A FAULT-TOLERANT HYBRID RESOURCE PROVISIONING MODEL FOR CLOUD COMPUTING

Thesis Submitted for the Award of the Degree of

DOCTOR OF PHILOSOPHY

in

Computer Application

By

Sheikh Umar Mushtaq

Registration Number: 12021177

Supervised By

Dr. Sophiya sheikh (26298)

Associate Professor, School of Computer Application,

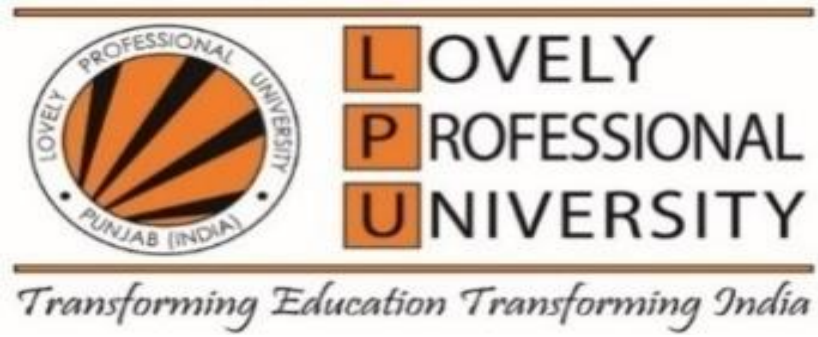
Lovely Professional University, Punjab

Co- Supervised By

Dr. Sheikh Mohammad Idrees

Researcher, Norwegian University of Science and Technology,

Norway



LOVELY PROFESSIONAL UNIVERSITY, PUNJAB

2025

DECLARATION

I, hereby declared that the presented work in the thesis entitled, "A Fault-tolerant Hybrid Resource Provisioning Model for Cloud Computing" in fulfillment of degree of **Doctor of Philosophy (Ph.D.)** is the outcome of research work carried out by me under the supervision of Dr. Sophiya sheikh, working as Associate Professor in Computer Application, in the School of Computer Application of Lovely Professional University, Punjab, India. In keeping with the general practice of reporting scientific observations, due acknowledgements have been made whenever the work described here has been based on the findings of another investigator. This work has not been submitted in part or full to any other University or Institute for the award of any degree.

(Signature of Scholar)
Sheikh Umar Mushtaq
Registration No.:12021177
Department of Computer Applications,
School of Computer Application,
Lovely Professional University, Punjab, India

CERTIFICATE

This is to certify that the work reported in the Ph. D. thesis entitled, "A Fault-tolerant Hybrid Resource Provisioning Model for Cloud Computing" submitted in fulfillment of the requirement for the reward of degree of **Doctor of Philosophy (Ph.D.)** in the School of Computer Application of Lovely Professional University, Punjab, India, is a research work carried out by Sheikh Umar Mushtaq, Registration No. 12021177, is Bonafide record of her original work carried out under my supervision and that no part of thesis has been submitted for any other degree, diploma or equivalent course.

(Signature of Supervisor)

Dr. Sophiya Sheikh

Associate Professor

Department of Computer Applications,

School of Computer Application,

Lovely Professional University, Punjab, India

ACKNOWLEDGEMENT

This thesis marks the culmination of a journey that has been both challenging and rewarding. It would not have been possible without the support, guidance, and encouragement of many individuals to whom I am deeply indebted.

First and foremost, I would like to express my deepest gratitude to Allah, the Most Gracious, the Most Merciful, for granting me the strength, knowledge, and perseverance to complete this thesis.

I extend my deepest gratitude to my Supervisor, ***Dr Sophiya Sheikh*** *whose wisdom, expertise, and unwavering support have been the cornerstone of this work. Dr Sophiya Sheikh, your insightful feedback, patience, and encouragement have guided me through every step of this research. I am profoundly grateful for the countless hours you dedicated to helping me shape this thesis.*

I am with immense gratitude to my co-supervisor, ***Dr Sheikh Mohammad Idrees*** *from Department of Computer Science (IDI), Norwegian University of Science and Technology, Gjøvik, 2815, Norway.*

My gratitude extends to ***Dr. Manmohan Sharma, and Dr. Amar Singh*** *for their tremendous advice and support during the duration of my research. I am also grateful to* ***Mr. Ajay Kumar Bansal****, HOD, School of Computer Applications, Lovely Professional University, Punjab, as their leadership has created an inspiring and conducive environment for research and learning, which has been instrumental in the successful completion of this thesis. I am truly grateful for their unwavering support and the opportunities they have provided. I would like to express my heartfelt appreciation to all the faculty members of the School of Computer Applications, Lovely Professional University, Punjab.*

To my RAC members, your constructive critiques and invaluable suggestions have significantly enriched this research. Your diverse perspectives and expertise have been instrumental in refining my ideas and pushing the boundaries of my work.

I am also deeply thankful to my colleagues and friends in Lovely Professional University. Your camaraderie, collaborative spirit, and intellectual exchange have made this journey enjoyable and intellectually stimulating. Special thanks to my roommate, Mohammad Irfan ul Haq whose friendship and support have been a source of strength and motivation.

I would especially like to thank my research mates Umar Bashir, Dr Talib Iqbal, Dr Qurat ul Ain, Ajay Nain, Rohit Malik, Ayaz Ahmad, Pawan, and Parvaz Ahmad Malla. It would

not have been possible to carry out this research without their emotional support and invaluable assistance.

A heartfelt thanks to my family, whose unwavering love and belief in me have been my foundation. To my parents, who instilled in me the values of perseverance and hard work, your patience, love, and constant support have been my anchor. Your belief in me, even when I doubted myself, gave me the courage to persevere. Thank you for being my rock through this journey. To my siblings, for their constant encouragement and understanding, I owe you everything. Your sacrifices and endless support have made this achievement possible.

Lastly, I dedicate this thesis to my beloved family, ***Mushtaq Ahmad Sheikh (Father)****,* ***Rafiqa Akhter (Mother), Sheikh Mayesser Mushtaq (Brother), Dr Qurat ul Ain (Wife)*** *whose inspiration continues to guide me. Your legacy of passion and dedication is a constant reminder of the values I strive to uphold. This journey has been a mosaic of countless moments of learning, perseverance, and growth.*

To everyone who has been a part of this journey, thank you for your faith, support, and encouragement.

This thesis is as much yours as it is mine

Abstract

The ongoing development of vast integration in various computational units and connected components has made it possible to achieve economic performance that was difficult to achieve with independent resources. Cloud computing is formed by the integration of these independent, and dispersed resources technically termed virtual machines (VMs). In addition, the grouping of these geographically scattered and heterogeneous VMs that can be shared among several end-users is termed a cloud environment. Additionally, the dynamic nature of the cloud may also make the entire system more prone to errors and faults since the scheduler lacks precise knowledge of the incoming tasks. It is also evident that in a dynamic environment, any of the accessible VMs might stop working or exit the system at any time. Besides, a reliable cloud system also requires an effective failure management approach known as fault tolerance. Apart from this, a cloud is only regarded as efficient if it can make the best use of its resources. For this purpose, numerous scheduling and load-balancing algorithms were introduced in the literature. Still, effective load balancing is one of the main issues in any computational environment. It is important to achieve an acceptable resource allocation across the computing resources so that the execution of the task is completed on time. Furthermore, the dynamic and adaptive adjustments implemented for reallocating to mitigate the risks and to ensure uninterrupted services could often result in uneven utilization of resources in hand. Hence, ensuring monitored equilibrium between fault tolerance and load distribution demands meticulous attention to avoid unintended influences and associated overheads. Addressing this issue of non-uniform load distribution, fault tolerance needs the support of effective load balancing. The main goal of this research is to provide a hybrid resource provisioning model that not only focuses on fault-tolerant scheduling but also integrates it with an efficient load balancing technique to reduce the fault overheads. Since the cloud environment is dynamic, several issues might develop in accomplishing the identified problem. To address these issues, we have introduced some fault-tolerant algorithms incorporating resource provisioning that adapt load balancing for efficient QoS optimization.

The thesis starts with an Introduction chapter, bordering the motivation, problem statement, and objectives of the study. A thorough Literature Review follows, reviewing existing scheduling with fault-tolerant and load-balancing techniques while recognizing gaps in existing investigations. To enhance fault tolerance, RFRTS model is introduced in Chapter 3, aiming to maintain the system reliability and makespan. Building upon this, HFSLM is proposed in Chapter 4, which integrates dynamic load balancing with fault tolerance to optimize makespan, resource utilization and fault overheads. HFSLM is evaluated by

comparing it with FTHRM, MAX MIN, MINMIN, OLB, ELISA, and MELISA on both small and large task scales having task ranges from 250 to 1000 and 10000 to 50000 respectively. Further, the associated fault overheads have been reduced by accompanying the model with a load balancing strategy. The associated overheads have been compared after fault tolerance and after load balancing. A notable reduction in the associated overheads was observed following the implementation of load balancing. Further, CRFTS model proposed in chapter 5, which incorporates clustering mechanisms for improved task-to-VM mapping. The scheduling has been performed by using the clustering approach and the fault tolerance has been performed by reserving the nearest neighboring VM for the affected task. CRFTS is evaluated by comparing it with HEFT, FTSA-1, DBSA, E-HEFT, and LB-HEFT while varying task number from 25 to 1000. The thesis concludes in Chapter 6 with key findings, contributions, and future roadmap and research directions. This work contributes to the advancement of cloud fault tolerance and load balancing, providing scalable and efficient scheduling solutions for dynamic cloud environments.

Keywords: Advance Resource Reservation, Dynamism, QoS Parameter, Load Balancing, Fault-Tolerance, Resource Utilization, Makespan

List of Tables

List of Figures

List of Notions

Symbol	**Meaning**
• T	Set of Tasks
• N	Number of Tasks at any Instance
• V	Set of Virtual Machines
• m	Number of VMs at any Instance
• t_id	Task id
• t_size /L	Task Size/Task Length
• V_id	Virtual Machine id
• VM_f / V_f	Set of Failed Virtual Machines
• S	Speed of VM
• $W(t_i)$	Weight/Length of t_i
• MIPS	Million Instruction Per Second
• M/ μ	Mapping Between T and V

- AR — Advance Reservation
- RT_j — Ready time of VM
- TET_j — Total Execution Time of VM
- EST_{ij} — Early Start Time of t_i on v_j
- AR slot — Advance Reservation Slot
- T_u / T_f — Set of Unexecuted or Failed Task
- t_u / t_f — Unexecuted Task or Failed task
- UT — Average Resource Utilization
- ARM — Advance Reservation Matrix
- LB — Load Balancer
- T_r — Ranked Task Set
- $CT(t_m,v_k)$ — Completion Time of t_m on v_k
- $R(t_m,v_k)$ — Reservation Window of t_m on v_k
- TET — Total Execution Time
- C^l — Low Cluster
- C^m — Mid Cluster
- C^h — High Cluster
- AFT_{ij} — Actual Finish Time of t_i on v_j
- $t_p(t_i, v_j)$ / $E(t_i, v_j)$ / $PT(t_m)$/ $P(t_i, v_j)$ — Total Processing Time of t_i on v_j
- $o(VM_f)$ — Order of Failed VM Set
- p — Number of Failed Resources in VM_f
- V_f — f^{th} Failed VM in VM_f
- $o(T_f)$ — Order of Failed Tasks
- q — Number of Failed Tasks in T_f
- TAT — Turnaround Time
- r — Response Rank Value
- T_p — Prematurely Terminated Task set
- UT — Average Utilization
- MCT — Minimum Completion Time
- U/ T_u — Tasks executing on minimum underloaded VM/ least loaded VM
- € / A_{load} — Average Execution Time of Tasks Allocated over O

- t_i — i^{th} Task
- V_j — j^{th} Virtual Machine
- FT_{ij} — Finish Time of t_i on v_j
- u — Number of Unexecuted Tasks or Failed Tasks
- V_f — Set of Failed VMs
- v_f — Failed VM
- f — Number of failed VMs
- F — Flowtime
- O / T_o — Set of Tasks Allocated over the Most Loaded/Overloaded VM
- V_u — Least Loaded VM / Underloaded VM
- V_o — Most loaded / Overloaded VM
- TC — Task Cluster
- VC — VM Cluster
- f — Number of Failed VMs / $|V_f|$
- Pp — Average Progress Percentage
- V_h — Set of Healthy VMs
- O_m — Makespan Overhead
- O_{ut} — Utilization Overhead
- $M_{afterFT}$ — Makespan after Fault Tolerance
- $M_{beforeFT}$ — Makespan before Fault Tolerance
- $M_{afterLB}$ — Makespan after Load Balancing
- $UT_{afterFT}$ — Utilization after Fault Tolerance
- $UT_{beforeFT}$ — Utilization before Fault Tolerance
- $UT_{afterLB}$ — Utilization after Load Balancing

List of Abbreviations

CSP	Cloud Service Provider
IaaS	Infrastructure as a Service
SaaS	Software as a Service
PaaS	Platform as a Service

QoS	Quality of Services
SLA	Service Level Agreement
RPO	Retrieval Point Objectives
RTO	Recovery Time Objectives
VPS	VM Provisioning and Scheduling
VMs	Virtual Machines
MTTF	Mean Time to Failure
MTBF	Mean Time Between Failure
MTTR	Mean Time to Reappear
LB	Load Balancing
DLB	Dynamic Load Balancing
CC	Cloud Computing
DCCWOA	Dynamic Clustering Cuckoo Whale Optimization Algorithm
GBFD	Greedy-based Best Fit Decreasing
GWO	Grey Wolf Optimization
FTHRM	Fault-Tolerant Hybrid Resource Allocation
TAT	Turn-around-Time
WSADF	Workflow-scheduling applying -adaptable and dynamic-fragmentation
CPLCA	Checkpointed League Championship Algorithm
SDSC	San Diego Supercomputer Center
DBSA	Deadline Based Scheduling
PSO	Particle Swarm Optimization
BAR	Balance Reduce Algorithm
DCLCA	Dynamic Clustering League Championship algorithm
MSMO	Modified Sequential Minimal Optimization
CPSO	Canonical Particle Swarm

FTDS	Fault-Tolerant Dynamic Scheduling
NNCA_PSO	Nearest Neighbour Cost-Aware Particle Swarm Optimization
ACO	Ant Colony Optimization
GA	Genetic Algorithm
LCA	League Championship Algorithm
HPC	High-Performance Computing
VMM	Virtual Machine Manager
DAG	Direct Acyclic Graphs
AFTRC	Adaptive Fault Tolerance in Real-time Cloud Computing
AT	Acceptance Test
TC	Time Checker
RA	Reliability Assessor
DM	Decision Mechanism
EFTT	Efficient Fault Tolerance Technique
HBI-LB	Honeybee-Inspired-Load Balancing
PFTF	Proactive and Reactive Fault Tolerance Framework
ECB	Elastic Cloud Balancer
PLBFT	Proactive Load Balancing Fault Tolerance
DBPS	Deadline Pre-emptive Scheduling
TLBC	Throttled Load Balancing for Cloud
IVFS	Integrated Virtualized Failover strategy
CLB	Cloud Load Balancer
RR	Round Robin
ESC	Equally Spread Current
ESCEL	Execution Load
AFTRC	Adaptive Fault Tolerance in Real-Time Cloud
DBPS	Deadline Based Pre-Emptive Scheduling

VFT	Virtualization and Fault Tolerance Approach
TA & ESCE	Throttled algorithm and Equally Spread Current Execution algorithms
STLB	Starvation Threshold–based Load Balancing
GA-GEL	Genetic Algorithm and the Gravitational Emulation Local
WAMLB	Weighted Active Monitoring Load Balancing
CRUZE	Cuckoo Optimization-based Energy-Reliability aware resource scheduling technique
SIRI	Single Intervention at Random Interval
RFRTS	Reserved Fault Tolerance and Ranked Task Scheduling
FR-MOS	Multi-Objective Scheduling Algorithm with Fuzzy Resource Utilization
CWS	Cost-effective Workflow Scheduling
FCWS	Fault-tolerant Cost-effective Workflow Scheduling
HFSLM	Hybrid Fault-tolerant Scheduling and Load Balancing Model
MIPS	Million Instructions Per Second
HH	High task-High Machine Heterogeneity
HL	High task-Low Machine Heterogeneity
LH	Low task-High Machine Heterogeneity
LL	Low task-Low Machine Heterogeneity
CRFTS	Clustering and Reservation Fault-tolerant Scheduling
HEFT	Heterogeneous Earliest Finish Time
FTSA	Fault Tolerant Scheduling Algorithm
E-HEFT	Deadline Based Scheduling Algorithm
E-HEFT	Enhancement of Heterogeneous Earliest Finish Time
LB-HEFT	Load balancing- Heterogeneous Earliest Finish Time
EFT	Earliest Finish time

TABLE OF CONTENT

Chapter 1
Introduction to Cloud Efficiency: Scheduling, Fault-tolerance, and Load Balancing Techniques

This chapter sets the stage for examining several aspects of cloud efficiency, focusing on the crucial areas of scheduling, fault tolerance, load balancing, and the corresponding Challenges.

1. Introduction

Over the last 10 years, the use of Cloud has grown substantially. More facilities are incorporated into the cloud environment and are allowed to be accessed by everyone globally. Likewise, Cloud Computing companies such as IBM, Yahoo, Amazon, and Google are providing global access to services to customers [1]. Moreover, these are metered services which we commonly term subscriptions, and are frequently applied in the Software as a Service (SaaS) delivery simulation [2].

The cloud environment consists of two components i.e., the frontend, and the backend. The front end is the main interface on the consumer side and is accessed through different networks over the internet [3]. The services of the Backend side are the core component in the cloud environment. The Backend side particularly deals with the CSP (Cloud Service Provider) and provides services by utilizing data center resources. In these data centers, different physical machines known as servers are being stored. These servers are being separated into multiple virtual machines by using the concept known as virtualization to deal with and handle the upcoming requests from the application dynamically as can be seen in Figure 1.1.

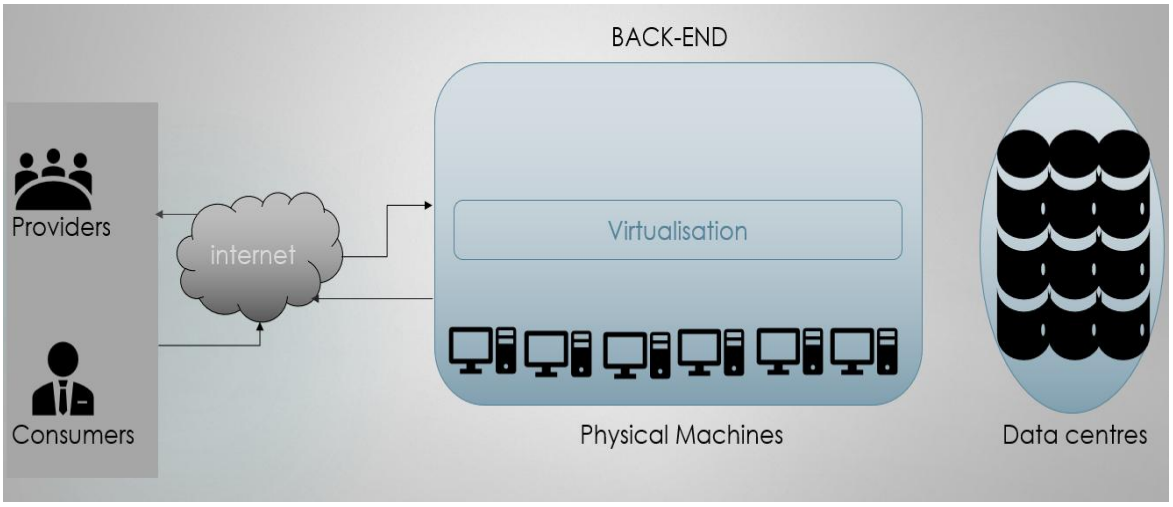


Figure 1.1: Cloud Computing Architecture

Additionally, some important components of the backend side are presented below:

Virtualization: The simulated computers that run on physical servers are possible because of virtualization. The technical term for these simulated computers is virtual machine (VM). VMs host user applications and provide virtualized environment to the users. The hypervisor is accountable for controlling the virtualization. It warrants that multiple VMs can run on an individual physical server without influencing each other.

Storage Servers: These are the storage options offered by the cloud. These include block storage for VMs, and object storage for different types of files and object data.

Computable Services: Different types of scalable and computing services are provided on a demand basis. These include GPU, memory, and other CPU services. These services are used by consumers to execute applications and other processing.

Database Services: CSPs provide various kinds of database management services. These include relational DBMS (Database Management System) like NoSQL and SQL DBMS. These services play an important role in the storage and retrieval of data.

CDN (Content Delivery Network): These are distributed networks that offer the storage and retrieval of content to the consumers. CDN improves content delivery speed and minimizes latency.

This concept allows users to purchase the services based on their needs and can be treated as metered services which we commonly term subscriptions and is frequently used in the SaaS delivery model[2]. The basic overview of Cloud Computing is shown in Figure 1.2. All the components work with one another to handle the overall process of the cloud environment. The Cloud Auditor acts as police to guarantee the top quality and integrity of services in the cloud provided by the service providers.

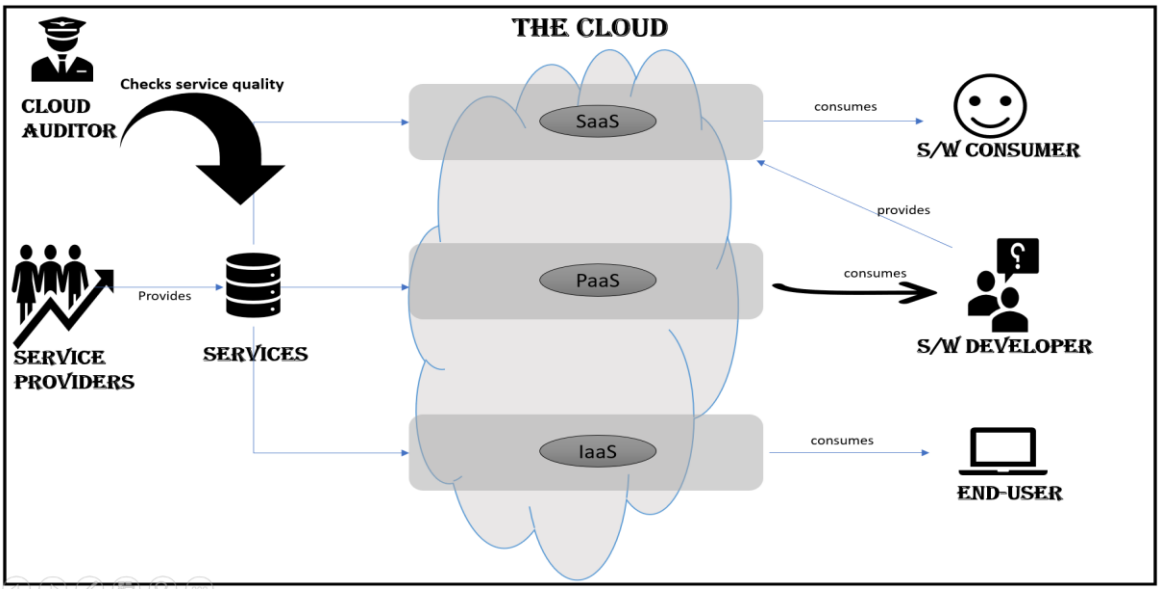


Figure 1.2: Overview of Cloud Computing

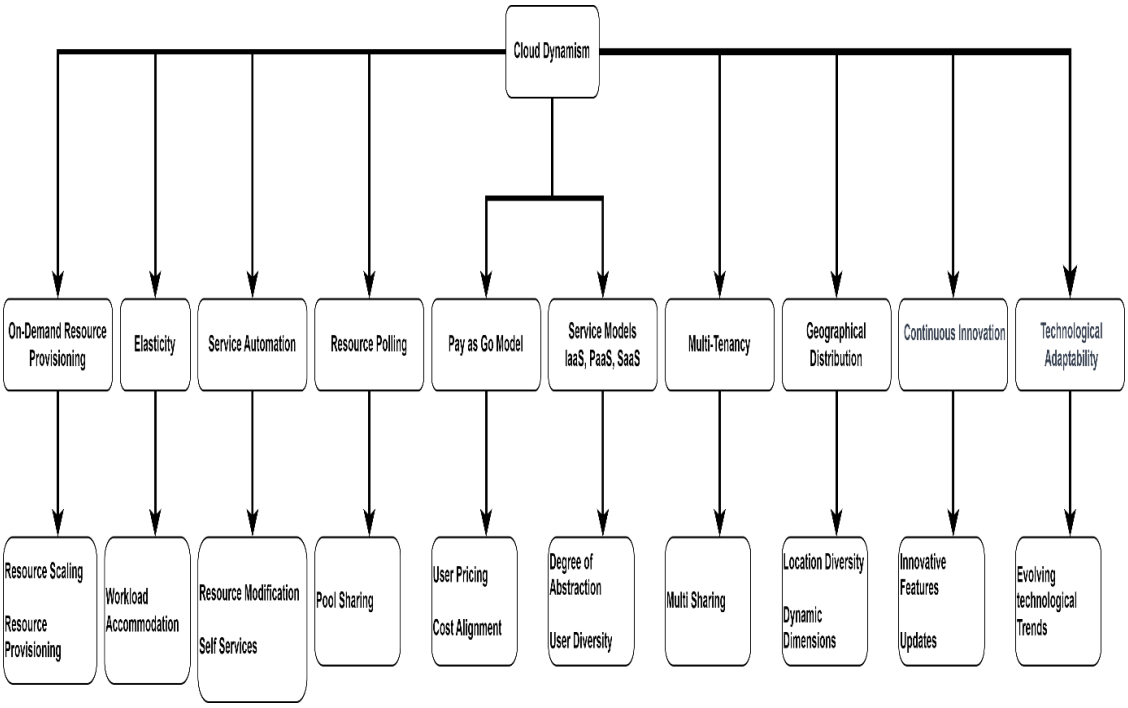


Figure 1.3: Dynamism Aspects of Cloud

In cloud architecture, there are mainly three services [4], Infrastructure (IaaS), Software (SaaS), and Platform as a Service (PaaS) [5,6].

The cloud services are well known for its flexibility, adaptability, and agility. However, at the same time, these contributions of cloud services make it a most dynamic environment. Figure 1.3 shows the aspects of cloud dynamism.

In the era of cloud computing, Service Level Agreements (SLAs) represent formal and negotiated agreements delineating the terms and conditions between customers and service providers. These agreements ensure the provision and delivery of cloud services in a manner that meets the expectations of customers, fostering satisfaction with the services provided. Some of the SLAs in cloud computing are performance metrics, availability, time, scalability, backup, security, recovery, etc. In the dynamic landscape of cloud computing, SLAs serve as a cornerstone, fostering a mutual understanding of responsibilities and expectations between providers and customers. This clarity in contractual terms enables organizations to navigate the complex realm of cloud services with confidence and strategic intent. The thesis focuses on some of the main challenges and resource provisioning techniques that ensure SLAs and other QoS (Quality of Services) parameters.

1.1. Resource Provisioning in Cloud Computing

Optimizing performance in cloud computing is crucial to ensure that cloud-based applications and services operate efficiently, providing optimal performance to users while utilizing cloud resources effectively. These optimization techniques focus on improving

response times, minimizing latency, boosting throughput, and maximizing resource utilization. Resource provisioning means opting for, installing, and operating the cloud resources at runtime to confirm guaranteed execution for applications. The CSPs take various steps for resource provisioning to follow the SLA with consumers. The hybrid provisioning of cloud resources is achieved by strategies like Scheduling, fault tolerance and, Load Balancing, etc. Besides this, the provisioning aims to enhance the QoS parameters like Makespan, Flow Time, Average Resource Utilization, Reliability, etc. The hybrid resource provisioning framework for the cloud has been presented in Figure 1.4.

Based on user needs, the component Cloud Controller maps all incoming requests to the accessible VMs. The different components cooperate properly established on multi-objective functions to optimise QoS parameters and increase the SLA between the CSP and the user.

