasks increases the total execution time, increasing the makespan. Hence the simulation outcome, as shown in Figure 6.11a, decreases with an increase in task count. There are chances that some tasks miss their deadline due to high task count. So the Success Ratio, as given in Figure 6.11b, decreases

for increased task count. Further, the increase in task count increases the total energy consumption because of the rise in VM's active time. Thus the total energy consumption decreases with increases in task count in Figure 6.11c. From the simulation outcomes, it can be observed that the proposed acceptance test and backup deallocation technique helps REO to achieve better performance in comparison with FTESW and FESTAL.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 6.11: Impact of Task Count on System Performance



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 6.12: Impact of Arrival Rate on System Performance

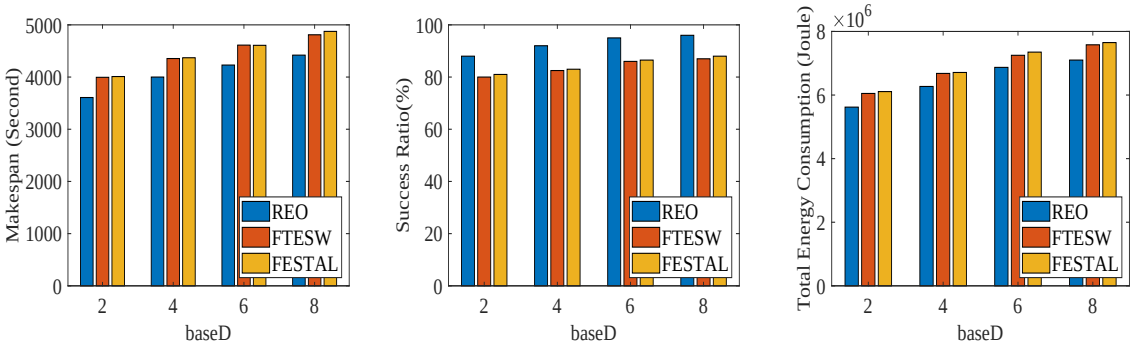
6.4.7 Impact of Arrival Rate on System Performance

Figure 6.12 shows the effect of arrival rate on system performance by varying arrival rate (λ) from 0.01 to 0.1 with step 0.02. The increase in arrival rate causes the presence of a lot of tasks within a short time, so a task gets less time to complete its execution. Since a small set of the task gets executed, makespan decreases with an increase in arrival rate as given in Figure 6.12a. Further presence of large tasks within a short time interval results in deadline misses for many tasks. Hence the Success Ratio decreases as given in Figure 6.12b with an increase in arrival rate. Besides, the reduction in the number of task execution decreases total energy consumption. As shown in Figure 6.12c for total energy consumption, the simulation result decreases with an increase in arrival rate. However, the proposed acceptance test for VM

fault detection allows REO to achieve improved system performance compared to FTESW and FESTAL.

6.4.8 Impact of Deadline Variation on System Performance

Figure 6.13 shows the effect of deadline variation on system performance. The $baseD$ value is varied in the range $[2-8]$. The rise in $baseD$ allows a large set of tasks to finish their execution, increasing the total execution time. The total execution time contributes to increasing makespan, as given in Figure 6.13a for an addition in $baseD$. Furthermore, many tasks complete their execution before the deadline because of an increase of $baseD$. So there is a rise in Success Ratio as given in Figure 6.13b. The total energy consumption increases with an addition in $baseD$, as shown in Figure 6.13c. This is because the large set of task execution increases the active time of VM. However, the proposed acceptance test helps REO complete its execution with a backup copy if a task's primary copy fails. So there is less number of task that miss the deadline or gets rejected. The simulation outcomes show the performance improvement of REO over FTESW and FESTAL.



(a) Makespan (b) Success Ratio (c) Total Energy Consumption

Figure 6.13: Impact of Deadline Variation on System Performance

6.5 Summary

This Chapter introduces a Primary-Backup based fault-tolerant scheduling algorithm to enhance system performance in terms of energy consumption and reliability. The scheduling goal is realized by the PB concept and BB overlapping mechanism. The experimental outcomes implied that the proposed PB-based approach performs better compared to existing algorithms considered for comparison. Compared to FTESW, the performance improvement of REO in Success Ratio is 10.4%, makespan is 8.7%, and the total energy consumption is 7.9%. Similarly, REO has an improvement of 10.9% in Success Ratio, 8.3% in makespan, and 8.9% in total energy consumption.