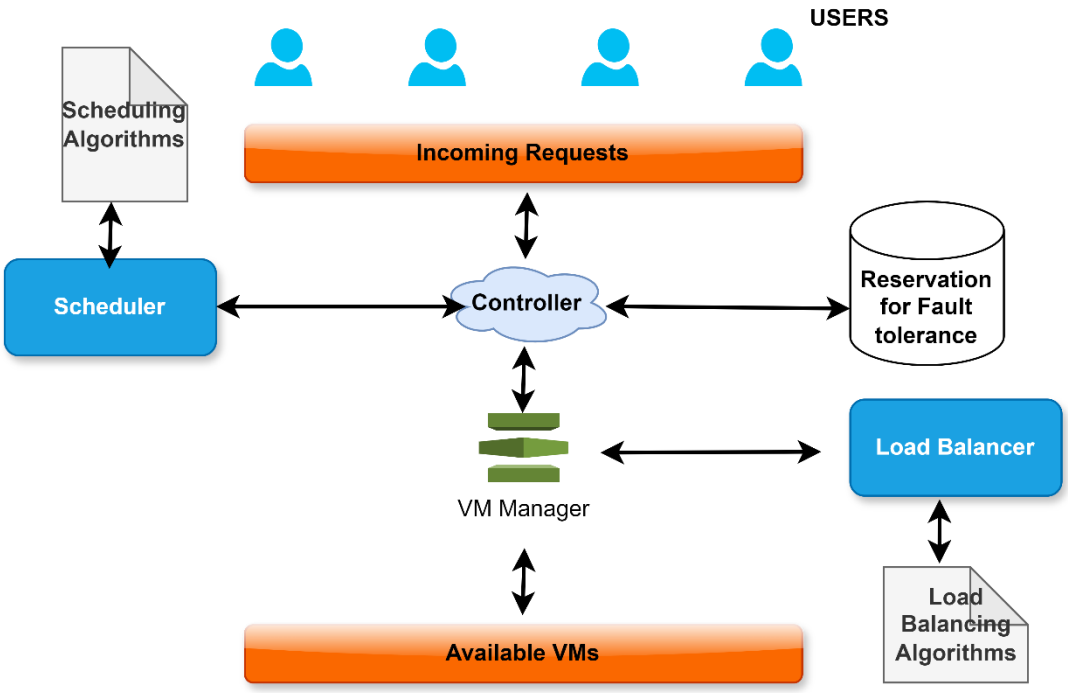


Figure 1.4: Hybrid Resource Provisioning Framework for Cloud

1.1.1. Task Scheduling

Scheduling in cloud computing entails the efficient and coordinated assignment of resources to tasks or jobs to enhance system performance. The main objectives of scheduling include ensuring fairness, optimizing resource utilization, and meeting Service Level Agreements (SLAs). Effective scheduling ensures that resources are allocated optimally, preventing underutilization or overloading of servers, which could otherwise cause performance issues. The following are key aspects of scheduling:

- *Task Queuing:* Incoming tasks are placed in a queue and scheduled for execution according to their priority, resource requirements, and several other constraints.
- *Job Prioritization:* Prioritizing jobs based on their importance or urgency is a crucial aspect of task scheduling in cloud computing and other computational environments. This process ensures that critical tasks are executed promptly, which can be vital for maintaining system performance, meeting user expectations, and adhering to service level agreements (SLAs).
- *Fair Scheduling:* Resource starvation occurs when certain tasks or users are perpetually deprived of the necessary resources to execute their operations, typically because those resources are consistently allocated to higher-priority tasks. This can lead to significant performance issues, dissatisfaction, and failure to meet Service Level Agreements (SLAs). To mitigate this, schedulers in cloud computing and other computational environments aim to distribute resources equitably among all tasks and users.
- *Distributed Scheduling:* Distributed scheduling refers to the process of coordinating and managing task execution across multiple data centers or clusters in different locations. The primary goal is to optimize resource utilization, improve performance, and ensure that tasks are executed in the most efficient manner possible, considering the distributed nature of the infrastructure.

1.1.2. Load Balancing

Load balancing is a method engaged to evenly disperse incoming network traffic or computational tasks across several resources, such as servers or virtual machines (VMs), to achieve optimal resource utilization and maintain high availability. The following are key aspects of load balancing:

- *Even Distribution:* Load balancing ensures that tasks or requests are evenly distributed among available resources. This prevents any single resource from becoming overloaded while others remain underutilized.
- *Optimal Resource Utilization:* By distributing workload efficiently, load balancing maximizes the use of available resources. This helps in achieving better performance and responsiveness from the system.
- *Application Awareness:* Some advanced load balancers can consider application-specific metrics or content when distributing traffic. This ensures that requests are directed to the most suitable server based on application requirements.

Load balancing plays a critical role in modern IT infrastructure, especially in cloud computing environments, where dynamic scaling and efficient resource utilization are essential for delivering reliable and responsive services to users.

1.2. Cloud Faults

There may be chances of faults in all these three layers in a similar way as they are possible in any type of software. Therefore, the detection and removal of faults is necessary for obtaining the best possible reliability as presented in [7], [8]. Moreover, the deficiencies in the infrastructure of the cloud yield a direct impact on resource reliability and availability [4]. These deficiencies need to be critically analyzed and treated to boost reliability and robustness. DNN, a powerful deep learning tool exhibits is a promising solution for this [9]. Fault Tolerance is a significant technique that can notice, locate, and recover from faults and failures in the cloud environment. It makes the cloud more robust and enhances the efficiency of the environment [10]. Mainly, fault tolerance falls into two sub-areas i.e., Hardware Fault Tolerance and Software Fault Tolerance [11].

On the other hand, scheduling tasks appropriately is vital in delivering critical and essential services of the cloud. The ineffective scheduling of tasks increases the task execution time and waiting time. Besides, insignificant load balancing results in the under and over-utilizing of resources where the under-utilization of resources can lead to the wastage of resources, and over-utilization of resources can degrade the performance of cloud systems. Hence, proficient load distribution is essential to boost the performance of cloud-based applications.

There is a fundamental need to incorporate load balancing and scheduling in efficient fault-handling mechanisms due to architectural challenges in the cloud system. Therefore, this paper conducts a hybrid review employing fault tolerance with scheduling, load balancing, and analysis of QoS parameters optimization. This comprehensive review primarily centers on three core classifications of fault tolerance techniques, namely Reactive, Proactive, and Resilient Approaches. The Reactive Procedures are the conventional techniques of fault tolerance that include replication, detection, checkpointing/restarting, and recovery. In the Proactive Methods, the system is prevented from reaching a defective state that includes monitoring, prediction, and pre-emption. The actions are taken to minimize the defects, and thereby the failure condition is avoided. The Resilient Methods have shown a recent take-off in the literature and indicate a potential trajectory for the future of fault tolerance in cloud environments. This is because these methods are grounded on artificial cleverness and ML [10]. Besides, simulation toolkits play an analytical role in evaluating settings of cloud computing. These toolkits allow us to simulate and evaluate the cloud set-ups cost-efficiently without the requirement for massive infrastructure. Some of the most effective and powerful simulators have been discussed in [12]. Comparative analysis has been

performed in among various simulators concerning various parameters to determine the features and functions of each toolkit [13].

1.2.1. Fault Tolerance in Cloud Computing

Faults in any resource may affect the task execution time and QoS parameters of the cloud, which will eventually reduce the deed of the system. The efficient fault tolerance policy helps to identify and overcome errors in the cloud architecture, and thereby the performance metrics are boosted. The fault tolerance capability should be considered with other techniques like scheduling and load balancing for the effective performance of the system. Moreover, the load balancing and scheduling approaches should do their respective standardizes along with fault tolerance. In case of a crash or connection error, the system should be capable enough to provide an alternative VM to handle these failures for smooth and uninterrupted task execution. Because these crashes in any nodes will affect the efficiency of the entire system. Therefore, handling faults enhances the utility of the system to accomplish the tasks precisely and accurately resolving the occurrence of internal defects [14]. An inclusion of fault tolerance with other reliability-related techniques like scheduling and load balancing will make the cloud environment more efficient, specifically for the real-time and dynamic processing of tasks [15]. Hence, fault tolerance is a major aspect that ensures robustness, reliability, and other performance metrics in the cloud environment [16], [17].

1.2.2. Fault, Error, and Failure Taxonomies

The fault is the condition of the system when it loses the ability to function for an expected output due to an unexpected condition or defect in any of the internal or external components. The main faults within the cloud environment are enumerated as follows:[18]

The Network Faults: These defects arise due to network interruption in any connection, nodes, cluster, etc., [19], [20].

The Physical Faults: When any of the hardware resources like CPU, memory, storage, etc., fails, these types of faults will occur. The power failure also gives rise to these types of faults [18].

The Process Faults: These are the common faults in a cloud environment that occur because of the unavailability of any resource, software, etc., [19].

The Service Expiry Fault: This type of fault arises if the service clock of the resource runs out when the application is in use [19].

The Media Fault: Any crash in the media of the cloud will lead to these types of faults [15].

The Processor Faults: This type of fault mainly occurs because of malfunctioning in the operating system [21] .

The Restrictions Faults: This type of fault occurs when any fault arises and is unnoticed or ignored by the controlling or any other responsible agent [22].

The Parametric Faults: If the optimizing parameters are ambiguous or do not differ and remain unexplained, this type of fault occurs [22].

The Time Restriction Faults: These faults occur when the particular application is not completed by the predefined deadline [22].

The fault tolerance mechanism makes the cloud environment efficient by providing necessary services even in case of failure of one or multiple components. If there is any kind of fault in the system, it leads to error, and error, in turn, culminates in failure.

Fault: The abnormal state of any coordination when assigned tasks cannot be performed. Usually, the fundamental cause of this state is the presence of some bugs in single/multiple components of the system [23], [24]. Faults are categorized into various groups, as depicted in Figure 1.5.

Error: A system experiencing faults may transition into an error state. Compromised performance due to errors can subsequently result in incomplete or complete failure of the system. Errors have been classified into the following categories, as shown in Figure 1.6.

Failure: The presence of an error can takes the system to the failure state, and it has an absolute effect on the user. Moreover, the failure is recognized by the user by seeing the incorrect output of the system [23], [25], [26]. The failures have been classified into the following categories, as exhibited in Figure 1.7.

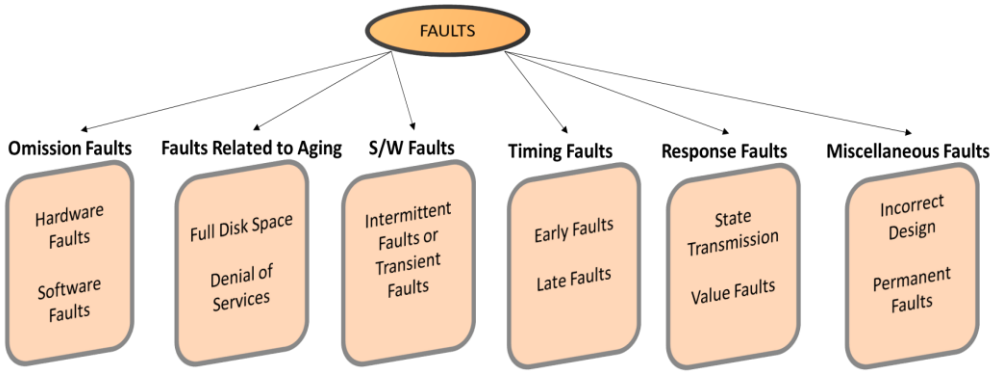


Figure 1.5. Showing Different Fault Categories

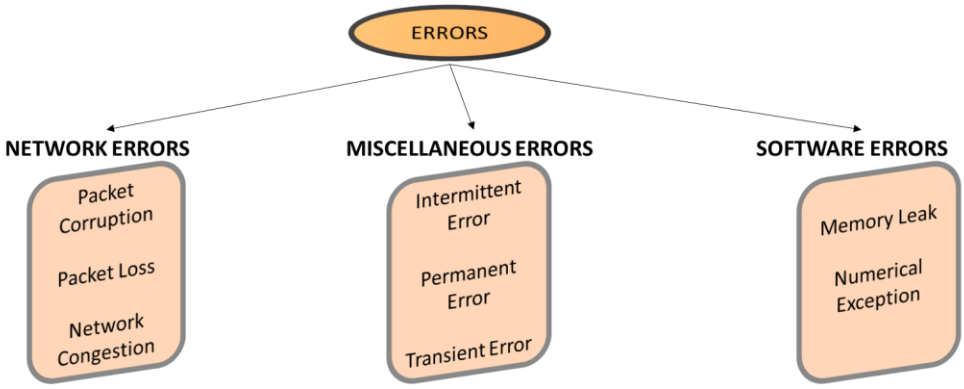


Figure 1.6. Showing Different Error Categories

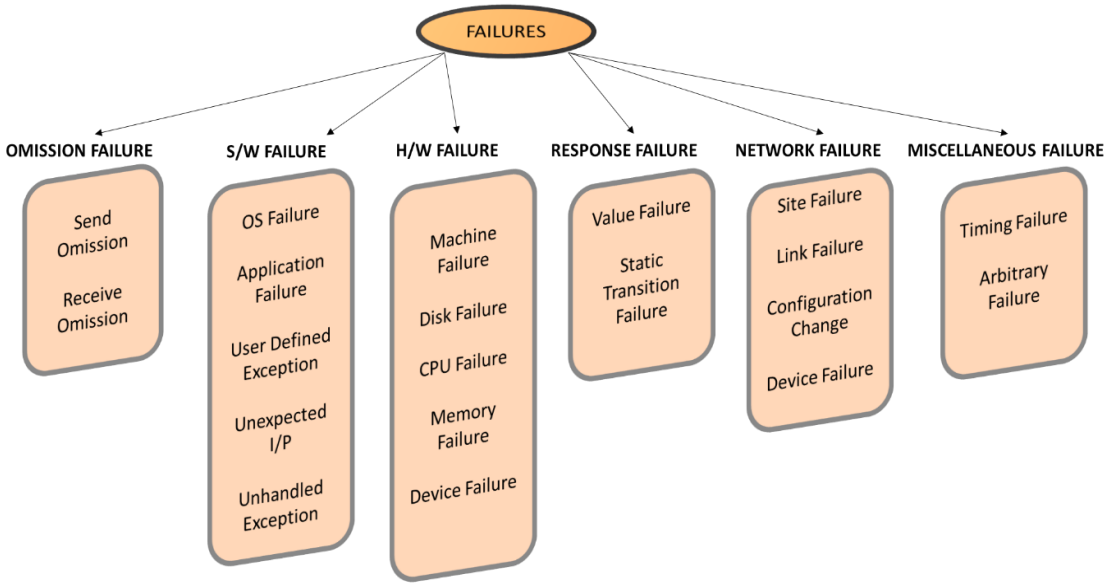


Figure 1.7. Showing Different Failure Categories

1.2.3. General Fault-tolerance Challenges in Cloud Computing

Ensuring a fault-tolerant cloud environment involves evaluating numerous challenges. Some of these challenges are discussed below:

Task and failure heterogeneity: The cloud utilizes different hardware and operating systems simultaneously and considers the underlying heterogeneous frameworks [27]. Resultantly, in handling the heterogenous type of faults, and eventually increasing the complexity to overcome them.

Automation: The extensive use of VMs in the cloud environment is increasing exponentially and managing these platforms in real time is more difficult. Therefore, there is a good need to automate fault tolerance strategies for complex networks [28].

Cloud halts: The main plan of fault tolerance is to provide uninterrupted service altogether in case of any service interruption or malfunction of any host server or network system. The Service Level Agreements [29] for all companies should be prepared accordingly.

Retrieval Points and Recovery Time Objectives targeting: This Point is established to preserve the set of track records that may be at risk of loss in the event of a server error [29]. On the other hand, Recovery Time is the time required by the procedure to get back on track or running after the failure [30]. The main aim is to decrease RPO (Retrieval Point Objectives) and RTO (Recovery Time Objectives) at the minimum possible rate [10].

Cloud Workload: Cloud workloads are the specific applications-related tasks/services or specific amounts of work executed on a cloud resource. The workloads could be of two types, i.e., Enabled, and Native loads. The Native workloads are also labeled as "born on the web" and are entirely cloud-developed applications. On the other hand, an enabled workload pertains to the computational tasks generated by cloud applications. Moreover, the Proactive and resilient approaches seem relevant [31] to fill the fault tolerance conditions of both Active and Native concepts [10].

1.3. Measures for Effective Cloud Reliability- A need for the hybrid framework

The claim for the cloud computing standard has enlarged intensely in the past few years as it allows the dynamic fetching and renouncing of computing resources that too in a device-independent and cost-effective manner with slight effort or communication from the service provider. Despite lots of enhancements in the cloud, it is still prone to many system failures which results in growing apprehension regarding the reliability of cloud public services. Reliability is the way of measuring the efficiency of the system and its value can be adjusted accordingly after performing computation where the default reliability is 100% [32]. The conditions of reliability must be met for stable and efficient processing of the cloud. It is also one of the critical Quality of Service constraints. Moreover, optimized QoS parameters play an important role in effective and adequate resource allocation and have been extensively inspected in Cloud Computing standards. These parameters are used to consider the efficiency of various Scheduling, Load Balancing, or Fault Tolerance techniques in the cloud.

The hybrid framework provides several advantages in comparison to single schemed framework discussed below:

- *High Availability and Reliability*

Reliability is the critical parameter that supports user trust by maintaining SLAs. The mechanism of fault tolerance enables the system to continuously operate even in the presence of faults and failures. Scheduling ensures the optimal allocation of computational resources to the tasks. This prevents avoiding pauses and reduces the threats of failures.

Load balancing prevents the single resource from becoming the target by uniformly distributing the load and thereby enhancing system reliability.

- *Optimizing Resource Utilization*

Unnecessary resource allocation for fault tolerance can lead to resource underutilization. A hybrid model improves the use of these redundant resources by incorporating them into scheduling and load-balancing strategies. Additionally, the overburden of resources is prevented by load balancing to boost the utilization of resources while dynamically adjusting the task distribution.

- *Enhancing System Performance*

Efficient scheduling with fault tolerance maintains the levels of performance and ensures the timely completion of tasks. Moreover, load balancing prevents the overloading of a single VM thereby escaping performance disgrace.

- *SLA Requirements*

Efficient scheduling with continuous operations even in the presence of failures is crucial in maintaining SLAs and ensures time constraints, reliability, and other deadlines.

- *Advancing Scalability*

The scaling factor of the cloud environment necessitates the need for robust fault-tolerant scheduling to ensure reliability throughout the system. Moreover, the growing number of cloud users simultaneously increases cloud tasks. This requires the load balancing system to ensure the corresponding scaling without performance degradation.

- *Efficient Cost*

Lowering the requirement for excessive redundancy via a hybrid model can lead to minimum cost while ensuring reliability.

- *Fault associated Overheads*

The faults often lead to overheads even when handled. The load balancing integrated with fault tolerance will reduce the overheads.

In conclusion, integrating these three factors provides a complete and comprehensive approach to address the varied challenges faced in cloud environments, leading to extra robust, efficient, and reliable cloud-essential services.

Below is presented an explanation that includes a real-world example illustrating the necessity of hybrid models:

Illustrative Example

Consider the scenario, where the CSP hosts several services and applications for its clients, utilizing solely fault tolerance mechanisms (single-model schemes). In often cases, fault tolerance frequently results in redistributing the workload from faulty VMs to the unaffected VMs. This redistribution often upsets the load equilibrium between VMs, which leads to an unequal workload distribution and a deterioration in overall service performance. However, if CSP implements the hybrid model which integrates multiple reliability measures would enhance reliability and provide robust services to the clients. In our example, if CSPs employ the hybrid model that performs load balancing after fault tolerant measures. This will help CSPs to simultaneously minimize the risks of non-uniform load distributions and other overheads associated with fault tolerance and progress the QoS.

Besides, to make this emerging domain more observable for future researchers, there is a need to analyze the up-to-date methods concerning these factors [10], [29]. This review is also inspired by peer surveys of the existing literature along with their limitations. Moreover, it represented the analysis of some important aspects of the existing literature such as QoS, static/dynamic, environmental setup used, fault tolerance approaches, and fault models, and presented the results in the graphical visualization form. The analysis provided offers a comprehensive perspective on the existing research efforts that have been the focal point of existing studies. The overall comparison of the top-cited surveys with the proposed survey is also illustrated in the subsequent sections.

1.3.1. Cloud Scheduling Approach

Cloud scheduling is performed by mapping the incoming task to the most suitable available VM. The objective of ascertaining the sequence in which events or tasks should be executed in the cloud and simultaneously analyzing the required QoS parameters is termed Scheduling. Cloud Scheduling mainly includes the following:

Prediction of future incoming workloads and Normalizing the QoS parameters.

Selection of the most optimal VM and executing the particular task via, Heuristic/Meta-Heuristic algorithms.

Generally, the VM/task scheduling is done in two ways:

On-Demand Scheduling: This scheduling considers the dynamic cloud workloads on demand and VMs are provided quickly by cloud service providers as required. However, it may lead to the problem of workload dispersal. In other words, multiple tasks may be processed by a single VM at a time (Over-provisioning Problem) resulting in degrading the performance of the system.

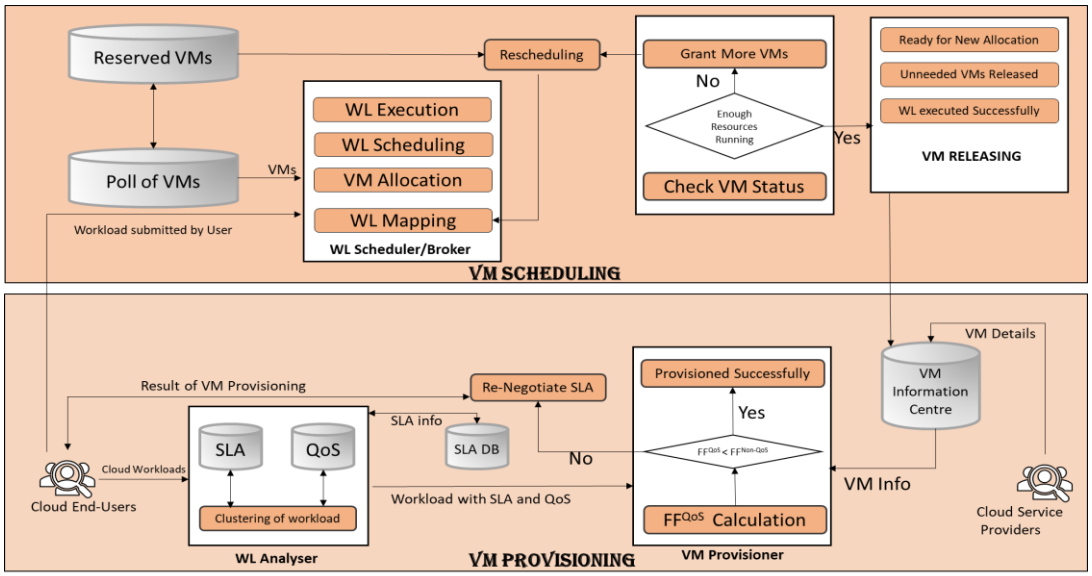


Figure 1.8. VM Provisioning and Scheduling (VPS)

Long-Term Reservation: This scheduling reserves the resources for the long term. However, providing many VMs can lead to Under-provisioning problems in some situations.

These Under and Provisioning problems may cause the wastage of VMs and task execution time, and thereby the overall cost of services may increase. Hence, a well-organized and effective provisioning technique is essential that examines and schedules the cloud workloads efficiently. Figure 1.8 explains the process of VM Provisioning and Scheduling (VPS) [33].

The main aims of VM provisioning are:

- *Fulfill the User's demand without SLA violation.*
- *Prior prediction of user requirements based on incoming workload size.*

In cloud provisioning, the SLA is settled between the end users and Cloud Service Providers after fully analyzing the incoming workloads. Before scheduling (mapping) the incoming workload (applications/tasks) to the particular VM/resources, the running VMs are monitored regularly for load estimation [34]. If the VM is found overutilized, then that particular VM is disabled temporarily for any future assignments and these VMs are not allocated immediately after mapping. Afterward, the task executing capability of the VM is also tested before any further allocation. This study also contains a review of various research papers focusing on the principles of load-balancing and scheduling. In the cloud, efficient scheduling of jobs is the main factor ensuring high-performance applications. However, in the cloud, scheduling not only has to pact with the dynamism and the widespread nature of the cloud, but it should also consider the optimization of other important parameters. The matching of tasks to the corresponding machines and scheduling the organization of execution of these tasks refers to mapping. Efficient mapping minimizes

the total execution time of the meta-task. The meta-task is identified as a collected work of independent tasks having no inter-task dependencies. The mapping of such meta-tasks is being achieved statically (i.e., offline or in an analytical manner). The general problem of optimally mapping tasks to machines is NP-complete [35]. Task scheduling is the fundamental step of VM management in the cloud. Task scheduling can be of two types: Static and Dynamic Scheduling [36].

1.3.2. Fault-Tolerant Approaches

Cloud is a dynamic system that supports several dispersed resources i.e., VMs that are heterogeneous and complete millions of user tasks. Nevertheless, this VM has the flexibility to join or exit the system at any given time. Thus, achieving fault tolerance is a critical issue in such dynamic systems. Additionally, the execution of a fault-tolerant system also leads to the optimizations of various QoS parameters and cloud characteristics. Therefore, significant benefits can be attained. It also assures task execution on time, in case of any unexpected scenarios like failure, resource disconnection from the system, task migration, any other unanticipated user operation, etc. Moreover, while numerous previous studies have tackled fault tolerance and task allocation, only a limited number have examined issues at the processor level. In recent literature, a handful of works have delved into extensive research on scheduling and load balancing while incorporating fault tolerance [22]. The concept of abstraction has been split into different layers, i.e., Infrastructure as a Service, Platform as a Service, and Software as a Service layer. There is a necessity to implement appropriate fault tolerance techniques for fault diagnosis to determine several faults in these service levels. This chapter includes various fault diagnosis methods corresponding to these service layers, along with fault categories. The defects in any layer can have an impact on its top layer because of the layer interrelationships [22] as shown in Figure 1.9.

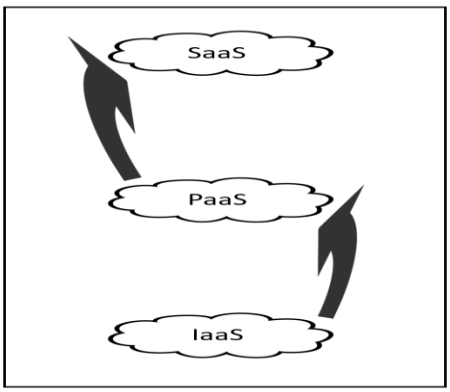


Figure 1.9: Layered Architecture Relation of Cloud Fault Tolerance

There are chances of faults in all these three layers. To identify and recover from these faults some software-level algorithms are applied. The deficiencies in the infrastructure of the

cloud yield a direct impact on resource reliability [4]. These deficiencies in the cloud need to be critically analyzed and treated to boost the reliability and robustness of the cloud environment. Fault Tolerance is a significant technique that loads the cloud environment with some important advantages like noticing, locating, and recovering from faults and failures. This makes the cloud environment more robust and enhances the efficiency of outcomes of the cloud environment [10]. Mainly, fault tolerance falls into two sub-areas i.e., Hardware Fault Tolerance and Software Fault Tolerance [29].

This chapter mainly focuses on the methodologies of fault tolerance methods which are reviewed based on three main strategies: Reactive Methods, Proactive Methods, and Resilient Methods. The Reactive Methods are the conventional techniques of fault tolerance that include replication, detection, checkpointing/restarting, and recovery. In the Proactive Methods, the system is prevented from reaching the defective state. The actions are taken to minimize the defects, and thereby the failure condition is avoided. These methods include approaches like monitoring, prediction, and pre-emption. The Resilient Methods have shown a recent take-off in the literature and are most probably the future of fault tolerance in the cloud environment. These methods are based on artificial intelligence or machine learning [10].

Moreover, to reach higher levels of strength in cloud computing, the failures need to be accessed and handled effectively [25], [37]. Extensive work has been proposed in the literature to make the cloud fault-proof. Some approaches proposed in the literature can be labeled as mentioned in Figure 1.10.

1.3.2.1. Reactive fault tolerance:

Once a defect has occurred, reactive fault tolerance is applied. Using this approach, we can decrease the impact of the fault in the cloud and thereby increase the system's robustness and reliability [30], [38]. The focus is on the device recovering in case of failure inside the system [29]. Furthermore, data replication and data transfer are used for restoration [39]. These approaches address Byzantine Faults, Crash faults, Hardware faults, and Host failure. Different fault-tolerant techniques that utilize a reactive approach are planned in Table 1.1.

1.3.2.2. Proactive fault-tolerance:

Predictionary provides pre-planned alternative solutions for the process of handling faults; therefore, fault prediction is proactive. Moreover, the faulty component is substituted with an alternative component runtime to avoid recovery from errors and faults [38], [40] [4], [41]. This approach provides the effectiveness of cost with maximum efficiency and reliability of the system [42] and addresses Software and Parametric faults. Some of the proposed proactive fault-tolerant techniques in the literature are listed in Table 1.2.

1.3.2.3. Resilient Fault-tolerance:

These techniques have some similarities with the Proactive approach. The defects are forecasted, and the effects are prevented or moderated by applying some methodologies. The forecasting utilizes some intelligent learning, which makes Resilient techniques different from Proactive ones. These approaches are adopted for general faults. In this strategy, the system is continuously monitored for faults, which makes it adaptive fault tolerance [10]. Some of the proposed Resilient fault-tolerant techniques in the literature are presented in Table 1.3.