The next Chapter concludes the thesis and highlights the future direction of work.

Chapter 7
Conclusions and Future Directions

The work presented in the thesis is guided by the need for multi-objective scheduling solutions for the real-time task in a cloud system. At first, the proposed cloud system model is discussed that includes the VM model, task model, generalized scheduling model, cost model, energy model, and the reliability model, etc. Then a brief review of various real-time task scheduling techniques, considering performance metrics like energy, cost, and methodologies used, is analyzed. In the remainder of this chapter, a summary of the original contributions made in the thesis are discussed. Finally, some directions for future work are highlighted.

7.1 Contributions

This section summarizes the contributions made in this thesis.

7.1.1 VM Scoring based Approach

In this work (Chapter 3), a scheduling algorithm is proposed for real-time tasks to minimize energy consumption and execution cost simultaneously. A scoring value concept similar to the TOPSIS analysis method, is used to rank a VM for a task. The scoring function is defined in terms of energy consumption and execution cost of a task on a VM. VM selection's scoring value helps to map a task to its optimal VM by energy consumption, execution cost, and timing constraint. Besides, an Energy and Cost Aware task scheduling (ECA) algorithm is presented considering the proposed VM scoring based scheduling architecture. The simulation results show the suitability of ECA for real-time tasks compared to some existing algorithms.

7.1.2 Learning Automata-based Approach

An LA-based scheduling framework for real-time tasks is introduced in Chapter 4. Later, an LA-based Scheduling (LAS) algorithm is presented for finding the solution to a bi-objective scheduling problem that includes the minimization of energy consumption and makespan. The solution is found using a reinforcement method where task assignment is rewarded if it improves the scheduling objective; otherwise, it is penalized. The process continues until a

maximum iteration is found, and the task assignment with maximum reward is considered the best solution. Further, the LAS algorithm is explained with an example. An extensive set of experiments were conducted to show the effectiveness of LAS over its peers.

7.1.3 Game Theory based Approach

A game theory based scheduling algorithm is discussed in Chapter 5. First, the scheduling problem considers energy consumption and reliability and is modeled as a non-cooperative scheduling game, RTSG. Then Vickery auction mechanism is used to find the Nash Equilibrium that indicates the best possible solution for the RTSG game. Further, the RTSG game outcome is compared with a cooperative game-based solution and an auction-based approach. The simulation results show the effectiveness of the proposed game-based scheduling solution compared to similar approaches.

7.1.4 Primary-Backup based Approach

In Chapter 6, an acceptance test mechanism considering schedulability and response time failure for the detection of VM failure is presented. A reliability and energy-aware fault-tolerant scheduling algorithm, REO, is proposed using the PB concept and BB overlapping technique for the real-time task on a cloud system. A series of experiments on a simulated cloud environment is conducted to demonstrate REO's performance improvements. Further, the comparison of REO with some existing approaches shows its effectiveness over others.

7.1.5 Summary

As energy conservation is an essential aspect of a cloud system, in this thesis, four multi-objective scheduling solutions are proposed for the real-time task considering energy consumption as one of the objectives. Different approaches employed to find the solution includes the Multi-Objective Decision Making (MODM) technique (Chapter 3), Learning Automata (LA) (Chapter 4), game theory (Chapter 5), and Primary-Backup (PB) technique (Chapter 6). MODM technique is used to minimize the energy consumption and execution cost jointly. The bi-objective energy consumption and makespan minimization problem is solved using the LA method. Game theory based task scheduling solution is to optimize the energy consumption and reliability in a cloud system. A fault-tolerant scheduling approach is discussed using the PB technique and BB overlapping technique to optimize energy consumption and reliability.

7.2 Future Research Directions

The scope of this research is not limited to only these aspects. Still, it has room for further enhancement. Possible future research directions are outlined below.