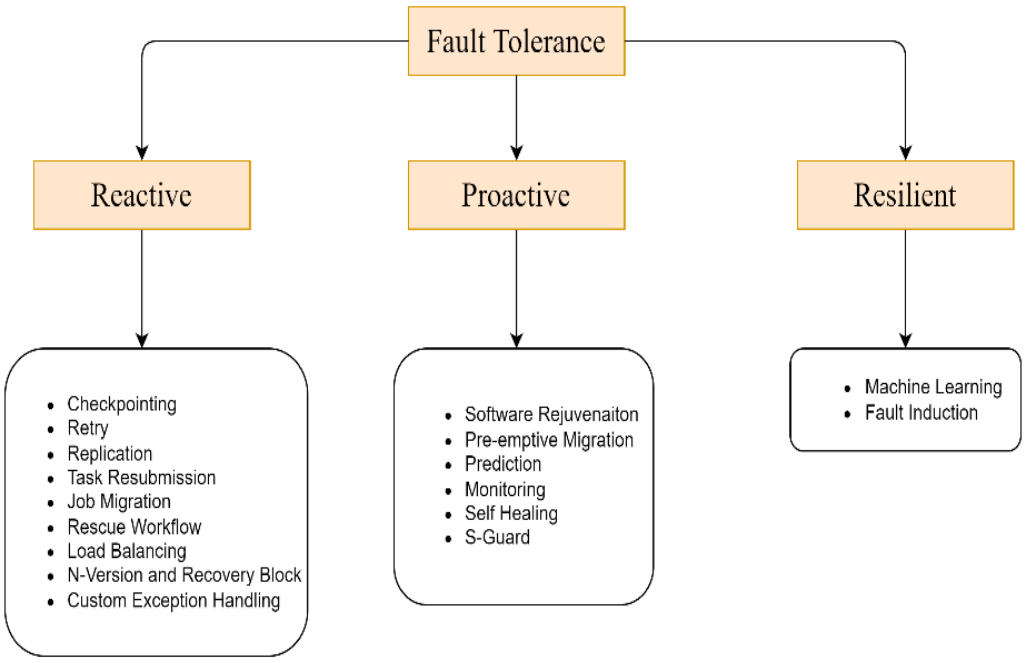


Figure 1.10. Showing the Categories of Fault Tolerance Techniques under Different Approaches

Table 1.1: Enlightenment of reactive fault-tolerant techniques

Strategy of Fault Tolerance	Classification/ Category	Enlightenment	Problems/ Issues Sheltered
Reactive Strategy	Check-pointing [22], [43]	The system state is saved periodically and in case of failure, the job is restarted from the last checkpoint rather than beginning. i.e., the job is restarted from the recent state.	Byzantine Faults, Crash faults, Hardware faults Host failure
	Retry [25]	In case of a fault in the task, we repeat the task with the same resource until it succeeds without consideration of the reason for the error.	
	Replication [25], [42]	In this approach, replicas of tasks are created and stored at diverse places. Until all these replicas are destroyed, the execution of the task will continue even in the presence of malfunctions and failures.	
	Task Resubmission [25], [43]	This approach submits and resubmits the failed task to the identical or alternative resource [10]. There is a resource loss in this technique by re-executing the unsuccessful task repeatedly [44].	
	Job Migration [26]	The failed job is migrated from the particular machine to an additional machine.	
	Rescue Workflow [26]	This approach lets the system continue working even in the presence of fault until the fault will not allow the system to progress further.	
	Load Balancing [38],[45] [23]	The total load is distributed among machines efficiently so that no machine will be under or overloaded [10],[46]. Load balancing helps to condense the hardware and time costs, hardware costs [48] & thereby improving system execution and efficiency [37], [38].	
	N-Version and Recovery Block [49]	These are the most commonly used methods of fault tolerance in the software atmosphere where N-version programming has N independent groups/developers for developing N different versions of software modules [50]. All these different individuals will try to cover all likelihoods of fault. Recovery blocks are used in case of conducting the duplication of any job and are the boundless technique to diminish the drain of any undesired incident [51].	
	Custom Exception Handling [29]	In this approach, the developers purposively insert some code or script into the software to handle certain errors at running time [10].	

Table 1.2: Enlightenment of Proactive Fault-Tolerant Techniques

Strategy of Fault Tolerance	Classification/ Category	Enlightenment	Problems/ Issues Sheltered
	Software Rejuvenation [25], [26]	In this strategy, the system is rebooted periodically, and every time the system starts from the new state. Mainly this strategy is used to address the issue of aged devices [48]	

Proactive Strategy	Pre-emptive Migration [41], [52]	This strategy involves the ongoing and constant observation of an application to track crucial resources like CPU and RAM [53]	Software and Parametric faults
	Prediction [54]	This approach requires a basic knowledge of system defects [55]	
	Monitoring [44]	This strategy more actively participates in carrying innovative resources such as planning, expanding, and migration [56]	
	Self-Healing [57], [58], [59]	This strategy mainly uses the divide-and-conquer technique to improve the performance of the system. It allows the system to classify, recognize, and heal the problems itself without the intervention of any administrator.	
	SGuard [49]	The SGuard strategy primarily depends on the recovery and rollback process and is mainly proposed for sharing the video services [17]	

Table 1.3: Enlightenment of Resilient Fault-Tolerant Techniques

Strategy of Fault Tolerance	**Classification/ Category**	**Enlightenment**	**Problems/ Issues Sheltered**
Resilient Strategy	Machine Learning [10]	Machine learning techniques mainly reinforcement learning [38] are involved in analyzing the features and characteristics of machines. Such strategies help the system to manage its faults according to its surroundings.	Adapted to General Faults
	Fault Induction [10]	This strategy is a recent strategy used in cloud environment [38] Failures are managed by making assumptions based on the reaction of the system.	

The reactive strategy does not require to enforcement of any qualification mechanism in the system until and unless the fault occurs. In such a strategy, efforts are being made to moderate the injurious effects only after the detection of faults in the machine. Efforts are being made to moderate injurious effects in the machine after the faults have happened [48]. In a Proactive strategy, the system is in continuous tracking to analyze the faults and eliminate them before they appear. The device state is continuously screened to guess the coming faults in advance so that corresponding steps will be taken to eliminate these upcoming faults. In Resilient strategies, the system operates even in the presence of faults, and the faults are removed in the given timeframe.

The corresponding pros and cons of these strategies are presented in Table 1.4 and Table 1.5 respectively.

Table 1.4: Pros of Fault-Tolerant Strategies

Reactive Strategy	**Proactive Strategy**	**Resilient Strategy**
Can handle rare faults [10] Methods like checkpointing, and restarting work well for a lengthy application [10]	Restoration from faults restricts the susceptibility of the system. The forecasting makes the system more effective [10] This strategy is more appropriate for real-time applications [10]	These strategies seem the future of Fault tolerance. The faults are discovered and eliminated continuously. This reduces the resource requirement as the system handles faults efficiently [10]

Table 1.5: Cons of Fault-Tolerant Strategies

Reactive Strategy	**Proactive Strategy**	**Resilient Strategy**
These strategies cannot be applied to real-time applications. Restoration from failure will increase the response time significantly [10].	As the prediction is required here, and wrong predictions will degrade the performance of the strategy [10]	Frequent modification is required as the cloud itself is the most dynamic environment. Learning time is required for the agent [10].

Reactive techniques have some similarities to the proactive approach. Moreover, the defects are forecasted, and the effects are prevented/moderated by applying some methodologies. The forecasting uses some intelligent learning that makes resilient techniques different from proactive ones. Compared to conventional fault tolerance techniques, resilient fault tolerance provides increased durability and adaptability in the event of system breakdowns. Some of the advantages of resilient fault tolerance over traditional fault tolerance are:

1. Dynamic environment

Resilient systems can bounce back from errors without sacrificing functionality because they can dynamically adjust according to shifting circumstances. They are made to respond quickly to changing threats and difficulties. However, conventional fault tolerance techniques could find it difficult to adjust to sudden or rapid shifts in the environment. They might not react to new kinds of errors as well since they frequently rely on predetermined rules.

2. Recovery

Often, automated recovery mechanisms found in resilient systems are capable of promptly detecting and fixing errors without the need for human interaction. This reduces the effect on coordinated functions and decreases downtime. On the other hand, to recover from

errors, traditional approaches might need more manual intervention as compared to Resilient ones. This could result in longer time frames for recovery and a higher chance of service interruption.

3. Real-time track reporting

Sophisticated analytics and tracking techniques that offer practical observations into the health of the system are frequently integrated into resilient systems. Further, active defect identification and prevention are made possible by these techniques. Unlikely, conventional approaches might be less successful in locating and addressing errors as they depend on frequent checks or event-generated reactions.

4. Optimization

Resilient systems are made to maximize the use of the resources at hand during fault recovery, guaranteeing that resources are distributed effectively to sustain critical operations. Besides, traditional techniques could use expensive strategies, which could result in more inefficiency and lower effectiveness of the system all around.

5. Flexibility and adaptability

Improved adaptability and flexibility are frequently displayed by resilient designs, enabling them to adjust to changing demands and adjust resources upward or downward in response to consumption.

However, traditional approaches could find it difficult to adjust dynamically or regulate shifting demands, which could result in inefficiencies during times of high consumption.

1.3.2.4. General Problem Formulation for Fault Tolerance Using Replication

Problem Statement: Problem formulation that focuses on the importance of fault tolerance in the circumstances of clouds.

Problem Scope: The fault tolerance in the cloud is addressed for continuous service delivery even in the event of failures or breakdowns.

Objectives: The main goal is to reduce fault-related service interruptions and downtime to maximize cloud service availability. Additionally, increasing resource utilization, loss of data, and maintaining SLA thresholds are also included in the formulation.

Problem Constraints: To guarantee that the efficiency effect of services is provided as needed. The fault tolerance techniques should add as little overhead as possible. Moreover, the solution should apply to the related computational resources.

Parameters: The parameters manipulated during fault tolerance are MTTF (Mean Time to Failure), MTBF (Mean Time Between Failure), MTTR (Mean Time To Reappear), etc. However, the parameters that are optimized are average resource utilization, makespan, recovery rate, failure rate, success rate, etc. There can be some decision parameters in fault

tolerance such as selection of alternative resources, fault detection algorithm, recovery mechanism, etc.

Problem Formulation: For fault tolerance in real-time systems, two important sets can be considered i.e., tasks set (T), and VM set (V). T: $\{t_1, t_2 \ldots t_n\}$, indicating that n real-time tasks at any instance in the Cloud environment. For each actual-time task $\{t_i \mid t_i \in T\}$, t_I has some set of attributes associated with it such as arrival time, dimensions, expected execution time, anticipated finish time, anticipated harvest time, deadline limit, etc. Deadline and harvest time can be related to each other as follows:

Exp HT = D – Min PT

V: $\{ v_1, v_2 \ldots v_m\}$, indicating that m number of accessible VMs in the Cloud environment. For each accessible VM $\{v_i \mid v_i \in V\}$, v_I has some set of attributes associated with it such as vm_id, capacity, cluster, etc.

Fault tolerance can be achieved by using any of the fault-tolerant approaches. Here we are utilizing the replication Fault tolerant technique. Here, the scheduler should possess the capability to generate the required amount of replicas separately for every real-time task.

For each $\{t_i \mid t_i \in T\}$

Enable the scheduler to generate replicas

Allocated VM to each replica,

Calculate the expected finish time $f_{i,j,k}$ for a given replica by the following equation:

$$F_{i,j,k} = A(t_i) + w(r_i) + e(r_{i,j,k})$$

Where, i, j, and k represent the key of the original real-time task, the key of the current replica, and the key of the allotted VM, respectively. A is the arrival time for the real-time task, w is the waiting time of the replica, and e is the expected execution time of the replica over the allotted VM.

Further, $e(r_{i,j,k})$ is computed by the following equation:

$$e(r_{i,j,k}) = \frac{task\ dimensions}{computional\ power\ of\ alloted\ VM}$$

After $e(r_{i,j,k})$ expires, the following condition is evaluated for every real-time task.

If $\forall$ *replica*(t_i) *= failed*

Mark t_i *"failed"*

Else Mark t_i *"Succeeded"*

Additionally, a reservation mechanism can also be used to achieve Fault tolerance where we reserve the VM in advance which will be allocated in case of fault.

Estimation Metrics: It comprises the estimation of some optimization parameters like recovery time, reached reliability, and effectiveness of resource use for both fault and regular operating conditions.

1.3.3. Load Balancing in Cloud

Load balancing is among the chief requirements of a cloud environment. Load balancing usually shifts the load from the highly loaded VM to the minimum loaded VM to ensure the uniform dispersal of load among VMs. It aimed to share the workload among computational resources to maintain load equilibrium and allow each resource to function within its designated efficiency threshold. The uneven distribution of load among VMs affects the improvement of response time, interaction overhead, output, and resource utilization of the system [46]. Furthermore, it improves VM availability and maintains reliability. Besides, the load can be balanced by implanting resource redundancy that fulfills scalability. Figure 1.11 demonstrates the illustration of load balancing in the Cloud system. The load balancer is responsible for adequately balancing the workload of several users among distinct VMs located at diverse locations such as the U.S., the U.K., India, etc. Numerous strategies have been proposed by researchers to attain the finest load balancing.

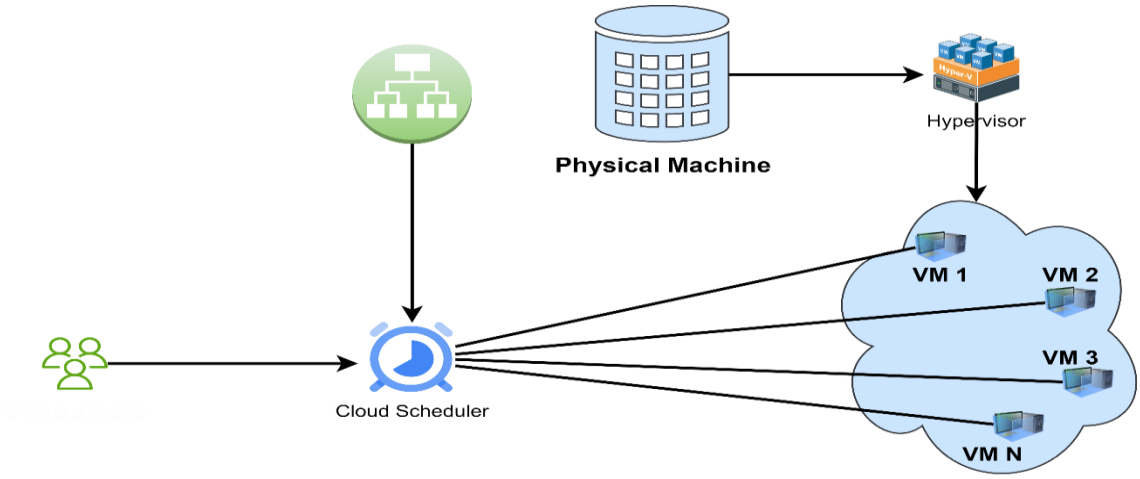


Figure 1.11: Load Balancing in Cloud

1.3.3.1. Types of Load Balancing

In the Cloud, VM can work independently or collectively as per the requirement and nature of the task. Each VM is capable of processing workload as per its processing capabilities. The main objective of load balancing is to attain adequate workload distribution among available VMs. In general, any load balancing algorithms comprise two elementary policies, i.e., the transfer policy and the location policy [43]. The transfer policy adjusts whether the VM is overloaded or not. It further deals with the dynamic aspects of a system. The transfer policy also elects if there is a necessity to initiate the load migration for the system. By

means of workload evidence, this policy determines when a node becomes suitable to act as a sender i.e., transfers a task to another VM. It further determines when a node acts as a receiver and retrieves a task from another VM. However, the location policy decides on a suitably under-loaded VM. It locates complementary VM to/from which a VM can send/receive workload to improve the overall performance of a system. Location-based policies are further categorized as receiver-initiated, sender-initiated, or symmetrically initiated. The location policy chooses an alternative VM for a job transmission transaction. If the VM is identified as an eligible receiver, the location policy looks for an eligible sender VM. If the VM is recognized as an eligible sender, the location policy seeks out a receiver VM to receive the jobs. Once a VM becomes an eligible sender or receiver, a selection policy will apply to prefer which of the queued jobs is to be moved now [46]. Based on the information and implementation used by these two policies, load balancing mechanisms are classified as mentioned below [60]:

- *Static Load Balancing*

In static load balancing, the task to be assigned to a VM is of fixed size. The system itself has very minimum load variations.

- *Dynamic Load Balancing*

Dynamic load balancing (DLB) distributes load competently so that the overall workload of the dynamic system is balanced efficiently, and maximum VM exploitation will be achieved in case of joining and leaving of VM in the system.

- *Adaptive Load Balancing*

Adaptive algorithms are a distinct type of dynamic algorithm where the parameters of the algorithm and the scheduling strategy itself are altered based on the global state of the system.

- *Periodic Load Balancing*

Periodic load balancing usually employs distinguished agents to assemble and allocate load information, reduces communicational overheads, and improves scalability. Periodic load distribution policies impose less hindrance with centralized systems as compared to the distribution system; therefore, they can support load distribution in larger systems.

- *Non-Periodic Load Balancing*

In a non-periodic load balancing, load information reaches the load balancer in a non-periodic fashion, i.e., at irregular intervals of time purely as per the requirement of the system. Thus, it is most suitable in a dynamic environment as a VM need not wait for its allocation to a newly arrived task.

- *Advance Load Balancing*

In advance load allocation, the tasks are assigned to the VMs in advance before execution. Generally speaking, load-balancing algorithms can also be categorized as hierarchical, decentralized, or centralized depending on where migration decisions are prepared [61], [62].

- *Centralized Approach*

In a centralized approach, a central controller VM is chosen among the VMs in the distributed system. This central controller VM has the total sight of the system load information. Furthermore, it elects the way to allocate jobs to other VM.

- *Decentralized Approach*

In a decentralized approach, all VMs in the distributed system contribute to making load-balancing decisions. Since load-balancing decisions are distributed in nature, which makes it is costly for each VM to obtain the dynamic state information of the total view.

Some of the advantages that inspire the implementation of load balancing in the Cloud are as follows:

Efficient VM Utilization in a Cloud Environment: In the cloud, VMs may be inadequately loaded further the general performance of the system will be affected. Moreover, the selected competitive VM can be highly utilized while the other VM may remain idle throughout the process and the underutilized VM may wait for a task. This scenario results in higher processing time and maximizes waiting time. To overcome such inconsistencies, VM utilization needs to be efficient by optimally balancing the load among resources.

Adequate Load Distribution: Ample Load distribution is necessary to attain the best possible performance of the system. It leads to utilizing the maximum computing capability of a particular VM and parallel task execution. Likewise, it ensures an adequate load allocated to every single VM according to its capacity in all conditions. It is necessary to dispense workload among all VMs uniformly according to their processing capacities to diminish the task execution time to the meanest possible value.

Minimization of Response Time: Inappropriate load distribution leads to several disparities resulting in higher response time which eventually results in an inconsistent state of the system. Thus, it is crucial to realize optimal load balancing to minimize the response time and achieve enhanced system throughput.

1.3.3.2. Challenges of Load Balancing in Cloud

CC encounters numerous challenges, with LB standing out as a particularly crucial issue that requires focused consideration. This encompasses concerns like (VM) migration, the

security of virtual machines, and the equitable prioritization of user QoS security and resource utilization. Efforts are directed toward finding optimal solutions to enhance the efficiency of resource utilization in the cloud. Some of the LB issues and challenges faced by the researchers are listed below [63]:

- *Geographical Node Distribution*

Cloud data centers are commonly dispersed across various locations to facilitate computing. Within these centers, dynamically distributed nodes serve as a centralized network, ensuring the effective processing of customer requests. While the literature provides various Load Balancing (LB) approaches, many have limited applicability, as they often overlook factors such as network delay, communication delay, the spatial distribution of computing nodes, and the availability of space and resources within the customer environment. Nodes located in extremely remote areas pose challenges due to the inadequacy of certain algorithms that are not well-suited to such environments [63].

- *Single Point of Failure*

"In the literature, specific LB algorithms are suggested where decision-making is not distributed across multiple nodes; instead, LB decisions are centralized to a single node. The potential drawback of such an approach is that if the central node or key devices experience malfunctions, it can significantly impact the overall performance of the computing system [63].

- *VM Migration*

Virtualization enables the creation of multiple virtual machines on a single physical unit, each with distinct settings and independent architectural structures. In cases of physical device overload, it is advisable to employ an LB method to seamlessly transfer all VMs to a remote setting, ensuring efficient resource allocation and system optimization [63].

- *Node Heterogeneity*

The use of homogeneous nodes for cloud load balancing is suggested in the literature. However, for a more efficient network and reduced response time, there is a need for a dynamic switch executed on heterogeneous nodes to cater to the diverse requirements of cloud computing consumers [63].

- *Data Handling*

Cloud computing not only addresses the challenges posed by outdated storage infrastructure but also introduces scalability and redundancy mechanisms. This transformative shift allows users to harness the expanding storage capabilities efficiently while maintaining data

integrity through duplication strategies, thereby ensuring a reliable and resilient storage environment [63].

- *Scalability*

The accessibility and on-demand scalability of cloud services empower individuals to swiftly adjust resource allocation, enabling rapid downsizing or scaling up as needed. An effective load balancing mechanism should take into account the dynamic changes in computational requirements, memory usage, device topology, and other factors to ensure optimal performance in response to evolving conditions [63].

- *Computational Complexity*

Cloud Computing (CC) algorithms should prioritize speed and simplicity for efficient implementation. The primary goal of a robust algorithm is to enhance the efficiency and quality of the cloud system, as outlined in [63].

- *Programmed Service Provisioning*

Central to cloud computing is its inherent flexibility that allows resources to be automatically allocated or distributed. The challenge lies in leveraging and deploying cloud services while maintaining productivity comparable to traditional systems and optimizing the use of available resources [52].

- *Energy Organization*

Efficient energy management in cloud computing not only fosters cost-effectiveness but also facilitates a collaborative approach to global resource utilization. By prioritizing power-saving measures, the cloud enables businesses to contribute collectively to a shared global capital pool, fostering sustainability and optimizing resource allocation [52].

1.4. Objectives of the Research

The following are the listed objectives formulated for the research:

- To study and analyze the existing VM reservation and fault-tolerance techniques for resource provisioning in the cloud environment
- To design and implement an efficient scheduling technique with fault tolerance for optimal workload and resource reservation.
- To optimize QoS parameters for the proposed approach under the cloud environment.
- To execute the comparative analysis of the proposed approach with the existing approaches.

1.5. Thesis Organization

The thesis is organized in different chapters. The flowchart of the thesis is presented in Figure 1.12.

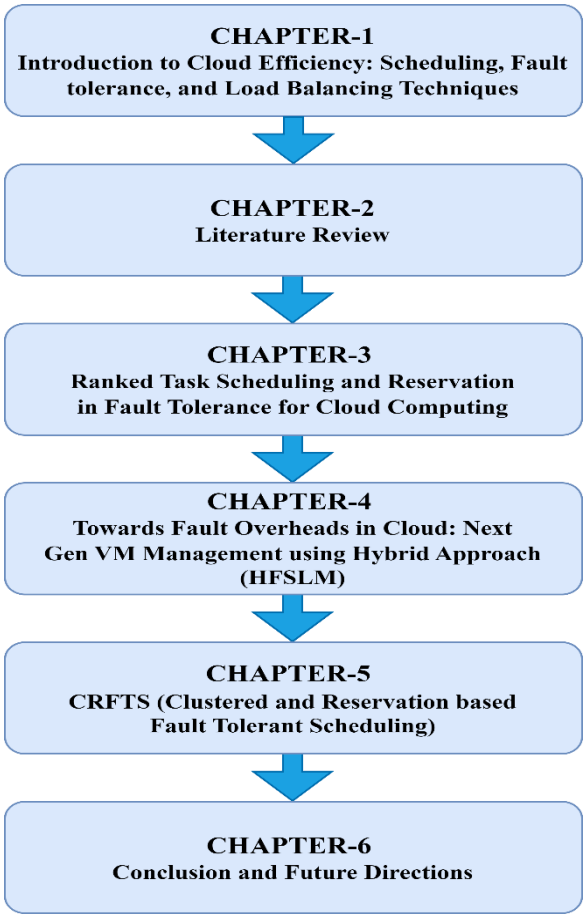


Figure 1.12: Thesis Organization

1.6. Summary in Context

In the realm of cloud computing, the seamless operation of systems hinges on effective task scheduling, fault tolerance mechanisms, and load-balancing strategies. Task scheduling and load balancing play an important role in resource provisioning in the cloud. Task scheduling involves the efficient allocation of computational tasks to available resources, ensuring optimal performance and resource utilization. Load balancing plays a critical role in distributing workloads across multiple servers or virtual machines, preventing overloads, and maximizing system efficiency.

Additionally, fault tolerance mechanisms are essential safeguards against system failures, enabling continuous operation by detecting, isolating, and recovering from faults.

Together, these components form the backbone of reliable and efficient cloud services. They enable scalable and resilient architectures that meet the demands of modern applications and users, ensuring high availability, responsiveness, and adherence to service

level agreements (SLAs). As cloud environments evolve to handle diverse workloads and dynamic resource demands, the integration of robust task scheduling, fault tolerance, and load balancing becomes increasingly crucial for sustaining optimal performance and reliability.

Chapter 2
Literature Review

A comprehensive review of the literature is essential to inspire the development and bridge the gap between existing and proposed research. This chapter delves into various techniques operating under similar conditions, examining their associated strategies to assess the issues and limitations inherent in current research endeavors. Furthermore, it is imperative to pinpoint the specific problem that future research endeavors can address to resolve or mitigate existing issues. Despite numerous studies already conducted in the related area, identifying gaps and potential avenues for further investigation remains essential. Hence, the goal of the present study is to uncover the constraints that can enhance the global productivity, execution, and accuracy of the cloud. Various algorithms have already been reported in the literature to allocate resources dynamically in the cloud environment. However, adding fault tolerance with scheduling and load balancing is also one of the primary challenges to working in the cloud environment.

This chapter comprises an extensive literature survey with hybrid objectives. Firstly, it aims to provide a literature review focusing on existing fault-tolerant techniques. Afterwards, it explores the literature related to fault-tolerant techniques integrated with two primary resource provisioning methods, namely scheduling and load balancing. The examination of fault-tolerant algorithms in conjunction with scheduling and load distribution methods is pivotal for optimizing task-resource mapping in the context of a dynamic and heterogeneous environment. This comprehensive analysis contributes to the understanding of strategies that can improve overall system efficiency and reliability.

Moreover, a kind of hybrid review is performed in this chapter by focusing on some other aspects simultaneously, such as Load balancing and Scheduling with fault tolerance. Fault tolerance techniques presented so far are reviewed based on considered parameters such as techniques like fault tolerance with scheduling, fault tolerance with load balancing, fault tolerance with QoS parameters, etc. Scheduling of tasks appropriately finds it good in delivering critical and proper services of the cloud. The improper scheduling of tasks may result in under and over-utilizing of resources where the under-utilization of resources can lead to the wastage of resources and over-utilization of resources can degrade the performance of cloud systems. Cloud Systems loaded with Load Balancing techniques reduce receiving and sending delays and prevent the nodes from overloaded situations as well [22]. So, there is a good need to solve the Load Balancing issues to boost the overall performance of cloud-based applications.

The advancement in cloud computing technology has reformed the approach computing assets are provisioned, utilized, and managed. Cloud computing offers a vast array of services that are flexible, scalable, and cost-effective. To improve the utilization of cloud resources, various dynamic resource allocation algorithms have been intended in the works. However, ensuring fault-tolerant scheduling and load balancing is a critical challenge that needs to be addressed to provide uninterrupted services in the cloud. Virtual machine reservation is one of the promising approaches that can mitigate these challenges by allocating reserved resources for fault tolerance and load balancing.

2.1. Research Methodology and Data Analysis

This section focuses on the setting of the methods that are used to perform the qualitative opinion of the literature in the review and the sources of considered state-of-the-art works. It also includes the incorporated methodology for the proposed research. In the end, we specified our significant contributions to this review.

SLR and Kitchenham standards are employed for review which also includes the selection and elimination of the published articles based on some aspects. The related articles were selected after analyzing the abstract, and afterward, a critical review/analysis was performed. The selection of the papers was achieved based on the standard in the database and the article itself. Furthermore, the inclusion was done based on the following conditions.

Searching Strategy

A systematic survey of fault tolerance with efficient scheduling and load distribution techniques proposed in the literature was conducted through well-known sources.

Several search keywords include Cloud Resources, Fault-tolerance, Task Scheduling, Load Balancing, QoS Parameters, Resource Optimization, failure in a cloud, essential cloud services, cloud architecture, scheduling techniques, etc., used in this study.

Duration and Validity of Study

- This review research mostly incorporates articles from 2009 to 2023 from well-believed journals, books, and conferences.
- The statistics of the considered year for publications from 2009 to 2023 are depicted in Figure 2.1.
- Very few studies are included from 2007 and 2008.
- The selected duration is chosen to capture a comprehensive range of data such as technical progressions, economic sequences, and policy variations, and confirm data availability pertinent to our study that replicates the evolution, progression, and trends applicable to our study objectives.

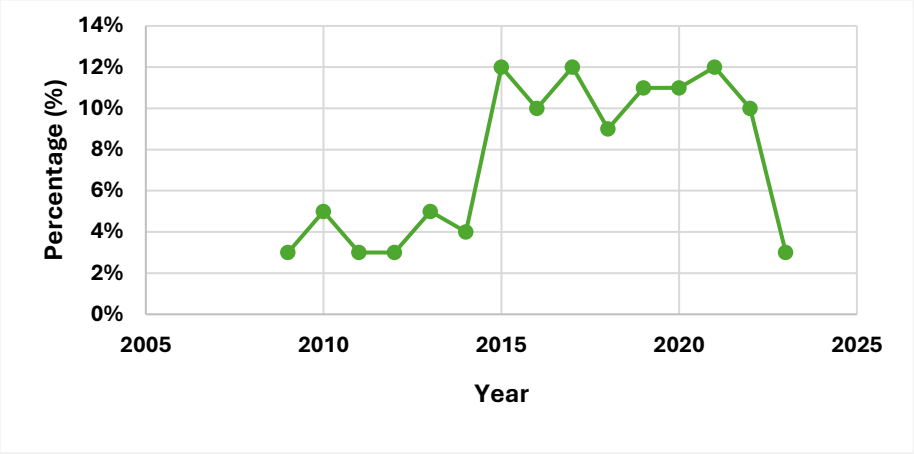


Figure 2.1. Percentage of the Included Papers (2009 to 2023)

Language and Selection/Inclusion Criteria

- The decision for the language criterion was specified as English. Because English is considered the primary language for scholarly and intellectual publications particularly in the fields of computer science and distributed computing. Regulating criteria for English articles ensured that we selected high-quality and broadly recognized studies, smoothing a thorough and appropriate review.
- The primary priority was given to hybrid fault tolerance approaches including either scheduling or load balancing.
- Hybrid fault tolerance approaches optimize some other QoS parameters as well. Figure 2.2 presents the detailed inclusion and exclusion of the studies.

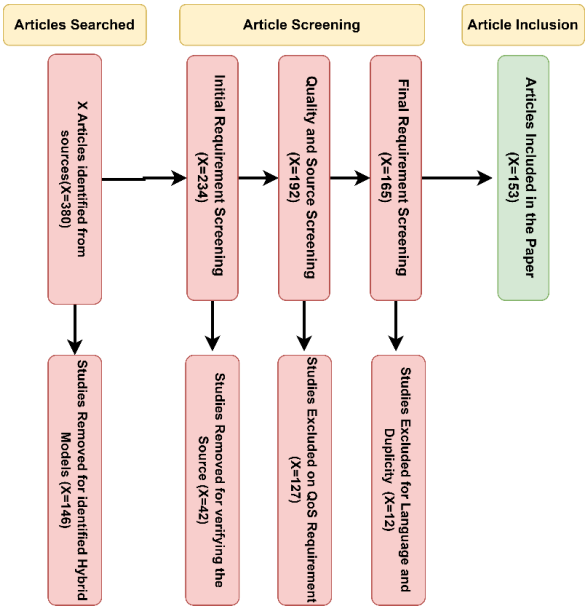


Figure 2.2. Methodology of Inclusion and Exclusion Criteria of the Studies

Data Processing and Analysis

- The data was initially organized into Excel and prepared for analysis.
- Data categorization was made based on different QoS parameters, the environment used for simulation, types of faults considered, and other thematic considerations. This categorization helps us to analyze the literature more clearly and precisely.
- The qualitative information was obtained by considering diverse QoS metrics, types of faults addressed, and the range of simulation environments utilized across a timeframe.
- Furthermore, the analysis also highlights the various fault tolerance methods employed in the existing literature.

Synthesis of the Analysis

- For meaningful conclusions and insights, the data was observed based on the objectives of the study.
- The patterns and relationships among the various studies were discussed for comparison and assessment.

Quality Assessment and Validation Procedure

The presented Methodology Adapted for this study can be summarized in four stages:

- Originally, the related articles were searched through the related keywords.
- Some articles were selected based on title, standards, and optimization parameters.
- Selected articles were gone through the abstract, and further inclusion and exclusion were performed.
- Finally, inclusive articles were extensively reviewed, analyzed, and incorporated into this survey.

2.2. Our Contribution and Features of the Study

The primary contributions of this survey include:

- This chapter presents an in-depth examination of the cloud environment. The main faults and fault taxonomy in cloud systems are also discussed in detail.
- Various researchers have already addressed fault tolerance and load balancing mechanisms, however, much of their work has focused on the employment of either fault tolerance or load balancing separately. The presented survey incorporates a review of fault tolerance with two other related aspects, i.e., load balancing and scheduling which is the peak need of the time and was found missing in the current surveys.
- Moreover, Table 2.1 presents a comparative analysis of our contribution to the recent and current top-cited studies respectively.

- The survey has been presented in two categories i.e., Fault tolerance with Scheduling and Fault tolerance with Load balancing.
- The generalized problem formulation of fault tolerance has also been presented to understand the workings of fault tolerance using the replication technique.
- We further outlined the difficulties associated with ensuring fault tolerance integrated with scheduling and load balancing in cloud computing systems and comprised a thorough examination of common problems faced. It will assist future researchers to promptly recognize or understand the problems related to the study.
- The study also presents feasible graphical observations about the literature such as parameters optimized, faults model addressed, the environmental tool used, etc. These detailed observations are presented separately for both categories and were not found in the existing surveys to the best of our knowledge. A dedicated discussion and observation section is designed for that purpose.
- This hybrid review aids in investigating the potential challenges of hybrid fault-tolerant models and provides a detailed roadmap for future research directions. The aim is to enhance migration methods, thereby mitigating failures among nodes.
- Moreover, the overall study provides a platform for future researchers to analyze the current state of the art regarding considered issues and find the appropriate future research problems.
- At the end of this chapter, there is a dedicated section highlighting the future research directions of the problem.

Table 2.1: Comparative analysis related to the contribution of the top-cited study and the proposed study

Authors	**Year**	**Fault Taxonomy**	**Fault Tolerance**	**Fault Tolerance Approaches**	**Load Balancing**	**Load Balancing Approaches**	**Scheduling**	**Hybrid review of Scheduling and Fault Tolerance**	**Hybrid review of Load Balancing and Fault Tolerance**	**Fault Tolerance Problem Formulation**	**Graphical Representation of Results**
[29]	*2021*	√	√	√	×	×	×	×	×	×	√
[28]	*2021*	×	×	×	×	×	√	×	×	×	×
[64]	*2021*	×	×	×	√	×	×	×	×	×	×
[58]	*2018*	√	√	×	×	×	×	×	×	×	×
[65]	*2021*	×	√	√	×	×	×	×	×	×	×

[66]	*2022*	×	√	√	×	×	√	×	×	×	√
[67]	*2020*	×	×	×	√	×	×	×	×	√	√
[68]	*2019*	×	×	×	√	√	×	×	×	×	√
Presented Survey	–	√	√	√	√	√	√	√	√	√	√

2.3. Our Motivation and Main Focus of the Study

Faults can lead to malfunctions that worsen a system's overall performance. Failures result in the breakdown/shutdown of a system, but occasionally, flaws cause performance to decline rather than the entire shutdown of the system. Various fault tolerance solutions can be employed to address different types of defects, such as network, physical, and process problems. However, it is crucial to achieve without comprehending the existence of the issue inside the architecture and the damage that the system flaw produced. Cloud is made up of comprises levels, each of which takes services from the layers below it. The failure at any layer has the potential to contaminate the layer right above it. Since faults at any one layer may affect the services that any of the layers provide. Thus, for high-performance computers, the appropriate fault tolerance system is needed to effectively handle these faults. The faults should be managed critically and dynamically to make the cloud environment more efficient and intelligent. Besides, in the cloud, efficient task scheduling leads to the maximum utilization of virtual machines, reducing operational costs, thereby revealing enhancements in the QoS parameters and eventually improving overall performance. Also, load balancing techniques need to be addressed comprehensively in different environments like static, dynamic, and nature-inspired cloud environments. Moreover, it is essential to thoroughly examine load-balancing techniques across various settings, including static, dynamic, and nature-inspired environments.

Various methods have been suggested in academic literature to address this concern and multifarious reviews are available in the literature for future researchers. While studying the existing surveys, it was observed that the surveys are not thorough enough, wide-ranging, and sufficient in certain ways. Although the authors in reference [10] have presented a comprehensive survey about fault tolerance, this survey does not focus on other aspects of the cloud like efficient load balancing and scheduling. Besides, [64] presented a vast survey focusing on load balancing across cloud resources but lacking in fault handling and cloud optimization. Similarly, [22] also provides a survey emphasizing fault tolerance frameworks, however, fails to significantly enhance the performance of the cloud

environment. In [65], only considering fault-tolerant approaches does not give prominence to major cloud aspects such as scheduling and load balancing. Similarly, the most recent survey presented in [66] focused on both scheduling and fault tolerance but no ways for optimal load distribution. Additionally, the observations presented in [67] were limited to a few aspects concerning fault-handling techniques, and only crash and byzantine fault models were considered. Also, there is no consideration of QoS parameters. Similarly, the recent survey was presented in [69] but was found limited to reliability. In other words, these reviews were not significantly focused on the discussed issues of the cloud related to fault tolerance with scheduling/load balancing simultaneously. After this comprehensive analysis, it was observed that none of the mentioned surveys offer extensive consideration of the above-mentioned scenarios of cloud computing. The QoS and other important aspects related to the clouds' fault tolerance concerns are focused on by the researchers in the existing surveys but are very limited. This renders the current review inadequate for analyzing the current art in cloud systems. Hence, there is a dire need to present a survey focusing on reliability-related aspects of the cloud. Therefore, we got motivated and moved to present this systematic and hybrid review. In this survey, we try to discover and explore the site of hybrid fault tolerance models that will focus not only on traditional fault tolerance techniques but also integrate some other important cloud aspects like scheduling/load balancing. This integration helps us to highlight the likely applications, challenges, and incipient trends.

2.4. Related Literature

Fault tolerance techniques presented so far are reviewed based on considered parameters such as techniques like fault tolerance with scheduling, fault tolerance with load balancing, fault tolerance with QoS parameters, etc. Scheduling of tasks appropriately finds it good in delivering critical and proper services of the cloud.

2.4.1. Scheduling with Fault-tolerance

Efficient scheduling in the cloud provides optimization of various Quality of Service parameters, especially task completion time. Besides, scalability, availability, security, and fault tolerance are the key features of cloud services. Instead of the complete breakdown of the system, the faults in the cloud lead to performance degradation only. Without fault-tolerant scheduling when one or more components of the system fail, the task execution, waiting time, response time, etc. may increase. This leads to enhanced throughput as well. However, Fault tolerance provides an alternative way for the process completion even if some of the resources may not work properly [37], [38]. Few works of literature have

proposed fault-tolerant scheduling algorithms with optimized parameters. Recently, in [50], the Dynamic Clustering Cuckoo Whale Optimization Algorithm (DCCWOA) has been suggested for supporting effective fault-tolerant scheduling in the cloud. The algorithm was tested for varying the tasks between 100 to 1000 with 8 virtual machines. The problem of fault tolerance was also investigated in [51], and a greedy-based best fit decreasing (GBFD) algorithm was proposed for increasing the success rate of task execution along with optimization of other parameters. The model was valued with numerous loads of PUMA datasets. Additionally, the computational complexity was claimed to be O(nm) where n is the VM number in the data center, and m represents computing nodes. In [70], authors proposed GWO (Grey Wolf Optimization) - based Task Scheduling evaluated on the 1000MI task dataset. Fault handling is carried out in the proposed work with efficient task scheduling by employing the task resubmission technique. Extending the chain of work and solving the problems of dependability relationships, learning automata was used and a self-adapting scheduling strategy namely, ADATSA was proposed in [71]. The model was experimentally evaluated on 53 servers with 3 Master nodes and 50 slaves. The complexity was proposed to be O(NK) + O(MS) where N represents cluster nodes, K represents resource category, M is average tasks on a node, and S is average state transitions. In [72], a Fault-Tolerant Hybrid Resource Allocation Model (FTHRM) was recommended which confirms fault tolerance and minimized Turn-around-Time (TAT). The proposed model employs a prior reservation process to distribute resources to the respective tasks, ensuring the guaranteed execution of tasks. Resource reservation is also enabled for time slots with resource organization as needed by the task set with adjusting VM heterogeneity. In case of resource failure, alternative resources are being supplied where the most preferred resource has having least former workload and the smallest execution time. The authors in [73] presented the framework for adaptive scheduling and fragmentation of tasks namely (WSADF) Workflow-scheduling applying -adaptable and dynamic-fragmentation which initially creates the fragments concerned with the number of VMs in the fragmentation phase and later the scheduling phase pick out the VMs concerned to reduce the usage of bandwidth. WSADF was evaluated on the workload ranging from 25 to 1000 and VMs ranging from 5 to 25. While making the task scheduling adaptable to both heterogeneity and homogeneous environments, CPSO and FIPS were proposed in [74]. The proposed task scheduling was evaluated on 30 servers under 1000 iterations. In this chain to integrate localized edge clouds with publicly accessible clouds and enhance scheduling effectiveness and scalability, a hierarchy-based edge cloud concept was introduced in [75]. Additionally, FTDS, a failure rescue technique is suggested to address the fears that arise while mobile

apps are being executed. For evaluation, the workflow was taken from 10 to 70 applications while taking the length of the workflow from 10 to 60. Besides, some of the SLA (Service level agreement) parameters like, CPU necessity, system bandwidth, and memory need to be considered with appropriate scheduling. In this regard, the pre-emption-based algorithm was proposed in [53] which pre-empts the resources from the low-priority task to the high-priority task in case of unavailability of the resources and provides reservation of resources reflecting numerous SLA parameters for facility deployment. The evaluations were carried out via 4 cloud simulations by performing 10 consecutive runs and 60 requests having 10 to 15 subtasks. The cost and deadline of the tasks are considered for defining the priority of the tasks. Moreover, it provides dynamic resource provisioning and an effective fault tolerance process. In this chain, a fault-tolerance aware task scheduling scheme was proposed in [55] namely Checkpointed League Championship Algorithm (CPLCA). This algorithm provides fault tolerance using the checkpointing strategy along with task migration and was evaluated by using workload in the form of Standard Workload Format accessible via the San Diego Supercomputer Center (SDSC). Efficient scheduling and fault handling mutually may ensure task execution and thereby fulfill the real-time environment of the cloud. However, heterogeneous systems and their complexities are increasing dramatically leading to failures. These failures can be eliminated by implementing efficient scheduling approaches. Therefore, the task scheduling problem on heterogeneous systems was addressed in [56]. Being an NP-hard problem, a heuristic algorithm Deadline Based Scheduling Algorithm (DBSA) was proposed to resolve it. The DBSA approach dynamically estimates the figure of permanent tolerating failures by calculating the makespan first till the system tolerates a fixed number of failures. Afterward continuously comparing the makespan with the specified deadline to get the successive number of tolerating failures. The model was evaluated in the workload ranges from 20 to 100 with 4 and 8 VMs. Gaussian Elimination, Fast Fourier Transformation, and Molecular Dynamics Code are used as a kind of application graphs for testing. Finally, the task is mapped to the appropriate processor without violating precedence constraints. Further, in [76] Cost-effective, NNCA_PSO was proposed by modifying Particle Swarm Optimization (PSO). During evaluations, the workload was varied from 70 to 560 and VMs were used from 4 to 8. Further, the Advance Reservation Fault Tolerance Model (ARFTM) was proposed in [77] which maps the tasks using MCT and tolerates faults using the advanced reservation technique. ARFTM was evaluated by varying the workload from 1 to 300.

However, in [78], the fact that “the network bandwidth is limited” and the scheduling policies should decrease the bandwidth usage in cloud computing was considered.

Moreover, the author proposes a data locality-based task scheduling approach, i.e., the Balance Reduce Algorithm (BAR). It will reduce network access and thereby reduce bandwidth usage and job completion time while not specifying the type and nature of workload used for evaluation. Furthermore, an improved Balance Reduce Algorithm was proposed with an improvement in machine failure handling. Later in [17], fault tolerance-based scheduling was proposed namely the Dynamic Clustering League Championship algorithm (DCLCA) to reduce the premature failure of the tasks. The model was evaluated in two scenarios where a parallel workload archive containing 73,496 tasks in the form of Standard Workload Format accessible via the San Diego Supercomputer Center (SDSC) was used in the first scenario. In the second scenario, workloads were produced as of CloudSim's Workload PlanetLab. All the surveyed methods are brief in Table 2.2.

Table 2.2: Comparative analysis of recent scheduling-based fault tolerance algorithms

Method	**Year**	**Parameters**	**Comparison Approaches**	**Outcomes**	**Limitations**	**Platform /Environment**
HFSLM [34]	2024	Makespan, Average Resource Utilization	Maxmin, Minmin, FTHRM, OLB, ELISA, MELISA	Efficient Resource utilization and makespan	No security was considered	Self Simulator
ARFTM [142]	2023	Reliability	MCT	Highly Reliable	Inadequate Load distribution	Self Simulator
RFRTS [79]	2024	Reliability	FCWS, FR-MOS	Reliability with varied load	No security	Self Simulator
DCCWOA [50]	2023	Makespan, Failure Ratio, and Failure Slowdown	ACO, GA, and LCA	58.19%, 19.88%, and 29.32% Makespan, Failure Proportion, and Failure Strike parameters respectively.	Limited optimization of QoS parameters	Cloudsim toolkit
(MSMO classifier) Modified Sequential Minimal Optimization accompanied Delta-Checkpoint [80]	2023	Accuracy and Prediction of Faults with reliability	Related ML-based Classifiers	Enhanced credibility for reliability	Reliability was not proved while comparing with MSMO	cloud simulation 3.0.3

GBFD [51]	2022	SERoV, Average Expenditure, Average Completion Time	FCFS algorithm, Cost-Greedy Dynamic Price Scheduling (CGDPS) algorithm [4], Modified Best Fit Decreasing (MBFD) algorithm	Optimizes performance of the cloud systems.	Lacks dynamic resource utilization and uniform load distribution	Cloudsim toolkit
GWO-based Task Scheduling [70]	2022	Makespan, Execution time, Communication delay	ANGEL, TTSA (Temporal Task Scheduling Algorithm), MapReduce Scheduling, and Dynamic Slot Scheduling	Effective task scheduling with fault tolerance is achieved with optimized parameters.	Evaluations were carried out only on four tasks	CloudSim, JDK7.0 and Eclipse
ADATSA (Self-adapting Task Scheduling algorithm) [71]	2022	Adaptability in circumstances, optimization of resources, and QoS	LAEAS, PSOS, and K8S scheduling engine	Better adaptability and QoS	Lack of heterogeneity in VMs	Amazon EC2 and Apache JMeter(v 5.4.0)
Fault-Tolerant Hybrid Resource allocation Model (FTHRM) [72]	2021	Turnaround Time, Flow Time, Resource Utilization	MCT	FTHRM improvises TAT from 32 to 40%, Lowers Flow Time to 26 to 45%, Provides 15 to 27% better average resource utilization than traditional MCT	The proposed system was not fully dynamic concerning the nature of tasks.	Simulation via C-Language
WSADF [73]	2019	Adaptability, Response time, Throughput	FPD in the fragmentation phase, CTC, SLV, and QDA in the Scheduling Phase	Adaptable to the environment, improvements in response time and Throughput.	Increased delays and average response time which eventually reduces throughput and efficiency	CloudSim Simulator
Canonical Particle Swarm Optimization (CPSO), Fully informed particle Swarm Optimization (FIPS) [74]	2019	Throughput, Utilization, Adaptability	CPSO in h-DDSS (Heterogenous Dynamic Dedicated Server Scheduling) and DDSS (Dynamic Dedicated Server Scheduling)	Scheduling is adaptable to both heterogenous and homogenous environments	May not manage the real-time data	Not specified

			FIPS in h-DDSS and DDSS			
Fault-Tolerant Dynamic Scheduling (FTDS) [75]	2019	Scalability, Success rate, Competitive Ratio	UES, IC-PCP in LIGO and Epigenomics	Improvement in scalability and performance, Achieves the trade-off between cost and system delay.	May consume energy	Amazon T2, RWP Model
Dynamic Clustering League Championship algorithm (DCLCA) [17]	2018	Makespan	MTCT, MAXMIN, Ant Colony Optimization, and Genetic Algorithm-based NSGA-II	In the case of 5 cloud users with 5 and 2 brokers and data centers respectively, DCLCA lowers makespan with an improvement of 57.8, 53.6, 24.3, and 13.4 %, and in the case of 10 and 5 cloud users and data centers, DCLCA shows improvement of 60.0, 38.9, 31.5 and 31.2 %	A limited number of cloud users, brokers, and data centers were considered.	CloudSim 3.0.3 toolkit with Eclipse Luna 4.4.0
Deadline Based Scheduling Algorithm (DBSA)[56]	2018	Makespan, Reliability, and PSS (possibility of Scheduling Success)	HEFT and FTSA	DBSA can successfully endure crashes and enhance reliability within time constraints.	Limited optimization of QoS parameters	Not Specified
Nearest Neighbour Cost-Aware Particle Swarm Optimization (NNCA_PSO) [76]	2018	Scalability, Makespan, and Monetary Cost	PSO and CA_PSO	High Scalability, Low Makespan, and Monetary cost	The model is less reliable	CloudSim toolkit
Checkpointed League Championship Algorithm (CPLCA) [55]	2017	Makespan, and Response Time	Ant Colony Optimization (ACO), Genetic Algorithm (GA), and the basic League Championship Algorithm (LCA)	CPLCA scheme produces an enhancement of 41%, 33%, and 23% on Makespan, and 54%, 57%, and 30% improvement in Response Time	Insufficient load balancing for a dynamic system.	CloudSim 3.0.3 toolkit has a modified CloudAnalyst GUI interface.

				than ACO, GA, and LCA respectively.		
Improved BAR (Balance Reduce Algorithm) [78]	2012	Task Completion Time	BAR (Balance Reduce Algorithm)	Minimizes Makespan even in case of failure by fault tolerance.	Not suitable for heterogeneous environment	Cloudsim

2.4.1.1. Scheduling and Fault Tolerance Frameworks

Various scheduling and fault tolerance frameworks are recommended in the literature. In this section, these frameworks are surveyed and presented. Comparative analyses of different scheduling and fault tolerance frameworks are presented in Table 2.3.

Proactive-based Scheduling and Fault Tolerance Frameworks

In this approach, the system can handle any disruptions or interruptions. The state of the system is monitored continuously for breakdowns and failure. Some of the proactive-based scheduling and fault tolerance frameworks found in the literature are mentioned below:

SHelp [81]*:* This approach was proposed by improving the existing framework namely, ASSURE [82] which was implemented at the rescue points.

PFHC [83]*:* This is a proactive framework of fault tolerance proposed for HPC (High-Performance Computing) applications in the cloud. This framework works on three chief modules: Node Monitoring Module is prepared with some special Lm-sensors [84], [85] to perform periodic monitoring for several parameters such as fan speed, CPU temperature, etc., for wellness.

WSRC [86]*:* This framework contains a module namely, a failure detector that checks the Virtual Machine Manager (VMM) periodically for any kind of variations such as delay in response time or mismanagement of memory. If any fluctuation is found, the VM running status is saved and VMM is repaired using the rejuvenation technique. Rejuvenation generally leads to high overheads however, WSRC uses variable time rejuvenation to control overheads.

SRFSC [87] *:* The software rejuvenation technique was used in this framework. This framework primarily works in three phases: In the first phase, the packet that has the information about the CPU and VM's memory usage is received by Aging Failure Detection. The other step is the evaluation of VM for failing grades. This step is known as Aging Degree Evaluation.

FTDG [88]*:* FTDG is a fault-tolerant framework where the pre-emptive relocation is being achieved. The architecture of this framework mainly comprises four functioning spaces. User Space is used by the user to submit their data flows. Graph Space transforms the submitted user data into Direct Acyclic Graphs (DAG). Moreover, the DAG is analyzed for the critical and non-critical paths. In Storm Space, Scheduling and fault tolerance mechanisms are applied. Hardware Space contains various data center resources.

Reactive-based Scheduling and Fault Tolerance Frameworks

In such frameworks, the faults are handled once they occur. Unlike proactive approaches, monitoring of system behavior is not required in such frameworks. Some of the Reactive-based scheduling and fault tolerance frameworks found in the literature are mentioned below.

AFTRC [89]*:* In (Adaptive Fault Tolerance in Real-time Cloud Computing), the received tasks are held in some input buffer and the task execution will be accomplished on a First Come First Serve basis. This model also consists of the other modules. The Acceptance Test (AT) is the module that checks the results of each embedded algorithm for accuracy and verifies the results. The Time Checker (TC) checks whether the result is obtained within the deadline or not. If not, then the specific task is sent back to the input buffer. The Reliability Assessor (RA) adjusts the reliabilities of VMs based on obtained results. The decision Mechanism (DM) takes the highest reliable node and selects the output from that.

BFTCloud [90]*:* This framework uses replication techniques and completes the user requests timely even in the presence of faults. The amount of replicas/nodes is utilized by employing the failure probability of all nodes. The failure likelihood of the replica group should constantly be less than the top-level failure likelihood. The functioning of the BFTCloud framework mainly works in five phases: *Primary Selection*: In this phase, the basal node is designated based on the rating by adding the priority weight and QoS value assigned to each node. The highest rating value node will be chosen as the primary node.

Replica Selection: In this phase, the number of replicas is selected by observing the QoS of every node from the viewpoint of both the primary node and the cloud module. The new QoS is calculated, and again rating will be done. *Request Execution:* This phase allows the nodes to complete the request and react to the cloud module accordingly. The cloud, in turn, checks the consistency of the obtained results based on different cases [91]. If the results are consistent, then the primary replica is assigned to the next request. *Primary Updating*: In case of a fault in the primary replica, this phase informs all other replicas to select the alternative. *Replica Updating*: This phase removes the faulty replica and adds the new nodes to decrease the failure probability.

FESTAL [92]*:* It is a fault-tolerant scheduling framework where the primary backup technique is realized to handle the faults. In this framework, the user tasks are queued up in some input buffer and assigned to the schedular having three controllers, i.e., Resource Controller, Backup Copy Controller, and Real-time Controller.

The Backup Copy Controller takes the backup. Afterward, the Resource Controller explores the two VMs, that can complete the task within the deadline. Based on the search results, two decisions can be made.

- *In case the two corresponding VMs are found, both task instances are scheduled on the respective VMs.*
- *In case no such VM is found, the task is rejected.*

In this framework, "If the anticipated end time is less than or identical to the task time-limit, a passive backup is utilized; otherwise, an active backup is employed.

Resilient-Based Scheduling and Fault tolerance frameworks

These techniques have some similarities to the proactive approach. Moreover, the defects are forecasted, and the effects are prevented/moderated by applying some methodologies. The forecasting uses some intelligent learning that makes resilient techniques different from proactive ones. Compared to conventional fault tolerance techniques, resilient fault tolerance provides increased durability and adaptability in the event of system breakdowns. Using resilient approaches, the system can recover swiftly and efficiently in dynamic contexts due to resilient fault tolerance, which provides a more systematic and flexible approach to addressing failures. When compared to conventional fault tolerance techniques, this strategy frequently results in increased overall performance, decreased interruptions, and enhanced system efficiency. In this context, EFTT (Efficient Fault Tolerance Technique) is a type of resilient-based approach. In [93], the author used Machine Learning to handle faults and generate solutions for FT.

Resilient Methods are of two types described below:

- *Machine Learning:*

ML was, nevertheless, applied as a sub-constituent of the general FT solution. Some solutions have intensively employed ML to forecast faults using a set of specified variables. Many applications have been working with ML while handling hardware faults. Here, artificial intelligence, or machine learning, is used to create a system that can operate autonomously like a human without the need for human concern. Machine learning procedures can be used to increase a system's reliability even in the case of fault tolerance. Such fault tolerance techniques are known as Resilient Fault-Tolerant Techniques as

discussed in Section 5.3. Machine learning techniques are typically used in proactive approaches, predicting failures before they happen by using historical system data. The Resilient techniques for fault tolerance are the emerging ones because the ML accesses data and even can learn from data. One of the similar learning methods namely, reinforcement learning was applied in [94] that studies the fitness of VMs in cloud environments. By using such types of learning, every VM participates in the learning process independently. As recommended in [95], fault tolerance in a distributed or parallel learning system is achieved by constantly tracking the input parameters in the server. Here, the entire system returns to the most recent checkpoint following an error. Checkpoints are not performed at every stage by such systems, even in the presence of a high number of calls and activity in the network. Forecasting defects are well-known in fault identification and handling, as stated in [96]. Quick error detection can prevent more serious system failures. Numerous processes make up this operation, and some of the most recent research investigations include model-based approaches that are quantitative, model-based approaches that are qualitative, and history-based. Apart from reinforcement learning, unsupervised learning is an additional technique for pattern recognition in the data without predefined output [97]. Such techniques do not allow for the estimation of the outcome since unsupervised learning lacks an output target. Instead, algorithms have chosen to depend on their expertise to pull out as much detail as they can from the data. The deep learning techniques were proposed in [98] as a rapid way to identify multicriteria errors in complicated industrialized analysis. Fault tolerance can benefit from the application of such AI-related techniques.