- The proposed task model is for inter independent tasks only, but there are also instances of dependent tasks and parallel workloads. So, the proposed techniques can be extended for other task models.
- The proposed scheduling approach in Chapter 3 is designed considering only execution cost. Further, it can be extended, taking into account communication and storage cost.
- In general, a game model can be cooperative or non-cooperative. The proposed work considers a non-cooperative scheduling game, which can be further designed as a cooperative scheduling game. Here, the Vickery auction mechanism is employed to find Nash solution; researchers can use other auction mechanisms like English auction, Dutch auction, etc. to do the same.
- The proposed PB based fault-tolerant approach can be extended to support multiple VM faults. Further, other fault tolerance techniques like a checkpoint, resubmission, etc. can be explored for real-time task scheduling in the cloud system.
- Reinforcement learning helps to gradually learn from the environment through reward and penalty for action. Further, in recent times, machine learning is a popular technique used in multidisciplinary fields. These learning methods can be explored for multi-objective real-time task scheduling problem.

References

[1] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 168–180, 2014.

[2] H. Chen, X. Zhu, D. Qiu, H. Guo, L. T. Yang, and P. Lu, "Eons: minimizing energy consumption for executing real-time workflows in virtualized cloud data centers," in *45th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 2016, pp. 385–392.

[3] S. K. Mishra, D. Puthal, J. J. Rodrigues, B. Sahoo, and E. Dutkiewicz, "Sustainable service allocation using a metaheuristic technique in a fog server for industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4497–4506, 2018.

[4] Y. Gao, Y. Wang, S. K. Gupta, and M. Pedram, "An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2013, p. 31.

[5] J. Wang, W. Bao, X. Zhu, L. T. Yang, and Y. Xiang, "Festal: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds," *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2545–2558, 2014.

[6] X. Qiu, Y. Dai, Y. Xiang, and L. Xing, "A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 401–412, 2015.

[7] X.-Q. Pham, N. D. Man, N. D. T. Tri, N. Q. Thai, and E.-N. Huh, "A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing," *International Journal of Distributed Sensor Networks*, vol. 13, no. 11, pp. 1–16, 2017.

[8] J. Koomey *et al.*, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, vol. 9, p. 161, 2011.

[9] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic energy-aware scheduling for parallel task-based application in cloud computing," *Future Generation Computer Systems*, vol. 78, pp. 257–271, 2018.

[10] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo, "Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 713–726, 2015.

[11] H. Han, W. Bao, X. Zhu, X. Feng, and W. Zhou, "Fault-tolerant scheduling for hybrid real-time tasks based on cpb model in cloud," *IEEE Access*, vol. 6, pp. 18 616–18 629, 2018.

[12] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 423–430.

[13] H. Yan, X. Zhu, H. Chen, H. Guo, W. Zhou, and W. Bao, "Deft: Dynamic fault-tolerant elastic scheduling for tasks with uncertain runtime in cloud," *Information Sciences*, vol. 477, pp. 30–46, 2019.

[14] Y. Ding, X. Qin, L. Liu, and T. Wang, "Energy efficient scheduling of virtual machines in cloud with deadline constraint," *Future Generation Computer Systems*, vol. 50, pp. 62–74, 2015.

[15] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, qos-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing dvfs and approximate computations," *Future Generation Computer Systems*, vol. 96, pp. 216–226, 2019.

[16] S. El Kafhali and K. Salah, "Efficient and dynamic scaling of fog nodes for iot devices," *The Journal of Supercomputing*, vol. 73, no. 12, pp. 5261–5284, 2017.

[17] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[18] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[19] M. Velasquez and P. T. Hester, "An analysis of multi-criteria decision making methods," *International Journal of Operations Research*, vol. 10, no. 2, pp. 56–66, 2013.

[20] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[21] N. Ranganathan and A. K. Murugavel, "A low power scheduler using game theory," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2003, pp. 126–131.

[22] J. N. Webb, *Game theory: decisions, interaction and Evolution*. Springer Science & Business Media, 2007.

[23] G. Baranwal, D. Kumar, Z. Raza, and D. P. Vidyarthi, *Auction Based Resource Provisioning in Cloud Computing*. Springer, 2018.

[24] R. A. Feldman and R. Mehra, "Auctions: Theory and applications," *Staff Papers*, vol. 40, no. 3, pp. 485–511, 1993.