- *Fault Induction:*

In this Resilient technique, failures are managed by making assumptions based on the reaction of the system. Moving forward in this technique, [99] proposed that a hybrid energy system be practically used to apply the multi-source power administration technique. The analysis shows how to improve fault tolerance, scalability, efficiency, and dependability. The concepts proposed in [93] are being used by some of the most well-known firms in the world, including Google and Amazon, to increase their fault tolerance. Here the authors have employed the software namely gameday. GameDay is software intended to highlight significant shortcomings in methods for finding flaws and dependencies between different components of the system. In a GameDay scenario, team members from every level of the business must collaborate to find a solution. In the repeatable tests if everything went perfectly, then the GameDay activity will be considered successful. Similarly, the authors in [100], employed game theory and declared that the kind of smart grid operator will swiftly

supply electricity through a dispersed system. Additionally, several classifiers have been compared for metrics like accuracy and fault predictions [101].

2.4.2. Load balancing with Fault tolerance

Load balancing with fault tolerance is a significant dispute in cloud computing. The efficient load balancing techniques require the inclusion of fault tolerance capacity as well. It enables the system to distribute the load to all the available nodes uniformly and simultaneously deals with detecting and removing the faults to maximize the performance and efficiency of the cloud environment. Various algorithms are surveyed and presented in Table 2.4**.** The authors have introduced Honeybee Inspired-Load Balancing (HBI-LB) in [102], a reliable and nature-inspired Fault Tolerant load-balancing approach. The assigned tasks in the suggested method were in the range of 100 to 500 in number and 2000 to 10000 in length. Further 10 and 15 fog centers and fog nodes were utilized respectively. The information of scheduling the other in-progress tasks about the status and load on the resources is given by other assigned tasks in the same way as the honeybees inform buddies about their position. Besides, in [103], the Proactive and Reactive Fault Tolerance Framework (PFTF) was proposed with ECB (Elastic Cloud Balancer). It avoids the situation in the cloud where some nodes are idle or minimum loaded, and some are overloaded. The proposed ECB enhances the scheduling quality in combination with the Job Shop Scheduling by considering and optimizing QoS parameters. The model was evaluated by taking the tasks in the range of 9 to 13 and task size in the range of 1000 MB to 8000 MB. Additionally, due to the dynamic nature of cloud infrastructure, real-time features such as availability and reliability need to be achieved. In this chain, Proactive Load Balancing Fault Tolerance (PLBFT) was proposed in [104] as an efficient fault-tolerant load-balancing model evaluated on the private cloud platform. This model relies on migrating the faulty VM to another destination host directly. Besides, the load in the destination VM is managed (in case of overload in the destination VM) before migrating the defective VM there. Furthermore, this approach shows high reliability as compared to other similar techniques. Load balancing and fault tolerance techniques are designed to provide highly reliable and available services. For further growth in the availability of cloud services, a combination of load-balancing and fault-tolerant techniques has been proposed [105]. The proposed model is highly reliable in case of task failure while taking the task number between 13 to 18, task execution time between 1 to 9, and task priority between 1 to 3 with four VMs. Moreover, in [106], Deadline Pre-emptive Scheduling (DBPS) was proposed based on cloud partitioning where the fault tolerance has been achieved by Throttled Load Balancing for Cloud (TLBC). The model was tested on a workload of 10 to 300 while not specifying the

number of VMs. However, a Machine learning-based approach was proposed in [107], namely, Fault-tolerance Load Balancing (FTLB), which embeds fault tolerance in load balancing with the optimization of other QoS parameters. The evaluation was performed using 100 computing cycles on three VMs. Furthermore, an Integrated Virtualized Failover strategy (IVFS) similar to AFTRC was proposed in [108]. It employs replication and checkpoint-restart in which Cloud Load Balancer (CLB) was added to AFTRC, and checkpointing was carried out by implementing the Reward Renewal Process (RRP) [109]. Once the load was received, it was transferred to CLB by the Cloud Controller (CC). The main job of CLB was to replicate the load on some suitable VM based on load information in case of failure.

Table 2.3: Comparative analysis of various fault tolerance and scheduling frameworks

Framework	**Approach**	**Used Techniques**	**Parameters**	**Features**	**Limitation**
Self-Healing (SHelp) [81]	Proactive	Self-healing, Restarting, Checkpointing,	Response time, Throughput, Availability	Speedy functionality, Fewer overheads than ASSURE	Not suitable for software faults
PFHC [83]	Proactive	Replication	Execution Time, Reliability	Lower cost, More suitable for HPC	Computational cost is very high
WSRC [86]	Proactive	Rejuvenation technique	Resource availability and other overheads	Improved availability and improved overheads using variable time rejuvenation	Restricted suitability
SRFC [87]	Proactive	Software Rejuvenation	Scalability, Throughput, Reliability	Improved availability, Multiple VM rejuvenation	Limited to software rejuvenation
Fault Tolerance Scheduling (FTDG) [88]	Proactive	Pre-emptive Migration	Reliability, Response Time, and Throughput.	Minimum Response time	Restrictive applications
AFTRC [89]	Reactive	Replication and Checkpointing	Accuracy, Reliability, and Availability	Applicable for real-time applications	Stunted availability of resources

					when the load is high
BFTCloud [90]	Reactive	Replication	Scalability, Throughput, Reliability	Highly Reliable and is qualified to tolerate all byzantine faults	Low Resource Utilization
Fault-Tolerant Scheduling Mechanism for Real-Time Tasks in Virtualized Clouds (FESTAL) **[92]**	Reactive	Replication	Throughput, Reliability, Availability, Usability	Energy-efficient Resource Utilization Framework	Execution can crash if both central and backup fail concurrently
Efficient Fault Tolerance Technique (EFTT) **[107]**	Resilient	Machine Learning	Throughput, Availability, Reliability, Response time	High Response time, High Availability and Reliability, Adaptive in nature	Insufficient resource utilization

Table 2.4: Comparative analysis of different proposed fault tolerance and load balancing algorithms

Model/Technique	**Year**	**Parameters**	**Compared with**	**Outcomes**	**Advantages**	**Limitations**	**Platform/ Environment**
Honeybee Inspired-Load Balancing (HBI-LB) **[102]**	2022	Average Response Time	Round Robin (RR), Throttled (TH), and Equally Spread Current Execution Load (ESCEL).	Average Response time was optimized than compared approaches	Maintains load equilibrium	The model was not evaluated on a large task scale. No fault-tolerant parameter was considered.	CloudSim 3.0.3-based Cloud Analyst tool
Proactive Load Balance Fault	2021	Execution Time, Reliability	Adaptive Fault Tolerance in Real-Time	PLBFT achieved the highest reliability	Better Fault prediction and tolerance	An increased number of migrations was	Cloud Simulator

Tolerance (PLBFT) **[104]**			Cloud (AFTRC)	calculations than AFTRC		observed which maximized execution time	
Load balancing with fault tolerance algorithm using Replication technique **[102]**	2021	Availability, Resource Utilization	Fault Tolerance Workflow Scheduling the FTWS [91]	Efficient task scheduling along with fault-tolerance	Optimized Availability and System Performance	Poor resource utilization	Amazon EC2
Proactive Fault Tolerance Framework (PFTF) **[103]**	2017	Execution Time, Network Congestion, Cost	High-Performance Linpack (HPL), Honeybee Foraging Algorithm	Improved Execution Time and Time Delay.	Network congestion delay was reduced by 47%, Reducing the cost	No consideration of Resilient Fault Tolerance	CloudSim 3.0 tool
CLBC (Load Balancer), Deadline Based Pre-Emptive Scheduling (DBPS) **[106]**	2014	Throughput, Completion Time, Execution Time, and Computational costs	Traditional related algorithms	The computational cost was minimized	Effective Load Balancing	Not suitable for deadline-based task accomplishment	Cloudsim
FTLB **[107]**	2017	Throughput, Availability, Reliability, Response time	Ant Colony, Osmosis LB, Honeybee Foraging, Artificial Bee Colony	High Response time, High Availability, and Reliability	Adaptive nature	Slow in function	Not specified
Integrated Virtualized Failover strategy (IVFS) **[108]**	2016	Pass Rate, Task Finish Time	Virtualization and Fault Tolerance Approach (VFT) [110]	High Node Pass Rate and Less Service Task Finish Time	High fault-tolerant, both forward and backward recovery	Not suitable for the large-scale environment	

The comparative analysis of different fault tolerance-based load-balanced algorithms is presented in Table 2.5. These algorithms were proposed to distribute the workload regardless of faults across the nodes, i.e., having the capacity to handle the faults.

Table 2.5: Comparative analysis of fault-tolerant-based load-balancing algorithms

Algorithm	Year	Parameters	Outcomes	Limitations	Platform/Environment
Hybrid Load Balancing [18]	2017	Response Time	Minimizes response time and overloading situations	Lacks migration in case of failure	Cloudsim
Throttled algorithm and Equally Spread Current Execution algorithms (TA & ESCE) [64]	2017	Waiting Time, Turnaround Time, Resource Utilization	Turnaround time and wait time were reduced and resource utilization was enhanced	Lacks migration technique for performance optimization	Cloud Simulator
Starvation Threshold–based Load Balancing (STLB) [111]	2019	Response Time	Increases in resource utilization rate, Minimizing migration cost and response time	Not suitable for dependent tasks	CloudSim
Enhanced LB (TA & ESCE) [3]	2017	Response Time	Evades overloading, reduced cost, and response time	Not optimizing other QoS parameter	CloudSim
Genetic Algorithm and the gravitational emulation local search GA-GEL [112]	2015	Makespan	Reduced Makespan	Uneven Load Distribution	CloudAnalyst
LBHM [113]	2018	Response Time, Processing Time	Processing and response time were reduced.	Increases Execution Time of the VM	CloudSim 3.0.3
LB strategy based on AC [114]	2014	Response Time	Reduces response time	Not optimizing other QoS parameter	CloudAnalyst
VM-Assign Load Balancing [114]	2014	Resource Utilization	Enhances Resource utilization.	No Dynamism was considered	CloudAnalyst
Modified Optimize Response Time [68]	2021	Response Time	Response time was enhanced	Insufficient load distribution	Not Specified
weighted active monitoring load balancing (WAMLB) [115]	2018	Resource Utilization	Effective resource utilization	Not optimizing other QoS parameters.	CloudAnalyst
Priority-based modified throttled algorithm (PMTA) [116]	2016	Response Time	Balanced load distribution and minimized response time	Starvation for low-priority tasks.	CloudSim3.0 and CloudSim-based tool
Enhanced LB (TA & ESCE) [117]	2017	Response Time & Machine Cost	Uniform load distribution with less cost	No considered weakness found	CloudAnalyst

Hybrid Approach (TA & ESCE) [118]	2019	Response Time, Processing Time, Cost	Cost-effective and minimum response time	Does not include any fault-tolerant strategy	Cloud sim
Improved WRR (weighted Round Robin) [119]	2018	Processing time, and cost	Avoid starvation and cost-efficient	The current workload of VM is not studied and lacks fault handling	Eclipse framework
STLB [107]	2019	Resource Utilization and overall cost	Increased utilization rate and Dropped overall migration cost	In-appropriate for dependent workload	CloudSim
LB Strategy [120]	2014	Availability	Uniform workload distribution and high availability	Increased response time because of FCFS allocation.	CloudSim
Token-bucket rate-limiting technique [121]	2023	Availability and Scalability	Good quality of services to customers	May cause load imbalance	Zuul gateway
cuckoo optimization-based energy-reliability aware resource scheduling technique (CRUZE) [121]	2020	Cloud service availability, energy consumption	Reducing energy consumption and increasing availability	May cause load imbalance	CloudSim toolkit
Single intervention at random interval (SIRI) strategy [122]	2023	Service Availability, penalty rate	Prevents SLA violations and offers high service availability	May cause load imbalance	Amazon EC2
Backpropagation (BP)-based OnlIne hardware fault Diagnosis System has been built, named BOIDS [123]	2020	Hardware-faults (transient, intermittent, and permanent faults)	More than 97% accuracy in diagnosing hardware faults	Only hardware fault models are considered	SpecInt2000 and MiBench set to 1c1t (1 core 1 thread)
PSO, Round Robin, (ESCE) Equally Spread Current Execution, Throttled Load balancing [124]	2023	Response time, Processing time of data center	Identified the valuable relationship between VMs and tasks	Lacks the dynamism of circumstances	cloud analyst platform

2.5. Discussions and Observations

The presented survey summarizes the focus of researchers on distinct hybrid fault tolerance-related frameworks. The main emergent and developing methods of fault tolerance in a cloud environment are categorized into three different categories: Reactive Methods, Proactive Methods, and Resilient Methods. The survey was conducted on two main hybrid fault-tolerant categories, i.e., scheduling with fault tolerance and load balancing with fault tolerance. On surveying, several observations were gathered and listed below.

2.5.1. Statistics of Hybrid Survey of Scheduling and Fault Tolerance Algorithms

While dealing with hybrid frameworks of scheduling and fault tolerance, researchers have focused on all three fault tolerance approaches, i.e., Reactive, Proactive, and Resilient. However, it is observed that more emphasis is on Proactive and less on Resilient ones. The related statistics of these approaches are depicted in Figure 2.3.

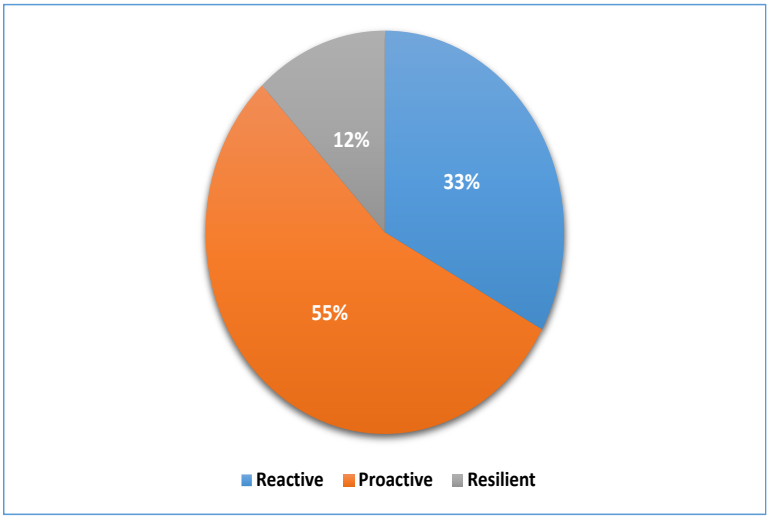


Figure 2.3. Showing Fault Tolerance Approaches Targeted by Researchers

Moreover, different techniques such as Replication, Migration, and Rejuvenation have also been employed while dealing with this hybrid framework. Replication techniques are mainly used for reactive approaches. On the other hand, Migration and Rejuvenation techniques are utilized for proactive approaches. It is also observed from the literature that replication and migration techniques were more frequently used to address the faults in the cloud. Moreover, self-healing and checkpoint restart techniques are used by the SHelp framework. The statistics of different approaches employed for Reactive, Proactive, and Resilient strategies in this hybrid framework are depicted in Figure 2.4.

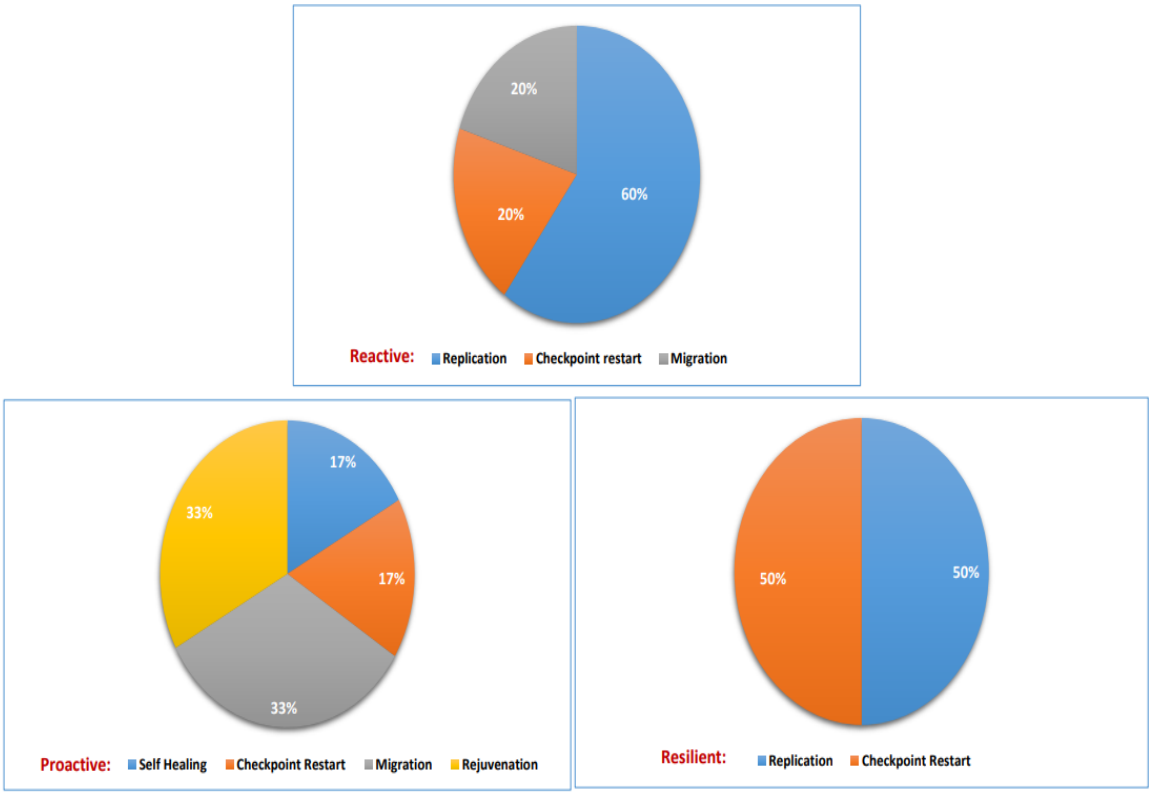


Figure 2.4. Showing Category-wise Percentage of Different Techniques used in Fault Tolerance

It is also noticed from the presented survey that different types of faults have been handled by using hybrid fault-tolerant scheduling and load-balancing frameworks. Moreover, it was observed that software faults, hardware faults, parametric faults, and crashes were resolved using a proactive approach. The reactive approach addressed configuration faults, parametric faults, byzantine faults, participant faults, and host failures. Likewise, resilient approaches are utilized to manage general faults. Additionally, the overall statistics of different faults handled by considered hybrid frameworks are depicted in Figure 2.5.

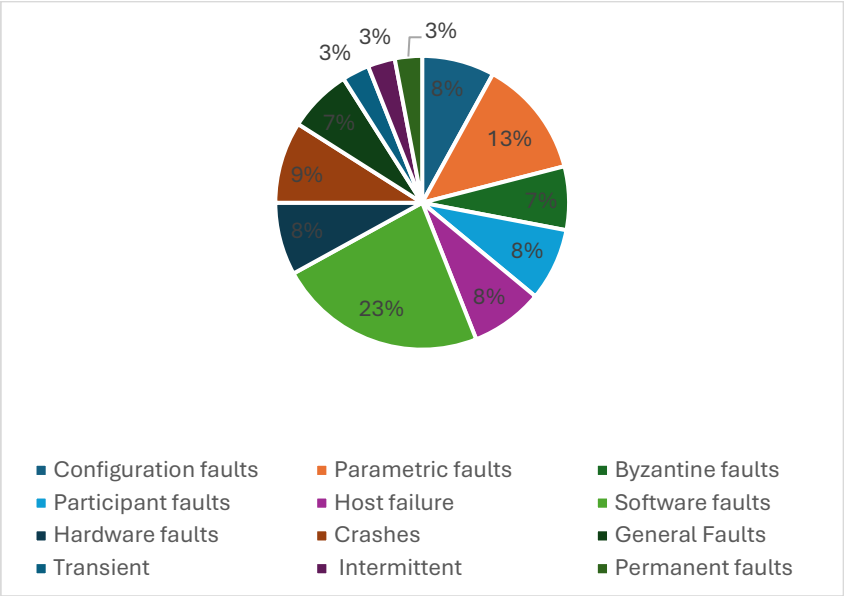


Figure 2.5. Showing the Percentage of Optimized Parameters in Surveyed Scheduling and Fault Tolerance

The statistics of the fault models focused in the surveyed articles show that researchers are more motivated towards software faults but the transient, intermittent, and permanent faults are found to be less in the eyes of the researchers. For several strong reasons, addressing these kinds of faults is essential in distributed systems/applications. First, proactive steps to guarantee system resilience are required due to the unpredictable nature of transient faults, which are brief interruptions in system performance. To reduce downtime and provide a consistent user experience, organizations must recognize and address transient issues. Second, a major threat to system reliability is intermittent failures, which are defined by irregular disruptions that might happen at any time. To avoid flowing failures and guarantee the stability of necessary executions to preserve the system's overall integrity, intermittent faults must be effectively managed. Furthermore, we cannot exaggerate the seriousness of permanent faults. These enduring problems may cause the system to deteriorate over time, impacting system operation and SLAs. Therefore, resolving permanent faults is essential for maintaining the system's lifespan and functionality while ignoring them might cause irrevocable harm and compromise the global sustainability of the system. Finally, the maintenance of system continuity, robustness, and reliability is the primary reason for managing the discussed hardware failures. In the end, proactive fault management techniques contribute to uninterrupted system/application performance during unexpected obstacles by protecting the integrity of crucial operations and improving SLAs and thereby user experience and satisfaction.

2.5.2. Statistics of Hybrid Survey of Load Balancing and Fault Tolerance Algorithms

It is also perceived in this survey that researchers have focused on the optimization of various parameters simultaneously along with fault tolerance. The response time was considered and optimized more frequently as compared to other QoS parameters. And least consideration is on task waiting time and the computational cost. Based on this survey, the statistics of various optimized parameters are presented in Figure 2.6. Besides, the considered frameworks include both dynamic and static environments, and the researchers are more motivated toward dynamism than static algorithms. Figure 2.7. depicts the statistics of the surveyed models that support dynamism.

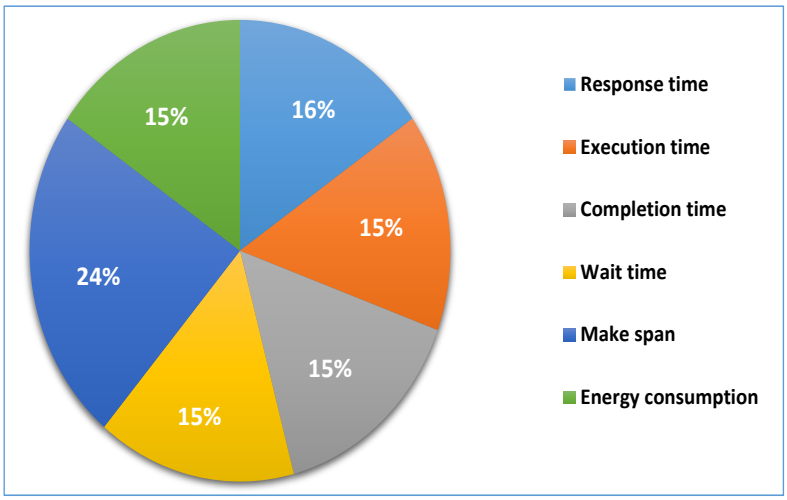


Figure 2.6. Showing the percentage of Optimized Parameters in Surveyed Load Balancing and Fault Tolerance

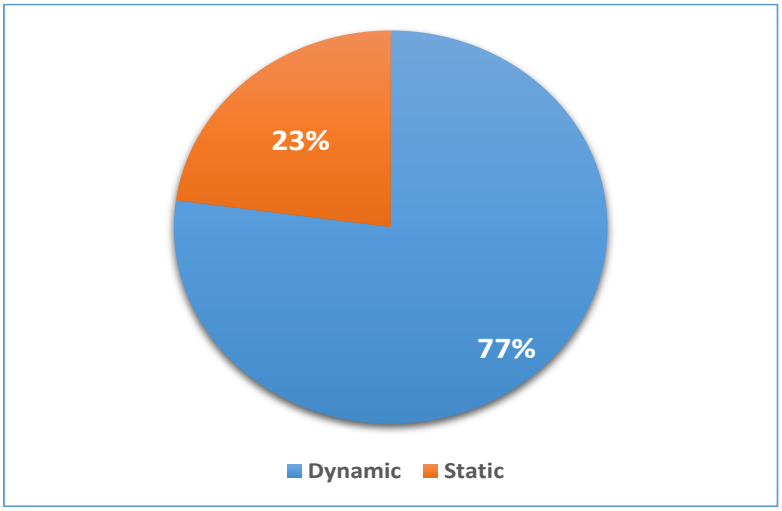


Figure 2.7. Showing the Percentage of Dynamism in Surveyed Hybrid Load Balancing and Fault Tolerance Frameworks

The analysis was carried out for the parameter optimization of the reliable cloud. Figure 2.8 presents the degree of optimization in metrics of scheduling with fault tolerance, scheduling with load balancing, fault tolerance, load balancing, and scheduling. Additionally,

parameter optimization analysis of various fault-tolerant approaches from the literature was also conducted and presented in Figure 2.9.

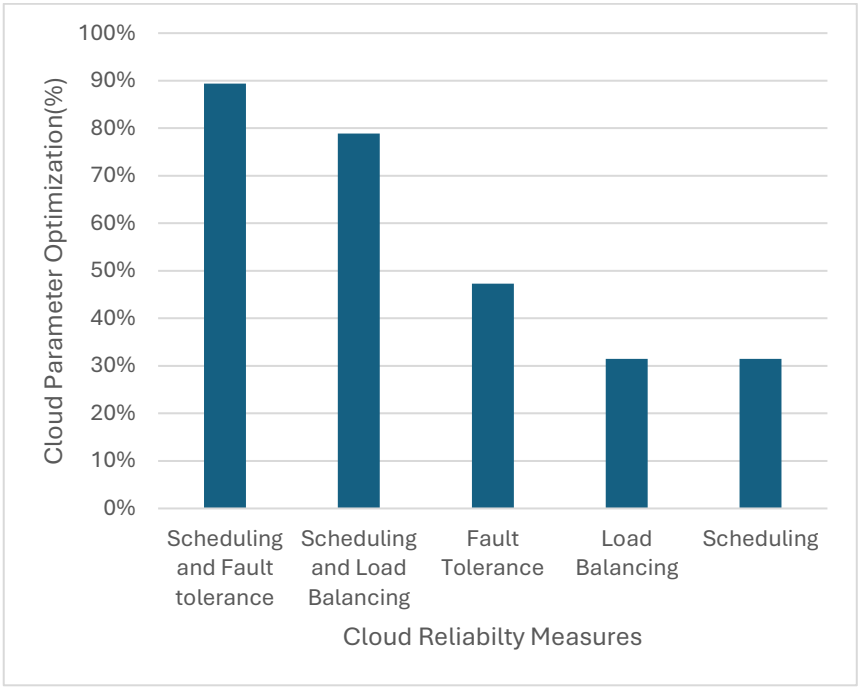


Figure 2.8. Showing the Analysis of Parameter Optimizations for Different Cloud Reliability Measures

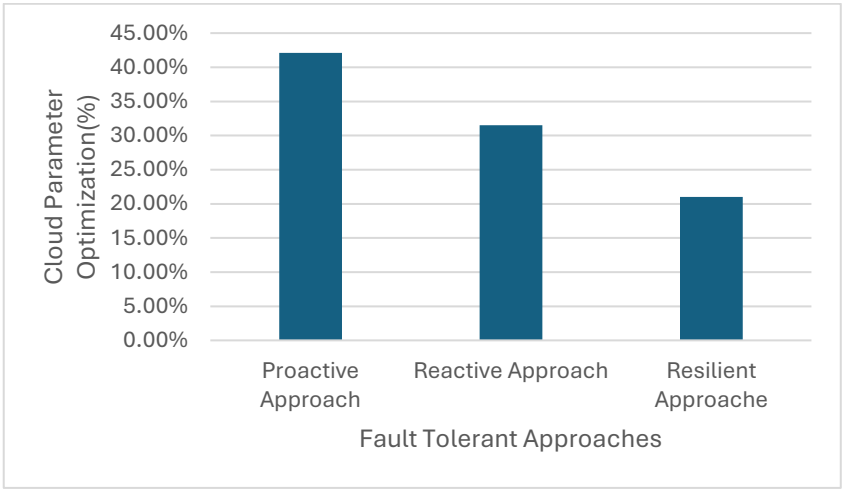


Figure 2.9. Showing the Percentage of Parameter Optimizations for Different Fault Tolerant Approaches

Finally, the observations regarding the platform or environment used for simulation in the presented surveys are statistically presented in Figure 2.10.

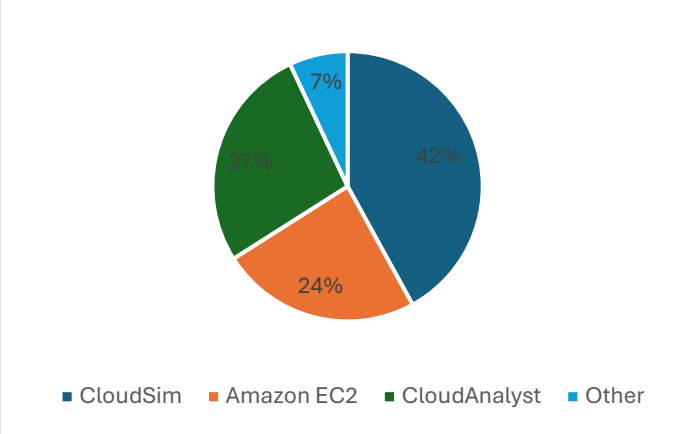


Figure 2.10. Showing the Percentage of Tools used for Simulation by the Researchers

2.6. Summary in Context

This chapter extensively reviews distinctive approaches and methodologies employed for enhancing the cloud environment to find the research gap. After due estimation and assessment, it is studied that several areas of research could be tracked to improve the efficiency of cloud approaches and to improve the service performance of cloud computing.

The consideration of the dynamic character of the cloud motivated us to propose the hybrid scheduling model integrated with fault tolerance, and load balancing for cloud setup. Although there are many scheduling algorithms available in the literature, the researchers are highly attracted and conservative towards developing various scheduling, fault tolerance, and load balancing algorithms. However, it is observed that these can enhance and optimize the cloud system up to a certain degree of scenarios. Moreover, the integration of both fault tolerance and load balancing in dynamic scheduling algorithms to optimize the QoS parameters has been overlooked in the literature. The integration of load balancing models with fault tolerance is a peak demand of time. Because the fault tolerance mechanisms may often reorder the prior scheduling VM assignment to fit and strong VMs in the occasion of a failure or fault, leading to uneven VM reassignment. This uneven VM reassignment becomes the cause of QoS degradation even if the prior Scheduling algorithm is highly optimized.

Besides, there are various demanding reasons why the integration of load balancing is important for optimal overall system performance. Some of the mounted demands are listed below:

- *Cloud settings frequently consist of varied and diverse VMs with random performance characteristics. Task scheduling within these heterogeneous VMs to achieve optimal performance can be complex.*

- *Scheduling always suffers from faults because of failures such as hardware and software failures, resource conflicts, data distortion, human errors, etc. This could lead to the premature termination of the corresponding tasks thereby disrupting the continuity of cloud services. Handling these disruptions in the cloud is necessary and critical to ensure the reliability and efficiency of task scheduling in cloud computing.*
- *Implementing fault tolerance can introduce the overheads associated with it. However, using load balancing with fault tolerance can reduce operational burdens and other complexities.*
- *Various bottlenecks and other congestion can be created on healthy VMs in fault-tolerant systems. This can impact overall system performance. This can be eased by intelligently distributing load flow across VM post to fault tolerance.*
- *Fault tolerance often necessitates redundant resources to grip failover circumstances, which can lead to resource overprovisioning and over-cost. Load balancing can be helpful in such cases as the integration of load balancing can dynamically adjust the load over VMs thereby dropping the requirement of additional capacities.*

Fault tolerance integrated with load balancing can help organizations overcome these limitations and create stronger, more effectual, and mountable distributed systems that can adapt to the altered loads because of fault tolerance. Therefore, we converge towards developing the dynamic scheduling model which not only handles faults but also handles the uneven VM reassignments by integrating effective load balancing constraints post to fault tolerance.

Chapter 3
Ranked Task Scheduling and Reservation in Fault Tolerance for Cloud Computing

A cloud computing platform has higher failure rates because of its highly dynamic nature and running of concurrent applications. However, the outcomes of running concurrent applications won't be accurate without VM synchronization. The issue of coordination between many VMs is their synchronization in working is rarely considered by existing solutions. Moreover, fault tolerance constitutes one of the most crucial components for cloud computing architecture to ensure high reliability. In this Chapter, Reserved Fault Tolerance and Ranked Task Scheduling (RFRTS) is proposed. Initially, the proposed ranked-based scheduling approach is used for task allocation, and later the idea of a reservation-based reactive fault tolerance method is suggested for a cloud system. To achieve the highest level of cloud computing infrastructure reliability, the suggested technique considers CPU faults and the VM reservation will ensure the assignment of an alternative VM to the affected task. The proposed fault-tolerant approach has been compared with three existing reliable fault-tolerant approaches namely multi-objective scheduling algorithm with Fuzzy Resource utilization [125] (FR-MOS), Cost-effective Workflow Scheduling Algorithm [126] (CWS), and Fault-tolerant Cost-effective Workflow Scheduling Algorithm [127] (FCWS) based on reliability. The outcomes unequivocally show that our suggested RFRTS algorithm surpasses the current FR-MOS, CWS, and FCWS considering reliability in all the states.

3.1. Proposed Model

This section presents an effective allocation to map the incoming tasks and available VMs. After the mapping of task and VM sets, we propose a framework that would estimate the reservation window based on the size of the task and the VM's capacities. This will move the affected tasks from unstable VMs to reserved and trustworthy ones based on the reservation window. The suggested model's primary goal is to increase system reliability through assured task execution using advance resource reservations.

3.1.1. The System Architecture

The Proposed System Architecture is comprised of three main layers and is presented in Figure. 3.1.

- Application Layer: The main interface for communication is provided by the Application layer. Moreover, requests for VMs are also generated by the user in this

layer. Application Layer consists of Users and Requests.

- Middleware: Middleware is the main component of this architecture and is mainly responsible for Task allocation and VM reservation.

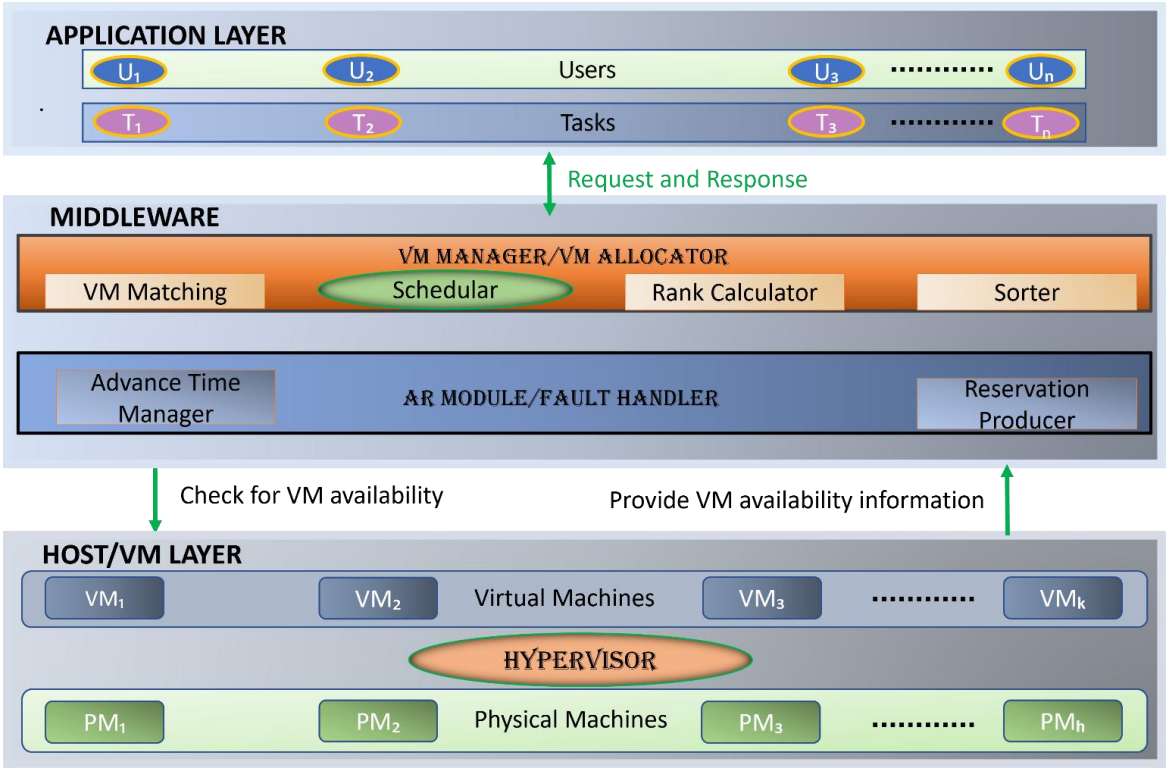


Figure. 3.1: System Architecture of RFRTS

The user tasks first arrive at Middleware, where they are assigned the rank value by the Rank Calculator. Later the tasks are sorted based on the decreased "r" value by the Sorter component. Here, the tasks wait for the VM Matching module which controls, directs, and observes the available VMs. Eventually, the Scheduler module handles user tasks to determine scheduling decisions.

The Fault Handler is activated upon detecting a failure in any of the VMs and generates an alternative VM for the affected task while utilizing the Reserved VMs.

Advance Time Manager: is responsible for reservation-related details such as AR slot, VM allotment, fresh reservations, cancellation of requests, etc.

Reservation Producer: This component checks the schedule for user requirements and the state of the VM. The required reservation is provided to the user if the user's requirement matches the schedule for the AR slot produced by the Time manager.

- VM Layer/Host Layer: This layer contains different VMs that are used by users to execute their tasks.

3.1.2. Problem Formulation

The VMs are assigned in set V={v_1, v_2, v_3… v_k}. while the tasks being taken as set T= {t_1, t_2, t_3… t_m}. Every VM has VM capacity, i.e., (C(v_k)), and every incoming task has task length, i.e., (L(TM)).

Further, the various assumptions of the proposed allocation strategy are:

- The lower task heterogeneity metric is used by the model.
- Task sizes range from one to one hundred million instructions.
- The benchmark for low machine heterogeneity is used by the model.
- The speed of the machine varies between one to ten Million Instructions Per Second (MIPS).

3.1.2.1. Ranked Task Scheduling

Every task (t_m) has its task id (t_id) which is assigned to the task on an FCFS (First Come First Serve) basis. That means the task having a smaller t_id is waiting for a longer time. Initially, the incoming tasks have been ranked based on r (response rank value). The response rank value (r) is calculated for every incoming task as below:

$$r = \frac{-t\_id + \mathrm{PT}}{\mathrm{PT}} \tag{1}$$

Where Burst time is calculated as:

$$PT = \frac{L(t_m)}{C(v_k)}$$

Further, the ranked tasks are taken as separate task sets, i.e., T_r. The tasks are distributed in order of rank value to the corresponding VMs. The r value for the task is calculated by adding the t_id and Processing Time of the task and dividing the obtained value by the processing time of the task. The ranked allocation considers both the wait time and processing time of the task. It also minimizes the wait time for large tasks and simultaneously encourages the small tasks to get a higher rank thereby giving an optimized QoS than that of Shortest Job First allocation.

The task allocation problem is mathematically represented as a mapping of each incoming task to the VM as shown below:

$$\mathcal{M} : \mathrm{T} \times \mathrm{r} \rightarrow \mathrm{V}$$

The bipartite graph between the task set and the VM set may be used to describe the suggested ranking approach. The set T_r contains the ranked tasks in descending order of r-

value rather than the t_id, hence the tasks in T_r may be arranged randomly concerning to t_id as indicated in Figure. 3.2.

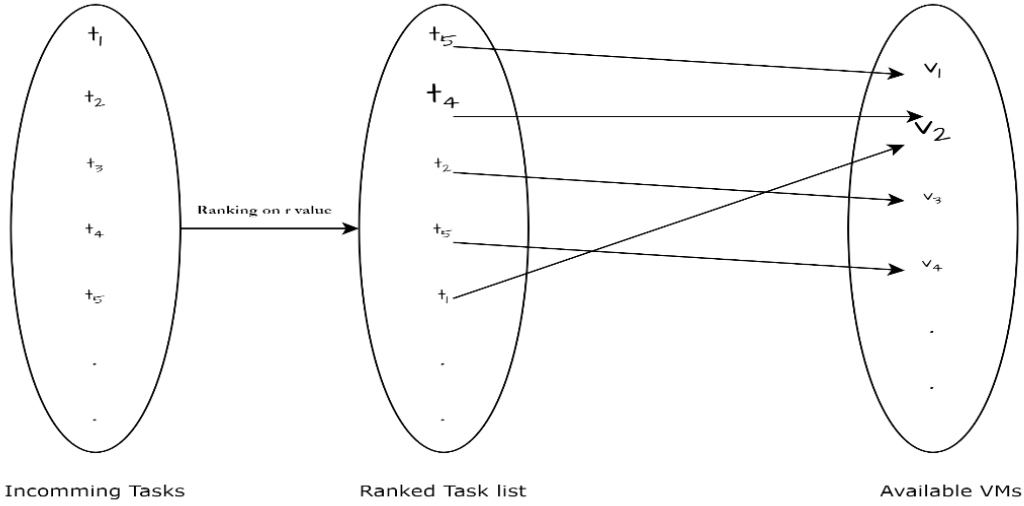


Figure. 3.2: Ranked Task Mapping

Proposed Pseudocode for the Ranked Task Scheduling

Using the proposed ranked scheduling approach, the system effectively maps the incoming tasks with appropriate VMs.

i.e., ***Input****: n(T), L(t_n), n(V), C(v_k), RT(v_k), RM[ ]*

Output *(Mapping (t_n, v_k), Reliability)*

1. Above all, the RT of all VMs is taken as zero indicating the VM has no load history.
2.
3. The task lengths range between 1 MI to 100 MI, and the machine capacity runs from 1 MIPS to 10 MIPS, according to the model's low task and low machine heterogeneity assumptions.
4. Rank tasks on the basis of Task_id, PT //task ranking algorithms.
5. *Do*
   Map the ranked tasks to the VMs. //allocation algorithm
   While $\forall v_k s$, *(RT(v_k) = 0))*
   Map task to VM having least RT.
6. Determine the ST and FT of the task.
7. Adjust the revised RT for v_ks after each allocation.

3.1.2.2. Reservation in Fault tolerance

Besides, the model also includes fault tolerance by utilizing resource reservation techniques, where the VM is reserved for the task for a specific pre-estimated window known as a reservation window (R).

If the advance reservation strategy is not employed, the task t_u may fail to execute on VM_f, i.e., the failed VM, or if the VM leaves the system for a certain time will result in the suffering of the corresponding task. To handle this situation, p failed VMs are defined as $VM_f = \{v_f : v_f \in V \ \& \ \sigma(VM_f) = p\}$ and q corresponding affected tasks are defined which were executing on these failed VMs. Now, these failed tasks need to be reallocated to some other available suitable VMs. The set of failed tasks is defined as $T_f = \{t_f : t_f \in T, \sigma(T_f) = q \ \& \ q <= N\}$. On reallocating, all the failed tasks T_f are migrated from VM_f to VM_j such that $VM_j \notin VM_f$.

Proposed Reservation based Fault tolerant algorithm

The proposed algorithm wins fault tolerance in case any of the VMs fail to execute the task at any point in time. The VMs are reserved for the computed time slot to ensure task execution. The reservations in the algorithm are done according to the following Pseudocode:

1 Initialize the Input parameters i.e., the task number, size of the task, number of VMs, the capacity of VM, etc.

2 Calculate the AR slot.

3 The initialization of the RM matrix with the tasks, Start time, and Finish time, State flag, and the computed reservation window.

4 Reserve VMs for a predicted timeslot to continue the processing of the task in case of VM failure.

5 Status=1 i.e., VMs are reserved for the calculated AR slot.

Repeat steps 2 to 5 for all t_is.

Additionally, $|T_u|$ = Fault (%age) * |T|

$$\text{Reliability} = \frac{|T| - |T_p|}{|T|} \tag{6}$$

3.2. Results and Observations

The proposed model was evaluated on reliability by comparing it with other reliable existing models namely, FCWS, FR-MOS, and CWS. We selected five distinct states of task numbers with varying lengths for the simulation we created: Small(S)[n = 50], Medium(M)[n=100], Medium large(M-L)[n=200], Large(L) [n = 400], Extra-large (E-L) [n=600].

Figure 3.3: Depiction of Reliability in Five Considered Task States

It can be seen from the depicted graph in Figure. 3.3 that the proposed model shows higher reliability than all the considered models in all states afterward FCWS performs better. Furthermore, it is evident from the figure that as the amount of tasks increases, the reliability of the considered approach decreases. However, as can be seen in the Improvement Percentage Table (Table 3.1), the suggested model exhibits a rise in the percentage of reliability improvement as the task count increases. This is because the suggested Model can efficiently handle various invoicing fault scenarios as the model is reserving the VMs for the dedicated window.

Table 3.1: Comparative analysis of improvements in the proposed RFRTS

FCWS	FR-MOS	CWS	Five states
0.30%	2.25%	2.04%	**S**
1.32%	2.36%	1.84%	**M**
1.53%	2.37%	1.84%	**M-L**
1.65%	2.18%	1.97%	**L**
1.26%	2.45%	2.56%	**E-L**

In S, the minimum improvement by the model was seen to be 0.30% while the maximum improvement was seen to be at 2.25%. In M, the minimum improvement by the model was seen to be 1.32% while the maximum improvement was seen to be 2.36%. In M-L, the minimum improvement by the model was seen to be 1.53% while the maximum improvement was seen to be 2.37%. In L, the minimum improvement by the model was seen to be 1.65% while the maximum improvement was seen to be 2.18%. In E-L, the

minimum improvement by the model was seen to be 1.26% while the maximum improvement was seen to be 2.45%.

3.3. Summary in Context

The research suggests a method for task ranking by considering task lengths and task wait times. Besides, the algorithm implies an allocation strategy based on the determined rank value. Also, we provide an idea of reservation for fault-tolerance in which VM reservations are made based on a pre-calculated reservation AR slot. The reservations are deeply explained in the later chapters. The focus of the proposed ranked task scheduling in the chapter is on makespan, flowtime, and average resource utilization. However, system reliability has also been focused on and enhanced by the proposed reservation. The major drawback of the suggested allocation is that it does not consider any load-balancing strategy. Hence the load may be inadequately distributed. However, it will unquestionably improve the task response times by focusing on the wait time of the tasks. The study's plans demand to consider response time for working with the suggested ranked scheduling technique. Additionally, the load balancing strategy will be accompanied to consider resource utilization as well.

Chapter 4
Towards Fault Overheads in Cloud: Next Gen VM Management using Hybrid Approach (HFSLM)

The major goal of the computational system is to effectively allocate resources escorted with fault tolerance to ensure the job execution is on time. The primary study concern is also regarding the mechanism for even distribution of load among virtual machines for further system improvements. Addressing all these issues simultaneously is a good need of time. Several methods have been developed and proposed in the literature to overcome the aforementioned research issues. However, very few researchers have included a significant contribution to addressing all these issues simultaneously with optimized QoS parameters. In this chapter, a novel Hybrid Fault-tolerant Scheduling and Load balancing Model (HFSLM) has been proposed to optimize the makespan and average resource utilization. Moreover, the model also provides solutions for several crucial concerns for a cloud system including VM failure and VM/task heterogeneity by reserving neighboring VMs in the event of failure. Furthermore, the model is escorted by a load-balancing algorithm for further optimization of the considered QoS parameters. HFSLM is evaluated by comparing it with FTHRM, MAX MIN, MINMIN, OLB, ELISA, and MELISA on both small and large task scales. The evaluation results show that the proposed HFSLM outperforms the compared approaches in all the considered cases.

The proposed hybrid model focuses on three issues i.e., efficient scheduling, fault tolerance, and load balancing. The model initially schedules the arriving tasks and maps them to the most suitable virtual machine thereby focusing on the optimized makespan and efficient utilization of virtual machines. Moreover, the proposed model adapts the system to respond to the faults by using the neighboring-based advance reservation technique. The neighboring-based advance reservation technique is the technique where the reservation slot is estimated in advance and the neighboring VM is reserved as an alternative VM for the affected task to guarantee the execution of the task till completion. In this case, the neighboring VM with the least history of the load (Ready Time) is preferred to be selected as an alternative VM. Furthermore, the model also escorts the proposed fault tolerance and scheduling algorithms with a load-balancing strategy to make further optimizations in various QoS parameters. The proposed model was evaluated for parameters like makespan and average resource utilization

by comparing it with FTHRM [128], MAX MIN [129], MINMIN [130], and OLB [50] on a low task scale (less than 1000 tasks). The evaluation has been done by adjusting the

number of tasks, size of tasks, number of VMs, and capacities of VMs in four different heterogeneity benchmarks given by Braun [35] i.e., low task-low machine heterogeneity, low task-high machine heterogeneity, high task-low machine heterogeneity, and high task-high machine heterogeneity. Besides, the proposed HFSLM was compared with ELISA [131] and MELISA [46] on very high task scales (greater than 10,000 tasks) and was evaluated using an average makespan, and the resource utilization was taken into consideration for minimum, average, and maximum cases.

4.1. Main Focus and Contribution

The consideration of the dynamic character of the cloud motivated us to propose the hybrid scheduling model integrated with fault tolerance, and load balancing for cloud setup. Although there are many scheduling algorithms available in the literature, the researchers are highly attracted and conservative towards developing various scheduling, fault tolerance, and load balancing algorithms. However, it is observed that there are very few dynamic scheduling algorithms that integrate both fault tolerance and load balancing models to optimize the QoS parameters. The integration of load balancing models with fault tolerance is a peak demand of time. Because the fault tolerance mechanisms may often reorder the prior scheduling VM assignment to fit and strong VMs in the occasion of a failure or fault, leading to uneven VM reassignment. This uneven VM reassignment becomes the cause of QoS degradation even if the prior Scheduling algorithm is highly optimized. Besides, there are various demanding reasons why the integration of load balancing is important for optimal overall system performance. Some of the mounted demands are listed below:

- *Fault tolerance often necessitates redundant resources to grip failover circumstances, which can lead to resource overprovisioning and over-cost. Load balancing can be helpful in such cases as the integration of load balancing can dynamically adjust the load over VMs thereby dropping the requirement of additional capacities.*
- *Implementing fault tolerance can introduce the overheads associated with it. However, using load balancing with fault tolerance can reduce operational burdens and other complexities.*
- *Various bottlenecks and other congestion can be created on healthy VMs in fault-tolerant systems. This can impact overall system performance. This can be eased by intelligently distributing load flow across VM post to fault tolerance.*

- *Similarly, other factors should be considered in fault-tolerant systems such as augmented latency, partial scalability, suboptimal resource utilization, etc.*

Fault tolerance integrated with load balancing can help organizations overcome these limitations and create stronger, more effectual, and mountable distributed systems that can adapt to the altered loads because of fault tolerance. Therefore, we converge towards developing the dynamic scheduling model in this work which not only handles faults but also handles the uneven VM reassignments by integrating effective load balancing constraints post to fault tolerance.

The scheduling in the proposed model has been done by initially rearranging both arriving tasks and available VMs. The newly incoming tasks and freshly installed or deleted VMs are also taken into consideration while performing the recommended scheduling. This consideration makes it the most suitable scheduling for fully dynamic computing infrastructures. Additionally, the scheduler offers efficient allocation concerning the user's needs at selected timeslots by using a reservation. Reservation is the technique where the VMs are reserved for the task till it completes its execution thereby resulting in the assurance of task completion. However, if the VMs are not reserved, they might fail permanently or stop working at any time which may result in the termination or interruption of the corresponding task. Therefore, the model delivers the system the fault tolerance that it needs to manage runtime system errors after conducting effective scheduling. Apart from fault tolerance, the model reallocates the load to reduce the imbalance caused by fault tolerance. The evaluations are conducted by assessing the proposed model with existing similar models such as MAX-MIN, MIN-MIN, OLB, FTHRM, ELISA, AND MELISA. The MAX-MIN algorithm was found optimal for resource allocation. The makespan and utilization obtained by MAX-MIN on low task heterogeneity were also found efficient. However, for high task heterogeneity, MAX-MIN was not found significant. Furthermore, the QoS parameters obtained in MIN-MIN were not optimized in varying task and machine heterogeneities. Apart from this, these are allocation algorithms and do not support any fault handling or load-balancing procedure. The most recent FTHRM model for fault tolerance was using advance reservation. However, this model did focus on uniform load distribution. Moreover, this has not migrated the tasks from the faulty VM to the reserved VM. For large sizes, ELISA and MELISA were shown to be the best tests; however, for small scales, these models were insignificant. Additionally, these models are load-balancing models and do not support any fault tolerance mechanism. After analyzing the related literature, it was determined that the models implemented so far, particularly the hybrid models, needed to

be improved for better QoS parameters. This is where we were motivated to propose a neighboring-based reservation technique for fault tolerance for the real-time cloud. In this work, we suggested the HFSLM model with effective fault tolerance and load-balancing strategies for better outcomes. Table 4.1 shows a comparative analysis of all the considered issues and parameters between the proposed HFSLM and related models.

Table 4.1: Comparative analysis of existing models and the proposed model

Technique/Parameters	**FTHRM**	**MAX-MIN**	**MIN-MIN**	**OLB**	**ELISA**	**MELISA**	**Proposed HFSLM**
Task scheduling	√	√	√	√	√	√	√
Fault tolerance	√	×	×	×	×	×	√
Load balancing	×	×	×	×	√	√	√
Task/VM Heterogeinity	×	×	×	×	×	×	√
Makespan	√	√	√	×	√	√	√
Resource Utilization	√	√	√	√	√	√	√
Dynamic scheme for inserting and deleting task/VM	×	×	×	×	×	×	√

4.2. Proposed Work

This section illustrates the demonstration of the Hybrid Fault-tolerant Scheduling and Load Balancing Model for the considered cloud environment. The proposed work is presented in four subsections. Initially, the System Model explains the System architecture of the proposed model. The Problem Formulation provides the mathematical explanation for the proposed HFSLM. The Proposed Algorithm and Pseudocode present the HFSLM in a semi-formal form as an algorithm and pseudocode. Later, in the Motivational Illustrative Example, the proposed model's operation is demonstrated as an example. The objective function of HFSLM is to minimize the makespan while maximizing the average resource utilization. Furthermore, the following list contains the notions that were utilized in the illustration and demonstration.

4.2.1. System Architecture

This work considers the heterogeneous system with respect to both tasks and VMs. The group of VMs has varying processing speeds and so do the sizes of the incoming tasks. The system architecture of the proposed HFSLM is shown in Figure. 4.1.

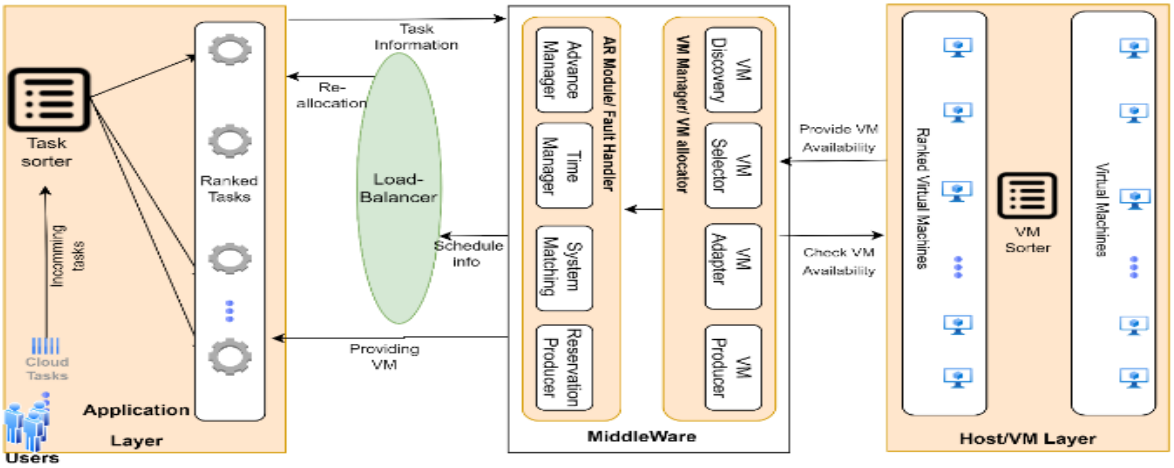


Figure 4.1: The Proposed System Architecture

There are three basic levels in the HFSLM system architecture. Together, the Application Layer, Middleware, and Host/VM Layer complete the model's functionality. The application layer receives the user tasks (incoming), and the Task Sorter sorts them in the ascending order of their size as they arrive. On the other hand, the VMs that are accessible are in the Host/VM layer, the VM sorter sorts the VMs in the ascending order of their speed. The middleware handles the primary allocation and fault tolerance. The middleware is made up of two primary parts: the VM allocator, which creates the schedule for receiving task information, and the Failure Handler, which reacts when any VM has a fault. Both works effectively together to schedule incoming tasks and reserve VMs. The different components in the VM allocator work in coordination and oversee the incoming task information for selecting the appropriate VM for accomplishing tasks. The task of identifying every accessible VM in the VM layer falls within the purview of the VM Discovery component. Once the available VMs are discovered, the suitable VM for the task is selected by the VM selector. After identifying the most suitable VM for the task, the VM Producer allocates the specified VM to the task. Further, the VM Allocator communicates the schedule generated by it to the AR Module and the Load Balancer. In response, the AR Module activates its components and generates the reservation in case of faults and breakdowns. The Time Manager component of the AR Module forecasts the AR Slot for the affected task and reserves the suitable VM for the computed AR slot in advance. After calculating AR Slots, the System Matching verifies if the task and VM are a good fit for generating reservations, and the Reservation Producer commits the produced reservation for the estimated AR Slot in the event of a fault. Additionally, the load balancer analyzes the generated schedule and plays the key role in uniformly distributing the load among VMs by identifying the

maximum overloaded and minimum underloaded VM and reallocating the tasks between them.