[25] X. Zhu, C. He, R. Ge, and P. Lu, "Boosting adaptivity of fault-tolerant scheduling for real-time tasks with service requirements on clusters," *Journal of Systems and Software*, vol. 84, no. 10, pp. 1708–1716, 2011.

[26] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Computing*, vol. 32, no. 5-6, pp. 331–356, 2006.

[27] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 380–393, 2008.

[28] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3501–3517, 2016.

[29] J. Kołodziej, S. U. Khan, L. Wang, and A. Y. Zomaya, "Energy efficient genetic-based schedulers in computational grids," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 809–829, 2015.

[30] N. Quang-Hung, P. D. Nien, N. H. Nam, N. H. Tuong, and N. Thoai, "A genetic algorithm for power-aware virtual machine allocation in private cloud," in *Information and communication technology-EurAsia conference*. Springer, 2013, pp. 183–191.

[31] A. Rehman, S. S. Hussain, Z. ur Rehman, S. Zia, and S. Shamshirband, "Multi-objective approach of energy efficient workflow scheduling in cloud environments," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 8, pp. 1–20, 2019.

[32] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, "Multi-objective approach for energy-aware workflow scheduling in cloud computing environments," *The Scientific World Journal*, vol. 2013, 2013.

[33] P. Guo and Z. Xue, "Cost-effective fault-tolerant scheduling algorithm for real-time tasks in cloud systems," in *17th International Conference on Communication Technology (ICCT)*. IEEE, 2017, pp. 1942–1946.

[34] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Transactions on Services Computing*, 2018, doi:10.1109/TSC.2018.2866421.

[35] S. Sahoo, B. Sahoo, and A. K. Turuk, "A learning automata-based scheduling for deadline sensitive task in the cloud," *IEEE Transactions on Services Computing*, 2019, doi: 10.1109/TSC.2019.2906870.

[36] K. Li, Y. Wang, and M. Liu, "A non-cooperative game model for reliability-based task scheduling in cloud computing," *arXiv preprint arXiv:1403.5012*, 2014.

[37] J. Yang, B. Jiang, Z. Lv, and K.-K. R. Choo, "A task scheduling algorithm considering game theory designed for energy management in cloud computing," *Future Generation Computer Systems*, 2017, doi: 10.1016/j.future.2017.03.024.

[38] Y. Zhang, X. Cheng, L. Chen, and H. Shen, "Energy-efficient tasks scheduling heuristics with multi-constraints in virtualized clouds," *Journal of Grid Computing*, vol. 16, no. 3, pp. 459–475, 2018.

[39] Z. Hu, B. Li, and J. Luo, "Time-and cost-efficient task scheduling across geo-distributed data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 705–718, 2017.

[40] E. Hwang and K. H. Kim, "Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud," in *Proceedings of the ACM/IEEE 13th International Conference on Grid Computing*. IEEE Computer Society, 2012, pp. 130–138.

[41] T. Shi, M. Yang, X. Li, Q. Lei, and Y. Jiang, "An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds," *Pervasive and Mobile Computing*, vol. 27, pp. 90–105, 2016.

[42] E. Grochowski and M. Annavaram, "Energy per instruction trends in intel microprocessors," *Technology@ Intel Magazine*, vol. 4, no. 3, pp. 1–8, 2006.

[43] K. Weins, "Cloud pricing comparison: Aws vs. microsoft azure vs. google cloud vs. ibm cloud," Tech. Rep., 2017, available [Online] http://www.infoworld.com/article/3237566/cloud-computing/cloud-pricing-comparison-aws-vs-azure-vs-google-vs-ibm.html.

[44] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on Parallel & Distributed Systems*, no. 5, pp. 936–943, 2011.

[45] J. Vilaplana, F. Solsona, I. Teixido, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.

[46] D. Bruneo, "A stochastic model to investigate data center performance and qos in iaas cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 560–569, 2014.

[47] X. Liu, S. Li, and W. Tong, "A queuing model considering resources sharing for cloud service performance," *The Journal of Supercomputing*, vol. 71, no. 11, pp. 4042–4055, 2015.

[48] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing resource allocation in three-tier iot fog networks: A joint optimization approach combining stackelberg game and matching," *IEEE Internet of Things Jornal*, vol. 4, no. 5, pp. 1204–1215, 2017.

[49] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li, "Profit maximization for cloud brokers in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 190–203, 2019.

[50] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based iot," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.

[51] H. Khazaei, J. Misic, and V. B. Misic, "A fine-grained performance model of cloud computing centers," *IEEE Transactions on parallel and distributed systems*, vol. 24, no. 11, pp. 2138–2147, 2013.

[52] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Modeling and performance analysis of large scale iaas clouds," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1216–1234, 2013.

[53] S. Sahoo, S. K. Mishra, B. Sahoo, D. Puthal, and M. S. Obaidat, "Deadline-constraint services in cloud with heterogeneous servers," in *International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2017, pp. 20–24.

[54] H. Chen, G. Liu, S. Yin, X. Liu, and D. Qiu, "Erect: energy-efficient reactive scheduling for real-time tasks in heterogeneous virtualized clouds," *Journal of computational science*, vol. 28, pp. 416–425, 2018.

[55] K. Li, "Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment," *Future generation computer systems*, vol. 82, pp. 591–605, 2018.

[56] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 772–783, 2018.

[57] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[58] G. Xing, X. Xu, H. Xiang, S. Xue, S. Ji, and J. Yang, "Fair energy-efficient virtual machine scheduling for internet of things applications in cloud environment," *International Journal of Distributed Sensor Networks*, vol. 13, no. 2, p. 1550147717694890, 2017.

[59] Z. Dong, N. Liu, and R. Rojas-Cessa, "Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers," *Journal of Cloud Computing*, vol. 4, no. 1, p. 5, 2015.

[60] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 510–519.

[61] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of virtual machines for real-time cloud services," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.

[62] G. Xie, G. Zeng, R. Li, and K. Li, "Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 62–75, 2017.

[63] K. Kathe and U. A. Deshpande, "A thermal aware routing algorithm for a wireless body area network," *Wireless Personal Communications*, vol. 105, no. 4, pp. 1353–1380, 2019.

[64] M. K. Gupta and T. Amgoth, "Resource-aware virtual machine placement algorithm for iaas cloud," *The Journal of Supercomputing*, vol. 74, no. 1, pp. 122–140, 2018.

[65] P. Arroba, J. L. Risco-Martín, J. M. Moya, and J. L. Ayala, "Heuristics and metaheuristics for dynamic management of computing and cooling energy in cloud data centers," *Software: Practice and Experience*, vol. 48, no. 10, pp. 1775–1804, 2018.

[66] Z. Cai, X. Li, R. Ruiz, and Q. Li, "A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds," *Future Generation Computer Systems*, vol. 71, pp. 57–72, 2017.

[67] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and cost trade-off management for scheduling parallel applications on utility grids," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1344–1355, 2010.

[68] K. Li, X. Tang, and Q. Yin, "Energy-aware scheduling algorithm for task execution cycles with normal distribution on heterogeneous computing systems," in *41st International Conference on Parallel Processing*. IEEE, 2012, pp. 40–47.

[69] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," in *28th international conference on advanced information networking and applications*. IEEE, 2014, pp. 858–865.

[70] Y. Ran, J. Yang, S. Zhang, and H. Xi, "Dynamic iaas computing resource provisioning strategy with qos constraint," *IEEE Transactions on Services Computing*, vol. 10, no. 2, pp. 190–202, 2015.

[71] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *Journal of Systems and Software*, vol. 113, pp. 1–26, 2016.

[72] K. Gai, M. Qiu, H. Zhao, and X. Sun, "Resource management in sustainable cyber-physical systems using heterogeneous cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 3, no. 2, pp. 60–72, 2017.

[73] X. Zeng, S. K. Garg, Z. Wen, P. Strazdins, A. Y. Zomaya, and R. Ranjan, "Cost efficient scheduling of mapreduce applications on public clouds," *Journal of computational science*, vol. 26, pp. 375–388, 2018.

[74] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015.

[75] G. L. Stavrinides and H. D. Karatza, "A cost-effective and qos-aware approach to scheduling real-time workflow applications in paas and saas clouds," in *3rd International Conference on Future Internet of Things and Cloud*. IEEE, 2015, pp. 231–239.