4.2.2. Problem Formulation

Initially, the set of incoming tasks represented by $T=\{t_1, t_2, \ldots t_n\}$ and the set of Virtual Machines signified by $V=\{ v_1, v_2, \ldots v_m\}$ has been taken over the proposed HFSLM. Every task (t_i) is executed on the allocated *VM* till the execution of the task is completed. The task is pre-empted, in case the assigned VM fails or becomes unavailable at any point in time.

The execution of an affected task will start from the beginning on an alternative VM assigned to it. Further, each task has its parameters like *t_id* and *t_size*. However, each VM has its parameters like *V_id,* and S. Apart from this, a few characteristics considered for VMs are:

- *The model considers "m" VMs for the mapping of "n" tasks.*
- *S of VM is taken in MIPS (Million Instructions Per Second)*
- *The available VMs do not apply to other applications.*
- *Each VM has its RT_j associated with it. RT_j is the time experienced to execute the load history on the VM.*

The problem modeled here is to generate a fault-tolerant allocation schedule in a dynamic environment like the cloud in a way that will optimize makespan and increase the average VM utilization. Mathematically, the problem can be viewed as an effective mapping (*M*) (eq. 1) between two sets i.e., set *T* and set *V*, which will optimize the given parameters.

$$M : T \rightarrow V \qquad (1)$$

The mapping between tasks and VMs graphically can also be treated as a bipartite graph as shown in Figure 4.2.

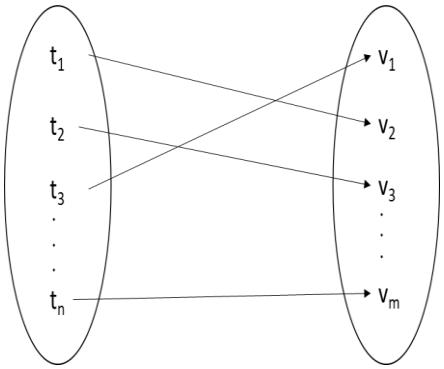


Figure. 4.2: Mapping between Tasks and VMs

4.2.2.1. Task to VM Mapping Model

The main difficulties in mapping between T and V are the dynamism of the system and the limited number of available VMs. Achieving fault tolerance in such a dynamic system is a challenging task. This section explains the detailed methodology to deal with the modeled problem. Initially, in the first algorithm, the tasks are allocated to the available VMs. The allocation process first sorts the incoming tasks and VMs based on *task_size* and VM speed respectively. Thereafter maps the sorted task set to the sorted VM set. After sorting incoming tasks and VMs, the VM is assigned to the tasks in the order until the ready time of any one of the available VMs is zero. Once the ready time of all VMs becomes greater than zero that means currently all the VMs have some load history. After this point, the allocation of further arriving tasks will be done to the VM having the least ready time. Doing this will again minimize its response time. The proposed allocation strategy handles the dynamically arriving tasks by employing a Neighbouring insertion policy. The newly arriving task will be inserted based on the arriving task size. i.e., the immediate greater and immediate lesser task (neighboring tasks) than the arriving task is identified, and the newly arriving task is allocated to that task's VM which has less ready time as shown in Figure 4.3. This insertion policy of tasks will again play a critical role in allocating the most suitable VM for the dynamically arriving task. Similarly, the newly added VMs are inserted in their correct position by employing the same insertion policy as shown in Figure 4.4. This insertion policy of incoming tasks and VMs in their respective positions allows the system to handle tasks and VMs runtime.

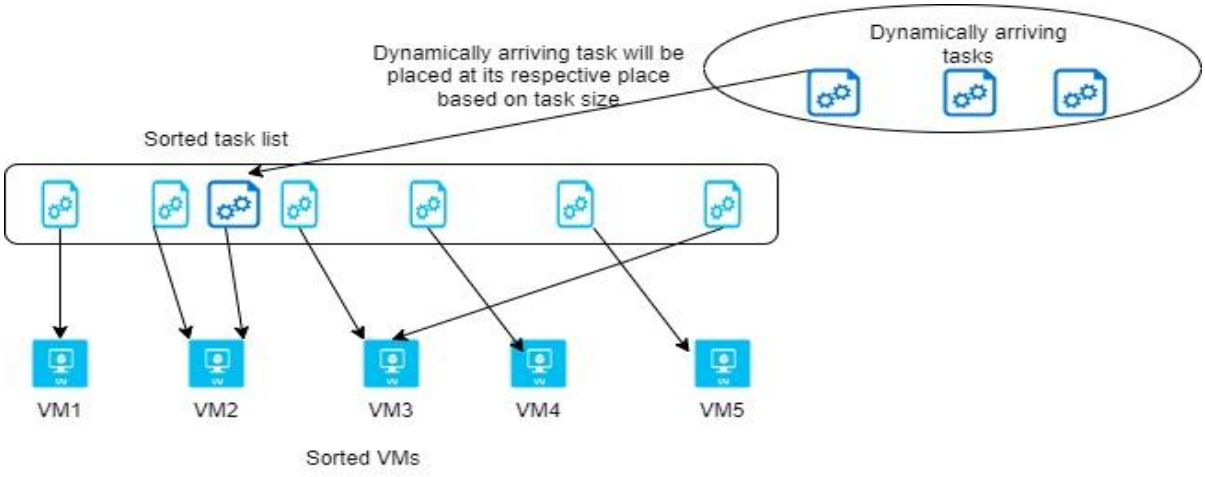


Figure 4.3: Allocation of Dynamically Arriving Tasks

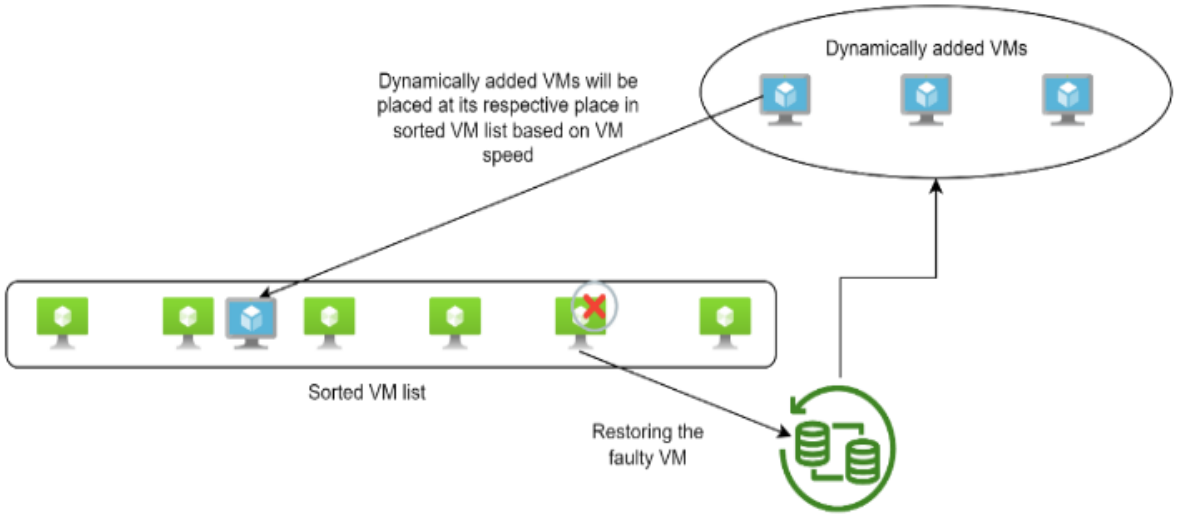


Figure 4.4: Adding and Deleting VMs in/from the System Dynamically

Because of the sorting and neighboring approach, every task will be allocated to the most suitable VM. The allocation of VMs to tasks is done in this order until the ready time of any one of the available VMs is zero. Once the ready time of all VMs becomes greater than zero that means currently all the VMs have some load history. After this point, the allocation of further arriving tasks will be done to the VM having the least ready time. Doing this will again minimize its response time. The task allocation algorithm is presented in Algorithm 1.

Algorithm 1: Task to VM Mapping

Task Sorting:

```
def task_sort(incoming_tasks):
    // incoming_tasks is a list of tuples (task_id, task_size)
    incoming_tasks.sort(key=lambda x: x[1]) // Sort based on task_size in ascending order
    return incoming_tasks
```

VM Sorting:

```
def vm_sort(available_vms):
    // available_vms is a list of tuples (VM_id, VM_Speed)
    available_vms.sort(key=lambda x: x[1]) # Sort based on Speed in ascending order
    return available_vms
```

Task Mapping:

```
def task_mapping(incoming_tasks, available_vms):
    mapped_tasks = []  // List to store allocated tasks
```

```
  for task in incoming_tasks:
    if task_status[task] == 0:
      for vm in available_vms):
        if ready_time[vm] == 0:
          map_task_to_vm(task, vm)  # Map task to VM
          mapped_tasks.append(task)
          break       // Move to the next task
# Map remaining tasks to VMs with the least ready time
  for task in incoming_tasks:
    if the task is not in mapped_tasks:
      min_ready_time = min(sorted_vms, key=lambda vm: ready_time[vm])
      map_task_to_vm(task, min_ready_time)
# Update mapped_tasks
      mapped_tasks.append(task)
# Update task status to 1 (mapped)
  for the task in mapped_tasks:
    task_status[task] = 1
```

Dynamically Arriving Task Mapping:

```
while (there are upcoming tasks):
        upcoming_task = get_next_upcoming_task()
        greater_task=find_right_neighbor(upcoming_task)
        lesser_task = find left neighbor(upcoming_task)
        greater_vm = find_vm(greater_task)
        lesser_vm = find_vm(lesser_task)
        if (greater_vm.ready_time<lesser_vm.ready_time):
                map_task_to_vm(upcoming_task, greater_vm)
       else:
                map_task_to_vm(upcoming_task, lesser_vm)
```

4.2.2.2. Neighbouring-based Reservation for Fault Tolerance

After VM allocation, a fault handling algorithm that enables the proposed work to win fault tolerance if any VM fails or leaves the system is proposed. This fault handling algorithm has been developed by employing the technique of advance reservation of neighboring VMs. The advance reservation is the technique where the AR time slot is computed or

estimated and the VM is reserved for that predicted time slot to guarantee the task execution till completion. In the beginning, the TET_j for all VM_s is taken as zero. It means that currently, the particular VM has executed no task. Afterward, TET_j is updated after the finishing of each task on the VM. Moreover, every VM has some load history which is termed as the ready time of the VM. Initially, RT_j is taken as TET_j as shown in eq. (2)

$$TET_j = RT_j \tag{2}$$

After the mapping of t_i and v_j as per the allocation algorithm explained above every ti will start its execution on some v_j. This starting time of the execution of t_i on v_j is termed as EST_{ij} *and* is calculated as in eq. (3)

$$EST_{ij} = TET_j \tag{3}$$

After the execution of t_i on v_j is over, AFT_{ij} is determined by adding the total processing time of t_i on v_j $\{t_p(t_i, v_j)\}$ to the EST_{ij} as shown in eq. (4)

$$AFT_{ij} = EST_{ij} + t_p(t_i, v_j) \tag{4}$$

Where $t_p(t_i, v_j)$ is the time taken to process t_i by v_j and is calculated as in eq (5)

$$t_p(t_i, v_j) = \frac{t_{size}}{S} \tag{5}$$

Furthermore, TET_j is updated after every execution of t_i and will be equal to AFT_{ij} as shown in eq. (6)

$$TET_j = AFT_{ij} \tag{6}$$

However, for calculating AR slots, the proposed algorithm takes Early Start Time and Actual Finish Time as input parameters and estimates the AR slot as the difference between EST_{ij} and AFT_{ij}.

If the advance reservation strategy is not employed, the task (t_u)may fail to execute on VM_f i.e., the failed VM, or if the VM leaves the system for a certain time will result in the suffering of the corresponding task. To handle this situation, p failed VMs are defined as:

$$VM_f = \{V_f : V_f \in V \ \& \ o(VM_f) = p\}$$

and q corresponding affected tasks are defined which were executing on these failed VMs. The set of failed tasks is defined as:

$$T_f = \{t_f : t_f \in T \ \& \ o(T_f) = q \ \& \ q <= n\}$$

Now, these failed tasks need to be reallocated to some other suitable healthy VMs so that T_f will execute without any interruption. On reallocating, all the failed tasks T_f are migrated from V_f to VM_j such that:

$$VM_j \in V \ \& \ VM_j \notin V_f$$

The model reserves the neighboring VM of the corresponding failed task as an alternative VM. Later, the TET_j is again updated as shown in eq. (7):

$$TET_j = TET_j + t_p(t_f VM_j) \text{ where } VM_j \in V \&\& VM_j \notin V_f \quad (7)$$

Detecting Failed VMs and tasks

The model supports fault tolerance by resource reservation technique to offer a backup VM for the impacted task in the event of VM failure. Eq. (8) is used to determine the AR slot:

$$AR_{ij} = AFT_{ij} - EST_{ij} \quad (8)$$

NOT expected_performance_metrics (v_j) function is operated for discovering failed VMs. This confirms if the performing_metrics (v_j) function returns False for any VM. The expected_performing_metrics (v_j) is believed to return True if the VM's performance metrics such as AR_{ij}, $E(t_i, v_j)$ are inside expected ranges, and False otherwise. By applying Not, the condition happens to be True when the performance metrics are irregular or abnormal.

For example: If the AR_{ij} is extended as expected period, indicating it might be frozen or crashed.

The corresponding tasks of failed VMs will remain unexecuted and are represented by a set $Tu = \{tu : t_u \in T, |T_u| = u \text{ AND } u <= n\}$ and f failed VMs are represented as $V_f = \{v_f : v_f \in V \text{ AND } |V_f| = f\}$. To ensure uninterrupted operation of Tu, the task set tu must now be redistributed from vf to other relevant healthy vj provided $v_j \in V$ AND $v_j \notin V_f$. After every redistribution of task t_u in T_u to v_j in V, the TET_j is updated as shown in eq. (7)

The proposed fault tolerance algorithm is presented in Algorithm 2.

Algorithm 2: Neighbouring-based Reservation Algorithm for Fault Tolerance

 # Identification of failed VMs and tasks

1. **Load** ARM (t_i, v_j, AR, Status) /*Advance Reservation Matrix initialize all slots as zero
2. **For** all t_is in T
   Compute EST_{ij} and AFT_{ij} using eq. 3 and 4
   Compute AR slot using eq. 8
3. Identify_failed_VMs()
   V_f = []
   T_f = []

   For all t_is in T

   if NOT expected_performing_metrics (t_i)

T_f.append(t_i)

V_f.append(v_j|v_j → t_f in T_f)

 # Neighbouring-based Reservation for Fault Tolerance

Initialize_ARM (incoming_tasks, Mapped VMs, task_size, VM_Speed, EST_{ij}, AFT_{ij}, AR_{ij}, Status)

T_f = {t_f | t_f ∈ T_f, o(t_f) = q and q ≤ n}

for each t_f in T_f:

while Status(t_f) = 1:

if (t_{i-1}, t_{i+1} ∈ T && RT(t_{i+1}.VM) < RT(t_{i-1}.VM)):

Select t_{i+1}.VM (right neighbor) as alternative VM for t_f for AR_{ij}

// Reserve the time slot for the selected task

else if (t_{i+1} ∈ T && ti-1 ∉ T):

Select t_{i+1}.VM as alternative VM of t_f for AR_{ij}

// Reserve the time slot for the selected task

else:

Select t_{i-1}.VM as alternative VM of t_f for AR_{ij}

// Reserve the time slot for the selected task

Update_ARM (incoming_tasks, Mapped VMs, task_size, VM_Speed, EST_{ij}, AFT_{ij}, AR_{ij}, Status(tf) = 1)

// Status(tf) = 1 implies the AR slot is reserved for t_f

4.2.2.3. Load Balancing

Apart from all this, a load-balancing algorithm is also proposed which escorts the whole system for uniform load distribution that might be disturbed after fault-handling throughout the system and further improves the makespan and utilization. The under and overloaded VMs are identified by the Load balancing algorithm and the load is shifted from the overloaded VM to the underloaded VM for uniform distribution of load among VMs. The VMs having the highest and lowest makespan are taken as maximum overloaded and underloaded VMs respectively. Then (ϵ), the average execution time of tasks assigned over the maximum overloaded VM is calculated as in eq (9). The tasks with execution time less than the ϵ are taken as separate sets (Σ).

i.e., $\Sigma = \{t_i | E(t_i, v_j) < \epsilon$

$$\epsilon = \sum_{i=1}^{o} \frac{tp(ti,vj)\ \&\&\ ti\ \epsilon\ O}{|O|} \quad (9)$$

Finally, the load is shifted from an overloaded VM to the underloaded VM as described in the load balancing algorithm. Figure 4.5 depicts the flowchart of the proposed work.

4.2.3 The Proposed HFSLM

The allocation in HFSLM is done in three phases: In the allocation phase, we perform Task Sorting, VM Sorting, and Task Allocation. However, for dynamically arriving tasks, HFSLM provides a distinct algorithm for the allocation. In the second phase: fault tolerance is achieved by proposing an innovative fault-tolerant algorithm namely Neighbouring-based Reservation Algorithm for Fault Tolerance. Following fault tolerance, the model addresses the evenly distributed load among VMs by recommending a load-balancing technique.

HFSLM (O, €)

Call Algorithm 1;

Call Algorithm 2;

$\Sigma = \{ti \mid E(ti, vj) < €\}$

Sort set Σ in descending order of execution time

for each task t_i in Σ:

shift_task_to_underloaded_VM(t_i)

update_makespan()

makespan.overloaded_VM = makespan. overloaded_VM - execution_time(t_i)

makespan.underloaded_VM = makespan.underloaded_VM +execution_time(t_i)

if(makespan.underloaded_VM < makespan. overloaded_VM):

continue // Take another task from Σ

else:

rollback (makespan.underloaded_VM, makespan. overloaded_VM) // To the previous state

$t_i = t_{i+1}$ //take next task from Σ

end for

Estimate the QoS parameters Makespan, UT, and Overheads // as per eq. (10, 11, 12, and 13)

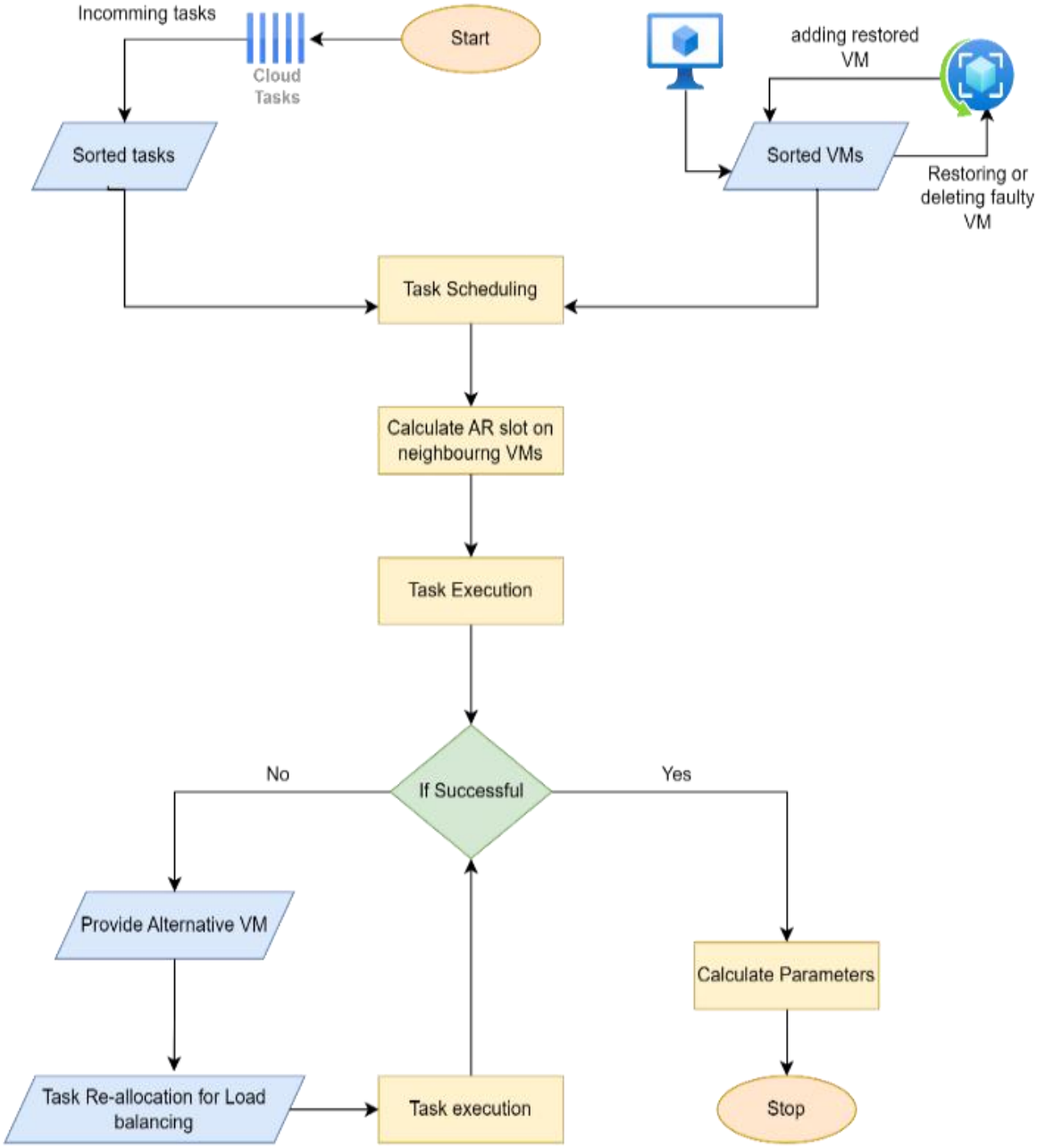


Figure 4.5: Flow Chart of HFSLM

4.2.4. Performance Metrics

Makespan (M) is taken as the highest or maximum among all TET_j and can be expressed as eq. (10).

$$M = max(TET_j), \forall Vj \tag{10}$$

Finally, the **Average VM utilization** of the system is defined as in eq. (11).

$$UT = \frac{\sum_1^k (TET - tp(tf \in Vf, Vj \in Vf))}{k * \text{Makespan}} \forall Vj \tag{11}$$

The **Fault Overhead** is the additional time or resources required to handle the faults to recover from failures and faults in the system. Fault Overheads in this chapter are caused by the following factors:

- Detecting AR Slot
- Reserving VM
- Reallocation etc.

Makespan Overhead (O_m) is the additional time required to complete the task and can be computed as follows:

$$O_m = \begin{cases} \frac{M_{beforeFT} - M_{afterFT}}{M_{beforeFT}} & After\ Fault\ tolerance \\ \frac{M_{afterLB} - M_{beforeFT}}{M_{beforeFT}} & After\ Load\ balancing \end{cases} \quad (12)$$

Average Resource Utilization Overhead (O_{ut}) is the additional resource required to complete the task and can be computed as follows:

$$O_{ut} = \begin{cases} UT_{beforeFT} - UT_{afterFT} & After\ Fault\ tolerance \\ UT_{afterLB} - UT_{afterFT} & After\ Load\ balancing \end{cases} \quad (13)$$

4.2.5. Computational Complexity of HFSLM

To compute the complexity of HFSLM, the basic operations are analyzed as a function of input size. We will express the complexity of the presented HFSLM in Big O notation:

Task Sorting and VM Sorting:

Sorting a list of n tasks concerning task size using QuickSort, the complexity is typically *O* (n log n)

Task Mapping:

- For n tasks and m VMs, we need to iterate through all VMs in the worst case for a suitable mapping. i.e.,
  O (n * m) iterations
- Operations of each iteration will take constant time.
- Complexity is O (n * m)

Dynamically arriving Task Mapping (for n arriving tasks):

- For identifying neighboring tasks of the arrived task, the algorithm takes constant time.
- Total number of iterations equals the number of arriving tasks (*n*).
- Complexity is O (n)

Neighbouring-based Reservation Algorithm for Fault Tolerance:

- For q failed tasks, the algorithm iterates for each failed task and performs constant time operations.

- The total sum of iterations depends on the number of failed tasks (*q*).
- Complexity is O (q)

Similarly, for the load balancing algorithm, if we assume k tasks in Σ have execution time less than ϵ, the complexity will be O (k)

HFSLM Algorithm Complexity:

The total complexity of the model is taken by adding all the individual complexities:

O (n log n) + O (n * m) + O (n) + O (q) + O (k)

As we can observe O (n * m) dominates other runtime operations because it depends on both the number of tasks and VMs. Therefore, the complexity of the model can be estimated as O (n * m).

4.3. Motivational Illustrative Example

This section demonstrates an explanatory and motivational example where the working of the proposed reservation-based fault tolerance and load balancing model has been expressed. An example to illustrate the model has been taken from the most recent paper where FTHRM [72] has been proposed and we have related our proposed model with FTHRM based on the same example.

Table 4.2: Instance of tasks and VMs

Task modeling for the illustration

Task (t_i)	**Task_size**
t_0	120 MI
t_1	260 MI
t_2	380 MI
t_3	90 MI
t_4	100 MI
t_5	220 MI
t_6	400 MI
t_7	280 MI
t_8	350 MI

VM modeling for illustration

Virtual Machine (Vi)	**VM_speed (s)**
V_1	10 MIPS, Ready time = 2 µs
V_2	12 MIPS, Ready time = 4 µs
V_3	14 MIPS, Ready time = 6 µs

Nine different independent tasks and three VMs have been taken to demonstrate the working of the proposed model. (*Note: the proposed model supports run-time dealing with both tasks and VMs as shown in Figures 4.3 and 4.4).* But for simplicity of an example, we are taking the instance of tasks and VMs as shown in Table 4.2. The Ready time of each VM is the previous load on the VM. Now, the allocation of tasks to the VMs has been done by the proposed strategy where the tasks and VMs are sorted initially according to increased *task_size* and *VM_speed*, respectively as shown in Figure 4.6.

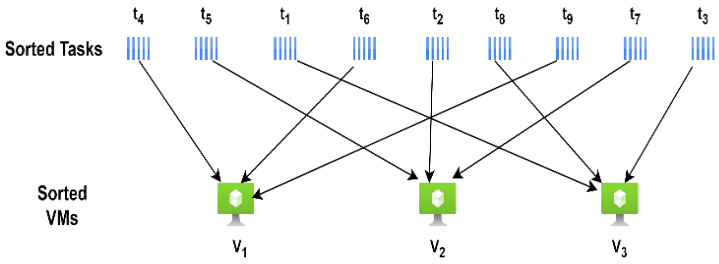


Figure 4.6: Task Allocation in the Proposed Strategy

The proposed allocation is illustrated in Figure 4.7. EST_{ij} is the time when the execution of t_i on v_j starts and AFT_{ij} is the time when the execution of t_i on v_j completes. AFT of the previously executed task on any VM becomes the EST of the next task on the same VM as explained in Figure 4.7. Furthermore, AFT_{ij} is computed by adding the execution time ($t_p(t_i, v_j)$) to the EST_{ij}. Here, $t_p(t_i, v_j)$ is computed as shown in eq. 5. TET of each VM is initialized to zero which indicates that no task has been executed on the particular VM.

However, in this example, each VM has its Ready time i.e., $RT(V_1) = 2$, $RT(V_2) = 4$, and $RT(V_3) = 6$. Now, the Ready time of V_j will be assigned to TET_j. In other words, $TET(V_1) = 2$, $TET(V_2) = 4$, $TET(V_3) = 6$.

4.3.1. Task Mapping

After sorting tasks and VMs, VMs are allocated to the tasks in the sorting order as shown in Figure 4.6. Initially, t_4 is allocated to VM_1 with EST = 2, t_5 is allocated to V_2 with EST=4, and t_1 is allocated to VM_3 with EST = 6. For t_4, $EST_{41} = 2$ because Ready time of VM_1 is 2, now to compute AFT_{41}, $t_p(t_4, v_1) = \frac{90}{10} = 9$ will be added to the EST_{41}. In other words, $AFT_{41} = 2+9 = 11$. After the execution of t_4 is over, $RT(V_1)$ will be updated to 11 and is the EST of the next task.