[76] S. Sahoo, S. Nawaz, S. K. Mishra, and B. Sahoo, "Execution of real time task on cloud environment," in *Annual IEEE India Conference (INDICON)*. IEEE, 2015, pp. 1–5.

[77] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Computing*, vol. 39, no. 4-5, pp. 177–188, 2013.

[78] C. Za'in, M. Pratama, E. Lughofer, M. Ferdaus, Q. Cai, and M. Prasad, "Big data analytics based on panfis mapreduce," *Procedia Computer Science*, vol. 144, pp. 140–152, 2018.

[79] S. Mini, S. K. Udgata, and S. L. Sabat, "Sensor deployment and scheduling for target coverage problem in wireless sensor networks," *IEEE sensors journal*, vol. 14, no. 3, pp. 636–644, 2013.

[80] K. S. Narendra and M. A. Thathachar, "Learning automata-a survey," *IEEE Transactions on systems, man, and cybernetics*, no. 4, pp. 323–334, 1974.

[81] M. Thathachar and B. R. Harita, "Learning automata with changing number of actions," *IEEE transactions on systems, man, and cybernetics*, vol. 17, no. 6, pp. 1095–1100, 1987.

[82] S. Misra, P. V. Krishna, K. Kalaiselvan, V. Saritha, and M. S. Obaidat, "Learning automata-based qos framework for cloud iaas," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 15–24, 2014.

[83] A. Rezvanian and M. R. Meybodi, "Finding minimum vertex covering in stochastic graphs: a learning automata approach," *Cybernetics and Systems*, vol. 46, no. 8, pp. 698–727, 2015.

[84] M. Ranjbari and J. A. Torkestani, "A learning automata-based algorithm for energy and sla efficient consolidation of virtual machines in cloud data centers," *Journal of Parallel and Distributed Computing*, vol. 113, pp. 55–62, 2018.

[85] A. A. Rahmanian, M. Ghobaei-Arani, and S. Tofighy, "A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment," *Future Generation Computer Systems*, vol. 79, pp. 54–71, 2018.

[86] J. A. Torkestani, "An adaptive learning to rank algorithm: Learning automata approach," *Decision Support Systems*, vol. 54, no. 1, pp. 574–583, 2012.

[87] R. D. Venkataramana and N. Ranganathan, "A learning automata based framework for task assignment in heterogeneous computing systems," in *Proceedings of the ACM symposium on Applied computing*. Citeseer, 1999, pp. 541–547.

[88] S. Sahoo, B. Sahoo, and A. K. Turuk, "An energy-efficient scheduling framework for cloud using learning automata," in *9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2018, pp. 1–5.

[89] G. Xie, G. Zeng, R. Li, and K. Li, "Quantitative fault-tolerance for reliable workflows on heterogeneous iaas clouds," *IEEE Transactions on Cloud Computing*, 2017, doi: 10.1109/TCC.2017.2780098.

[90] M. S. A. Latiff, S. H. H. Madni, M. Abdullahi *et al.*, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," *Neural Computing and Applications*, vol. 29, no. 1, pp. 279–293, 2018.

[91] Y. Ding, G. Yao, and K. Hao, "Fault-tolerant elastic scheduling algorithm for workflow in cloud systems," *Information Sciences*, vol. 393, pp. 47–65, 2017.

[92] S. Ghosh, R. Melhem, and D. Mossé, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Transactions on Parallel and distributed systems*, vol. 8, no. 3, pp. 272–284, 1997.

[93] M. Smara, M. Aliouat, A.-S. K. Pathan, and Z. Aliouat, "Acceptance test for fault detection in component-based cloud computing and systems," *Future Generation Computer Systems*, vol. 70, pp. 74–93, 2017.

[94] M. Gabel, R. Gilad-Bachrach, N. Bjorner, and A. Schuster, "Latent fault detection in cloud services," *Microsoft Re-search, Tech. Rep. MSR-TR-2011-83*, 2011.

[95] D.-M. Bui, S. Lee *et al.*, "Early fault detection in iaas cloud computing based on fuzzy logic and prediction technique," *The Journal of Supercomputing*, vol. 74, no. 11, pp. 5730–5745, 2018.

[96] A. V. Nimkar and S. K. Ghosh, "A security framework for virtual resource management in horizontal iaas federation," in *Advanced Computing, Networking and Informatics-Volume 2*. Springer, 2014, pp. 241–247.

[97] G. Fan, L. Chen, H. Yu, and D. Liu, "Modeling and analyzing dynamic fault-tolerant strategy for deadline constrained task scheduling in cloud computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–15, 2017.

[98] Z. Wen, J. Cała, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 929–941, 2016.

[99] X. Xiao, G. Xie, C. Xu, C. Fan, R. Li, and K. Li, "Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems," *Journal of computational science*, vol. 26, pp. 344–353, 2018.

[100] C. Li, J. Tang, and Y. Luo, "Cost-aware scheduling for ensuring software performance and reliability under heterogeneous workloads of hybrid cloud," *Automated Software Engineering*, vol. 26, no. 1, pp. 125–159, 2019.

[101] S. S. M. Nik, M. Naghibzadeh, and Y. Sedaghat, "Cost-driven workflow scheduling on the cloud with deadline and reliability constraints," *Computing*, pp. 1–24, 2019.

[102] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE transactions on Computers*, vol. 60, no. 6, pp. 800–812, 2011.

[103] R. Rathnayake, M. S. Karunarathne, N. S. Nafi, and M. A. Gregory, "Cloud enabled solution for privacy concerns in internet of medical things," in *28th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2018, pp. 1–4.

[104] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 380–393, 2008.

[105] C. Qu, R. N. Calheiros, and R. Buyya, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances," *Journal of Network and Computer Applications*, vol. 65, pp. 167–180, 2016.

[106] T. Guo, J. Liu, W. Hu, and M. Wei, "Energy-aware fault-tolerant scheduling under reliability and time constraints in heterogeneous systems," in *International Conference on Intelligent Computing*. Springer, 2018, pp. 36–46.

[107] P. Sun, Y. Dai, and X. Qiu, "Optimal scheduling and management on correlating reliability, performance, and energy consumption for multiagent cloud systems," *IEEE Transactions on Reliability*, vol. 66, no. 2, pp. 547–558, 2017.

[108] S. Malik, F. Huet, and D. Caromel, "Reliability aware scheduling in cloud computing," in *International Conference for Internet Technology and Secured Transactions*. IEEE, 2012, pp. 194–200.

[109] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, pp. 1137–1152, 1998.

[110] R. Al-Omari, G. Manimaran, and A. K. Somani, "An efficient backup-overloading for fault-tolerant scheduling of real-time tasks," in *International Parallel and Distributed Processing Symposium*. Springer, 2000, pp. 1291–1295.

[111] S. U. Khan and I. Ahmad, "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 346–360, 2008.

[112] D. Ye and J. Chen, "Non-cooperative games on multidimensional resource allocation," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1345–1352, 2013.

[113] K. Li, C. Liu, and K. Li, "An approximation algorithm based on game theory for scheduling simple linear deteriorating jobs," *Theoretical Computer Science*, vol. 543, pp. 46–51, 2014.

[114] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[115] T. T. Huu and C.-K. Tham, "An auction-based resource allocation model for green cloud computing," in *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2013, pp. 269–278.

[116] X. Wu, M. Liu, W. Dou, L. Gao, and S. Yu, "A scalable and automatic mechanism for resource allocation in self-organizing cloud," *Peer-to-peer networking and applications*, vol. 9, no. 1, pp. 28–41, 2016.

[117] Z. Su, Q. Xu, M. Fei, and M. Dong, "Game theoretic resource allocation in media cloud with mobile social users," *IEEE Transactions on Multimedia*, vol. 18, no. 8, pp. 1650–1660, 2016.

[118] L. Ding, L. Chang, and L. Wang, "Online auction-based resource scheduling in grid computing networks," *International Journal of Distributed Sensor Networks*, vol. 12, no. 10, pp. 1–12, 2016.

[119] W. Wang, Y. Jiang, and W. Wu, "Multiagent-based resource allocation for energy minimization in cloud computing systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 2, pp. 205–220, 2016.

[120] A. Bandyopadhyay, F. Xhafa, and S. Mukhopadhyay, "An auction framework for daas in cloud computing," in *International Conference on Emerging Internetworking, Data & Web Technologies*. Springer, 2018, pp. 732–741.

[121] G. Zhou, P. Jiang, and G. Q. Huang, "A game-theory approach for job scheduling in networked manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 9-10, pp. 972–985, 2009.

[122] S. U. Khan and I. Ahmad, "Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006.

[123] S. Das, S. Misra, M. Khatua, and J. J. Rodrigues, "Mapping of sensor nodes with servers in a mobile health-cloud environment," in *15th International Conference on e-Health Networking, Applications and Services (Healthcom 2013)*. IEEE, 2013, pp. 481–485.

[124] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–12, 2019, doi: 10.1109/TPDS.2019.2938944.

[125] N. Sorensen, "Industry Benchmarks Performance," https://www.cisco.com/c/dam/global/da_dk/assets/docs/presentations/vBootcamp_Performance_Benchmark.pdf, 2011, [Online].

Dissemination

Internationally indexed journals (*SCI, SCIE*)

1. **Sampa Sahoo**, Bibhudatta Sahoo, and Ashok Kumar Turuk. "A Learning Automata-based Scheduling for Deadline Sensitive Task in The Cloud." IEEE Transactions on Services Computing, https://doi.org/10.1109/TSC.2019.2906870, 2019.

2. **Sampa Sahoo**, Bibhudatta Sahoo, and Ashok Kumar Turuk. "An eigenvalue-based edge infrastructure for cloud-based CDN." International Journal of Communication Systems, Wiley, https://doi.org/10.1002/dac.3966, 2019.

Book Chapters

1. **Sampa Sahoo**, Bibhudatta Sahoo, and Ashok Kumar Turuk. "Video transcoding services in cloud computing environment." In Cloud Computing for Optimization: Foundations, Applications, and Challenges, pp. 417-433. Springer, Cham, 2018.

2. **Sampa Sahoo**, Sambit Kumar Mishra, Bibhudatta Sahoo, and Ashok Kumar Turuk. "Co-resident Attack in Cloud Computing: An Overview." In Encyclopedia of Big Data Technologies, Springer, https://doi.org/10.1007/978-3-319-63962-8_322-1, 2018.

3. **Sampa Sahoo**, Bibhudatta Sahoo, Ashok Kumar Turuk, and Sambit Kumar Mishra. "Real time task execution in cloud using mapreduce framework." In Resource Management and Efficiency in Cloud Computing Environments, pp. 190-209. IGI Global, 2017.

Conferences

1. **Sampa Sahoo**, Bibhudatta Sahoo and Ashok Kumar Turuk. "MCSA: A Multi-constraint Scheduling Algorithm for Real-time Task in Virtualized Cloud." In 2018 Annual IEEE India Conference (INDICON), 2018. [**Accepted**]

2. **Sampa Sahoo**, Sahil Kumar Sahu, Tanmay Kumar Rath, Bibhudatta Sahoo, and Ashok Kumar Turuk. "TCA: A multi constraint real-time task scheduling algorithm for heterogeneous cloud environment." In 2018 International Conference on Information Technology (ICIT), pp. 132-136. IEEE, 2018.

3. **Sampa Sahoo**, Bibhudatta Sahoo, and Ashok Kumar Turuk. "An Energy-Efficient Scheduling Framework for Cloud Using Learning Automata." In 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1-5. IEEE, 2018.

4. **Sampa Sahoo**, Maneesha Nidhi, Kshira Sagar Sahoo, Bibhudatta Sahoo, and Ashok Kumar Turuk. "Video delivery services in media cloud with abandonment: An analytical approach." In 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), pp. 1-6. IEEE, 2017.

5. **Sampa Sahoo**, Bibhudatta Sahoo, and Ashok Kumar Turuk. "RT-PUSH: a VM fault detector for deadline-based tasks in cloud." In Proceedings of the 3rd International Conference on Communication and Information Processing, pp. 196-201. ACM, 2017.

6. **Sampa Sahoo**, Syed Nawaz, Sambit Kumar Mishra, and Bibhudatta Sahoo. "Execution of real time task on cloud environment." In 2015 Annual IEEE India Conference (INDICON), pp. 1-5. IEEE, 2015.