Similarly, for t_5, $EST_{52} = 4$ because Ready time of VM_2 is 4, now to compute AFT_{52}, $t_p(t_5, v_2) = \frac{100}{12} = 8.3$ will be added to the EST_{52}. In other words, $AFT_{52} = 4 + 8.3 = 12.3$. After the execution of t_5 is over, $RT(V_2)$ will be updated to 12.3 and is the EST of the next task.

For, t_1, the same thing happens, and $RT(V_3)$ will be updated to 14.5 and is the EST of the next task as illustrated in Figure 4.7.

The next task i.e., t_6 will be allocated to that VM whose RT has been already computed i.e., the VM which is available or free. If RT has been computed for more than one VM, then the next task will be allocated to the VM having minimum ready time. (*Note: as the VMs and task are sorted, RT for all the VMs will also come to be sorted. The same happens with this example also, there will be some variation in case little or minimum variation in*

arriving task size i.e., in low task heterogeneity cases). Finally, after the allocation of all tasks is over, we compute the Makespan and Average VM Utilization.

Since, $TET(v_1) = 68, TET(v_2) = 71.2, TET(v_3) = 73..7$ as shown in Figure 4.7.

Now, Makespan is computed as $max(TET_j), \forall v_j$

i.e., $max(68, 71.2, 73.7)$

$$\boldsymbol{Makespan = 73.7}$$

Average VM Utilization(U) is computed as:

$$\frac{sum\ of\ all\ TETs}{Total\ number\ of\ VMs\ X\ Makespan}$$

$$UT = \frac{68+71.2+73.7}{3X73.7} = \frac{213.1}{221.1} = \mathbf{96.3\%}$$

Comparing the Makespan and Average VM Utilization with FTHRM, the Makespan of FTHRM was found to be as 80 and utilization was 84.76% for the same example. The allocation in FTHRM was done according to the MCT strategy. In other words, the proposed allocation used in the model surpasses the MCT strategy as well on both makespan and average VM utilization.

4.3.2. Fault-tolerance

As per the algorithmic flow of the model, after allocation, we are performing fault tolerance of the system using a neighboring-based advance reservation. Before reservation--based fault tolerance, we have assumed and illustrated the random fault tolerance first so that we can compare our neighboring reservation fault tolerance with the assumed random fault tolerance. The implementation results are also compared with FTHRM in the results section.

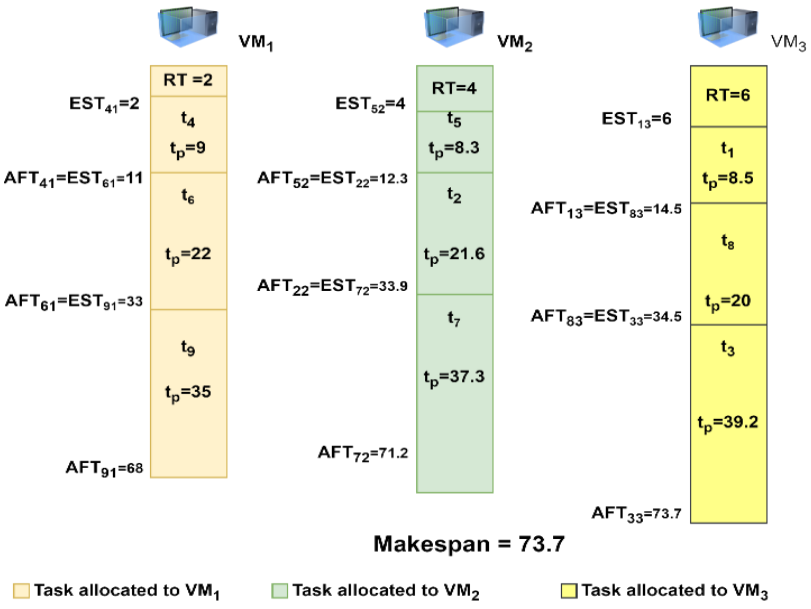


Figure 4.7**:** Allocation of Tasks in the Proposed Sorting Algorithm

4.3.2.1. Random Fault Tolerance without Neighboring Reservation:

Now, let's suppose V_3 failed at 34.5 as shown in Figure 4.8. Now if we randomly assign an alternative VM to the affected task which here is t_3. In Random allocation, we randomly pick any of the VMs and allocate them to t_3. Here we are migrating t_3 to VM_1 till t_3 completes its execution.

So, $AFT_{31} = 68 + 55 = 123.$

Hence, $TET(v_1) = 123, TET(v_2) = 71.2, TET(v_3) = 73.7 - t_p(t_3, v_3) = 73.7 - 39.2 = 34.5$ as shown in Figure 4.8.

Now, Makespan is computed as $max(TET_j),\ \forall v_j$

i.e., $max(123.\ 71.2,\ 34.5)$

$$\boldsymbol{Makespan = 123}$$

Average VM Utilization(U) is computed as: $\frac{sum\ of\ all\ TET}{Total\ number\ of\ VMs\ X\ Makespan}$

i.e., $UT = \frac{123+71.2+34.5}{3X123} = \frac{194.2}{246} = \mathbf{78.9\%}$

It is clear because of faults in any of the VMs, the makespan increases and utilization decreases.

$$O_m(After\ Faulttolerance) = \frac{M_{after\ FT} - M_{before\ FT}}{M_{before\ FT}} = \frac{123-73.7}{73.7}$$

$$\boldsymbol{O_m(After\ Faulttolerance) = 0.66}$$

It is clear because of faults in any of the VMs, the makespan increases and utilization decreases.

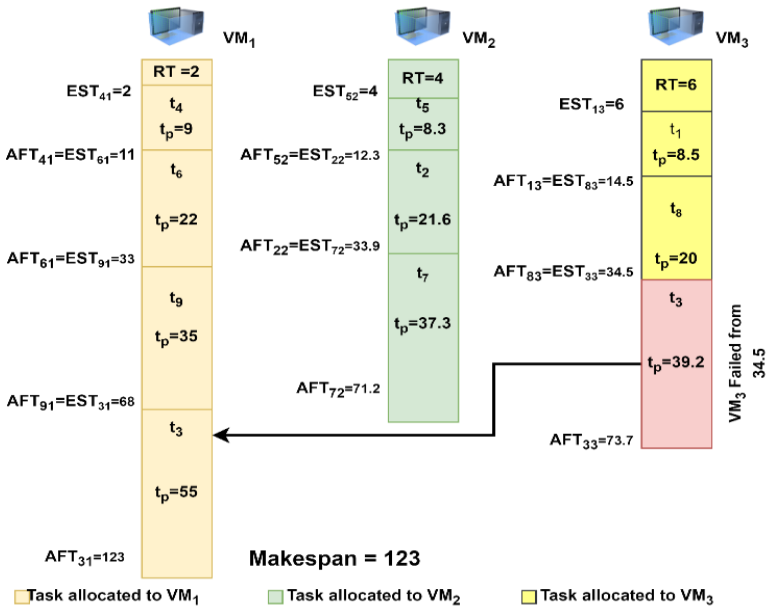


Figure 4.8: Random Fault Tolerance without Neighboring Reservation

4.3.2.2. Proposed Neighbouring Based Reservation:

Now, we are going to use a neighboring-based reservation to provide an alternative VM to the affected task. Neighboring-based reservation strategy selects the neighboring VM as an alternative VM for the affected tasks. As we have already sorted the tasks and VMs, therefore, reserving the neighboring VM will ensure that the same capacity of an alternative VM is reserved for the affected task. For using an advance reservation, we need to estimate the advance reservation slot represented as an AR_{ij} slot i.e., advance reservation slot for t_i on some VM_j and we are reserving the neighboring VM for the same AR slot to ensure task execution till completion of the task. AR_{ij} is estimated as the difference between AFT_{ij} and EST_{ij} given in eq 8.

Furthermore, all the information regarding tasks and VMs including AR slots are stored in a Matrix known as ARM as shown in Table 4.3.

The illustration of fault tolerance by reserving Neighbouring VMs is given in Figure 4.9. It is clear from the illustration, that, unlike random fault tolerance with reservation, VM_2 has been selected as an alternative VM because VM_2 is the neighbor of failed VM i.e., VM_3. The affected task i.e., t_3 has been migrated to VM_2 which will be of the approximately same capacity as that of VM_3. The computational flow is shown in Figure 4.9.

Again, let's suppose V_3 failed at 34.5 as shown in Figure 4.9. Now, using a Neighbouring-based reservation, we will select the Neighbouring VM for the affected task.

Table 4.3: Advance reservation matrix (ARM)

Task (*ti*)	**VM (V*j*)**	**Task_size (*w*(*ti*))**	**speed (s)**	***EST*(*ij*)**	***AFT*(*ij*)**	**AR slot (*ARsj*)**	**Status**
t1	VM_3	120	14	6	14.5	8.5	1
t2	VM_2	260	12	12.3	33.9	21.9	1
t3	VM_3	380	14	71.2	117	45.8	1
t4	VM_1	90	10	2	11	9	1
t5	VM_2	100	12	4	12.3	8.3	1
t6	VM_1	220	10	11	33	22	1
t7	VM_2	400	12	33.9	71.2	37.3	1
t8	VM_3	280	14	14.5	34.5	20	1
t9	VM_1	350	10	33	68	35	1

(*Note: here the failed VM_3 has only one neighboring VM i.e., VM_2. When the failing VM has both neighbours that time an alternative VM will be selected with the least Ready time*)

Here, we are migrating t_3 to its neighboring VM (VM_2) till t_3 completes its execution.

Since, $t_p(t_3, v_2) = \frac{550}{12} = 45.8$.

So, $AFT_{32} = 71.2 + 45.8 = 117$.

Hence, $TET(v_1) = 68, TET(v_2) = 117, TET(v_3) = 73.7 - t_p(t_3, v_3) = 73.7 - 39.2 = 34.5$ as shown in Figure 4.9.

Now, Makespan is computed as $max(TET_j),\ \forall v_j$

i.e., $max(68,\ 117,\ 34.5)$

$$\boldsymbol{Makespan = 117}$$

Average VM Utilization(U) is computed as: $\frac{sum\ of\ all\ TET}{Total\ number\ of\ VMs\ X\ Makespan}$

i.e., $UT = \frac{68+117}{2X117} = \frac{185}{234} = \mathbf{79.5\%}$

$$O_m(After\ Faulttolerance) = \frac{M_{after\ FT} - M_{before\ FT}}{M_{before\ FT}} = \frac{117-73.7}{73.7}$$

$$\boldsymbol{O_m(After\ Faulttolerance) = 0.58}$$

It is clear from the illustration that using reservation makespan and utilization improved. Although, in this example, the utilization is found to be increased by only 1%. This is because we have only three VMs here. Furthermore, the capacity of VMs varies only by 2MIPS. In real-time where we have a large number of VMs of extremely different capacities, this strategy will show huge improvements in both makespan as well as utilization. Same happens with O_{ut}.

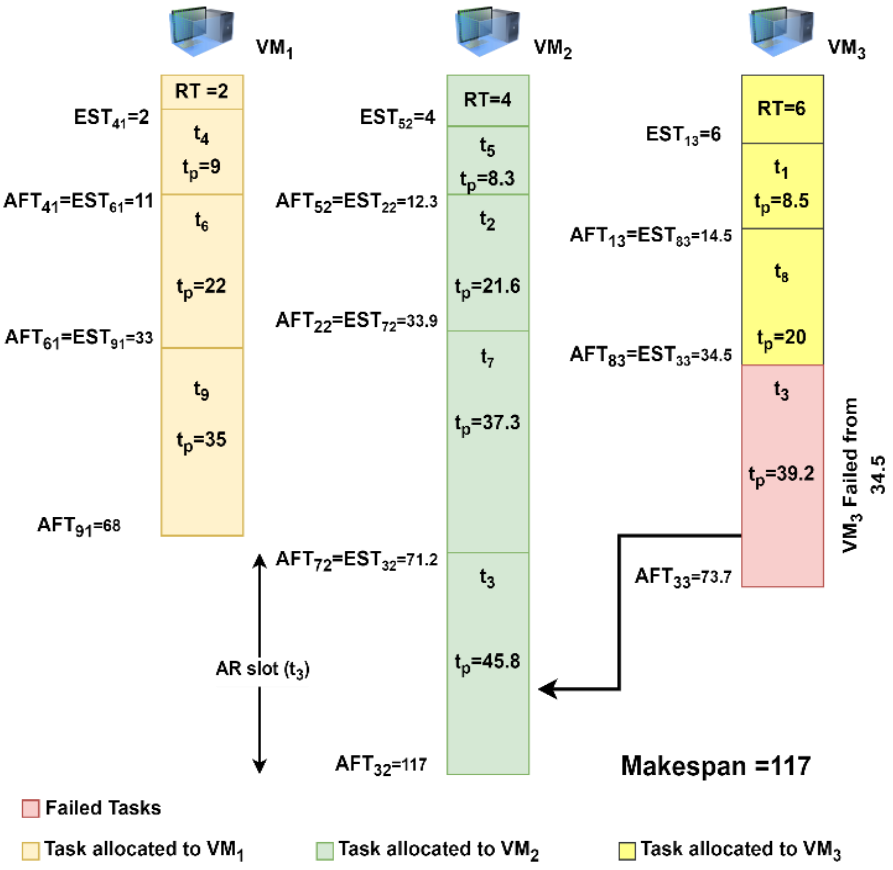


Figure 4.9: Fault Tolerance by Reserving Neighbouring VMs

4.3.3. Load Balancing

After performing fault tolerance, the model escorts the whole system with load balancing for further optimization of the makespan and utilization. The load balancing algorithm focuses on the maximum overloaded and minimum underloaded VM. After identifying the overloaded and underloaded VM, the load is shifted according to the given algorithm and depicted in Figure 4.10. In this example, after fault tolerance, the overloaded and underloaded VMs are identified as VM_2 and VM_1 respectively. Tasks executing on maximum overloaded VM and minimum underloaded VM are denoted as sets O and U respectively. Then, the average execution time (€) of tasks allocated over the maximum overloaded VM is calculated. The tasks having execution time less than the average execution time are taken as separate sets (Σ). I.e., $\Sigma=\{t_i \mid E(t_i, v_j) < €\}$

Overloaded and underloaded VMs are VM_2 and VM_1 respectively.

$O: \{t_5, t_2, t_7, t_3\}$ && $|O| = o$, $U: \{t_4, t_6, t_9\}$ && $|U| = u$

$$€ = \sum_{i=1}^{o} \frac{E(ti,vo) \&\& ti \in O}{|O|} = \frac{E(t5,v2)+E(t2,v2)+E(t7,v2)+E(t3,v2)}{4} = \frac{8.3+21.6+37.3+45.8)}{4} = \frac{113}{4}$$

€ = 28.5 as per eq. (9)

$\Sigma=\{t_2, t_5\}$ $E(t5, v2) = 8.3$ *and* $E(t2, v2) = 21.6$ (both are less than €)

Now Σ will be sorted in descending order of their execution time.

$\Sigma=\{t_5, t_2\}$

Now the algorithm will shift the load from VM_2 to VM_1 till the makespan of underloaded VM < makespan of overloaded VM (see load balancing algorithm). Here, we are migrating t_2 from VM_2 to VM_1. Now, Makespan is computed as:

$max(TET_j), \forall V_j$ *i.e., max (94, 95.4, 34.5)*

$$\boldsymbol{Makespan = 95.4}$$

Average VM Utilization(U) is computed as: $\frac{sum\ of\ all\ TET}{Total\ number\ of\ VMs\ X\ Makespan}$

$U = \frac{94+95.4}{2X95.4} = \frac{189.4}{190.8} = 99.2\%$

Since, $t_p(t_2, v_1) = \frac{260}{10} = 26.$

So, $AFT_{21} = 68 + 26 = 94.$

Hence, $TET(v_1) = 94, TET(v_2) = TET(v_2) - t_p(t_2, v_2) = 117 - 21.6 = 95.4, TET(v_3)$ will be the same i.e., 34.4 as it is a faulty VM as shown in Figure 4.10.

Now, Makespan is computed as $max(TET_j), \forall v_j$

i.e., $max(94, 95.4, 34.5)$

$$\boldsymbol{Makespan = 95.4}$$

Average VM Utilization(U) is computed as: $\frac{sum\ of\ all\ TET}{Total\ number\ of\ VMs\ X\ Makespan}$

$i.e.,\ U = \frac{94+95.4}{2X95.4} = \frac{189.4}{190.8} = \mathbf{99.2\%}$

$O_m(After\ Load\ balancing) = \frac{M_{after\ LB} - M_{before\ FT}}{M_{before\ FT}} = \frac{95.4-73.7}{73.7}$

$\boldsymbol{O_m(After\ Load\ balancing) = 0.29}$

The overhead of utilization (O_{ut}) is presented below:

$O_{ut}(total) = UT_{beforeFT} - UT_{afterRandomFT} = 96.3 - 78.9 = 17.4$

O_{ut} reduced using neighbouring reservation-based fault tolerance:

$UT_{afterNeighbouringFT} - UT_{afterRandomFT} = 79.5\% - 78.9\% = 0.6\%$

O_{ut} reduced by accompanying load balancing:

$UT_{afterLB} - UT_{afterRandomFT} = 99.2\% - 78.9\% = 20.3\%$

Therefore, the proposed load balancing has reduced the overhead by 20.3%.

The above illustration demonstrates the working of the whole hybrid model. Comparing the proposed model with FTHRM [72]. FTHRM shows the final utilization as 84.76 on the other hand proposed model shows the final utilization as 99.2% in the same example. It is because of escorting the proposed model with load balancing. Furthermore, FTHRM shows a makespan of 80 but the proposed model shows a makespan of 95.4. It is because the proposed model has reserved the neighboring VMs for the affected tasks and has also migrated the affected task from the failed VM to the neighbouring reserved VM. It is clear from the illustration that after load balancing the makespan and utilization have been optimized up to 18.8% and 20% respectively than before load balancing.

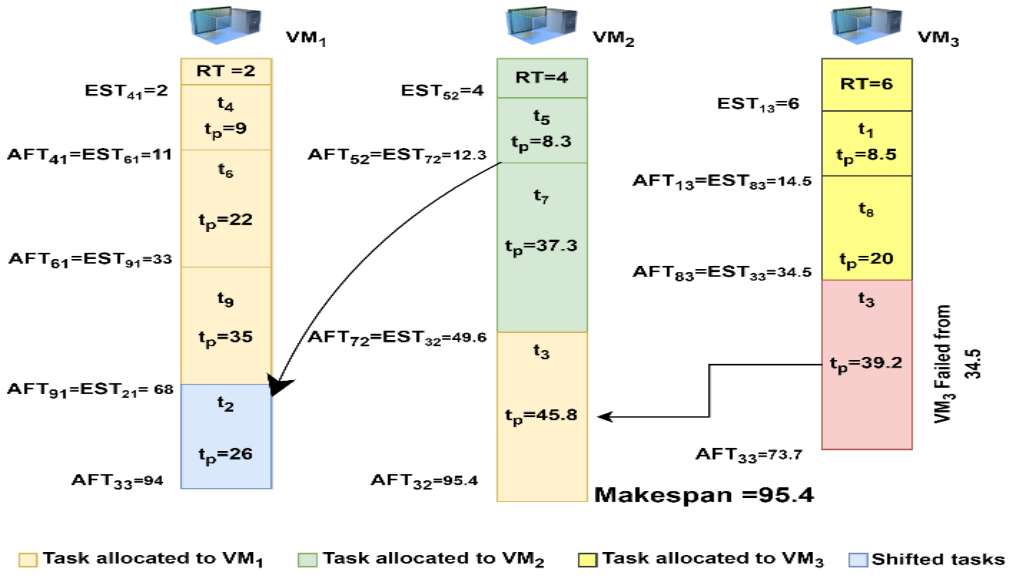


Figure 4.10: Load Balancing after Fault Tolerance

4.4. Results and Discussions

The proposed HFSLM is compared with four different approaches based on two main parameters i.e., makespan and average utilization. HFSLM is evaluated by comparing it with FTHRM, OLB, MIN MIN, and MAX MIN for less than 1000 tasks. Also, compared with ELISA and MELISA with greater than 10,000 tasks. The results were observed on varying the number of tasks and the number of VMs. Furthermore, task and machine heterogeneity are also varied to analyze the results of the proposed model more clearly. As mentioned in [35], the range of Expected Time to Compute (ETC) for ti on v_j is variable as heterogeneity varies from low to high for both the arriving tasks and VMs. By altering the heterogeneities of the incoming tasks and virtual machines, HFSLM is evaluated in this section.

4.4.1. Varying heterogeneity over small task scale

The evaluation has been done by adjusting the number of tasks, size of tasks, number of VMs, and capacities of VMs in four different heterogeneities given by [35], i.e., high task-high machine heterogeneity (HH), high task-low machine heterogeneity (HL), low task-high machine heterogeneity (LH), low task-low machine heterogeneity (LL). For all these four cases the performance of the proposed model and compared strategies have been analyzed and depicted graphically in the given figures. In comparison, the tasks have been taken on a small scale varying from 250 to 1000. On the other hand, the virtual machines have varied from 16 to 128. Additionally, the input parameters taken to analyze the considered model are further shown in Table 4.4.

Table 4.4: Simulation parameter used for HFSLM evaluation

S.no.	Input parameter	Range
1	No of tasks (n)	250 to 1000
2	No of resources (m)	16 to 128
Task size (t_size)		
2	Low task heterogeneity	1 MI to 100 MI
3	High task heterogeneity	100 MI to 3000 MI
VM Speed (S)		
5	Low machine heterogeneity	1 MIPS to 10 MIPS
6	High machine heterogeneity	10 MIPS to 100 MIPS

4.4.1.1. High task – High machine heterogeneity (HH)

In high task heterogeneity, the task size ranges from 100 MI to 3000 MI, and high machine heterogeneity ranges from 10 MIPS to 100 MIPS. A few observations regarding the considered parameters, i.e., makespan and average resource utilization are depicted in Figure 4.11 and 4.12 respectively. Further details of the observations are as follows:

- The proposed HFSLM is enhancing the makespan because of its planned features. For all the ranges of task number and VM number considered, the model surpasses other strategies by offering a minimum makespan. Apart from this, it is clearly seen in Figure 4.11, that at large task scale and small VM scale, the makespan shown is quite large but in that case also, HFSLM offers an optimized makespan.
- Furthermore, on comparing average resource utilization, HFSLM beats about 80% of the compared approaches. Additionally, for small-scale tasks, HFSLM shows better utilization than MAXMIN and as the number of tasks and VMs are going towards extremely large scales, HFSLM and MAXMIN go almost equally.
- Out of all the compared approaches, OLB performs worst in both makespan and utilization. It is probably because of the fact that OLB does not follow any plans and strategies.

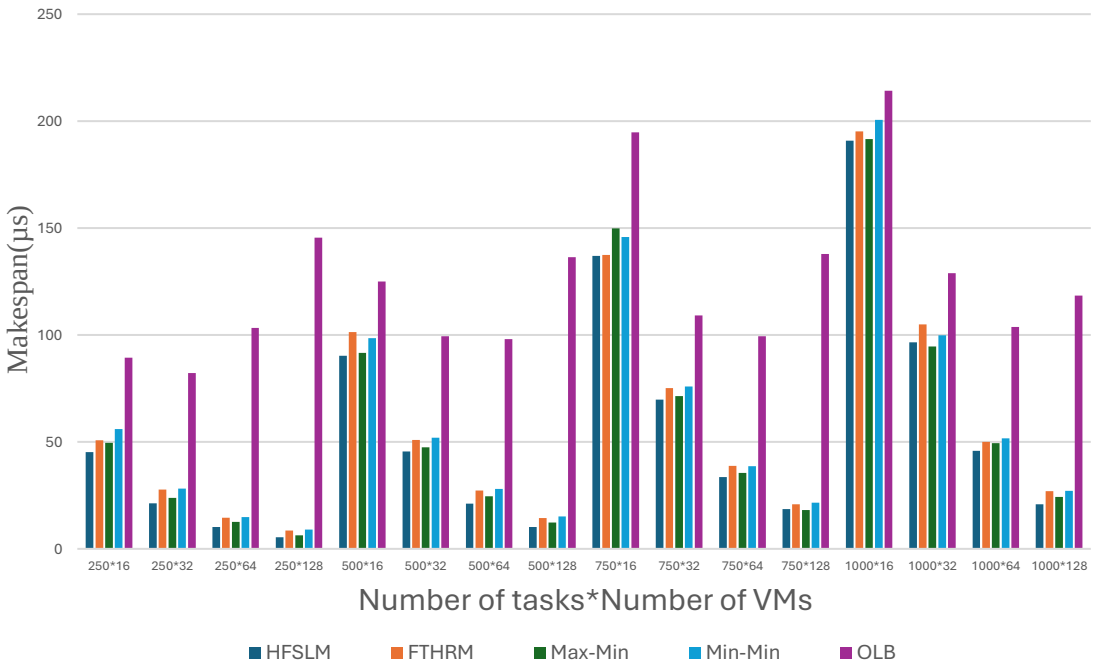


Figure 4.11: Makespan for varying Tasks and VM (HH)

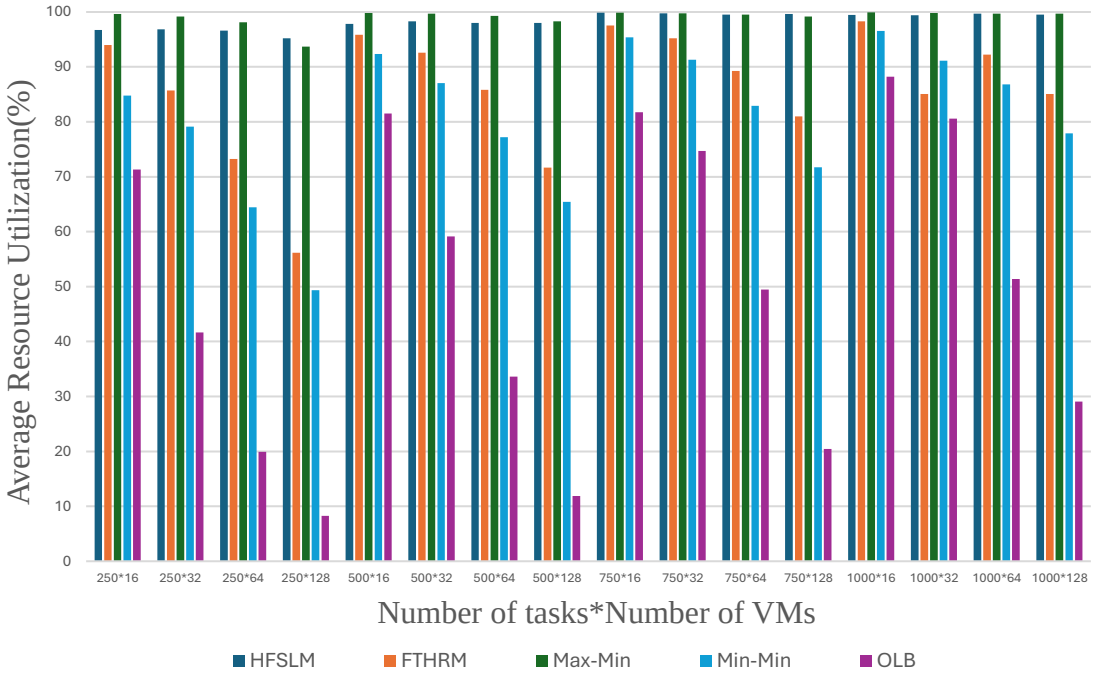


Figure 4.12: Avg. Resource Utilization for varying Tasks and VM (HH)

4.4.1.2. High task – Low machine heterogeneity (HL)

In high task heterogeneity, the task size ranges from 100 MI to 3000 MI, and low machine heterogeneity ranges from 1 MIPS to 10 MIPS. A few observations regarding the considered parameters i.e., makespan and average resource utilization are depicted in Figures 4.13 and 4.14. Further details of the observations are as follows:

- The proposed HFSLM is enhancing the makespan in HL because of its planned features. For all the ranges of task number and VM number considered, the model surpasses other strategies by offering a minimum makespan. Moreover, as depicted in Figure 4.13, HFSLM provides an optimized makespan even in the case of large task numbers and small available VMs. However, in such cases, MAXMIN offers the highest makespan.
- On comparing average resource utilization, HFSLM again efficiently beats about 80% of the compared approaches in HL also. In HL, the proposed model also beats MAXMIN in the case of a small task scale, and as the task scale goes up, both HFSLM and MAXMIN go with almost tie.
- Out of all the compared approaches, OLB provides the worst makespan. However, in some cases, MAXMIN also did not show an optimized makespan. Additionally, in the case of average resource utilization, OLB offers very limited utilization than all the considered approaches.

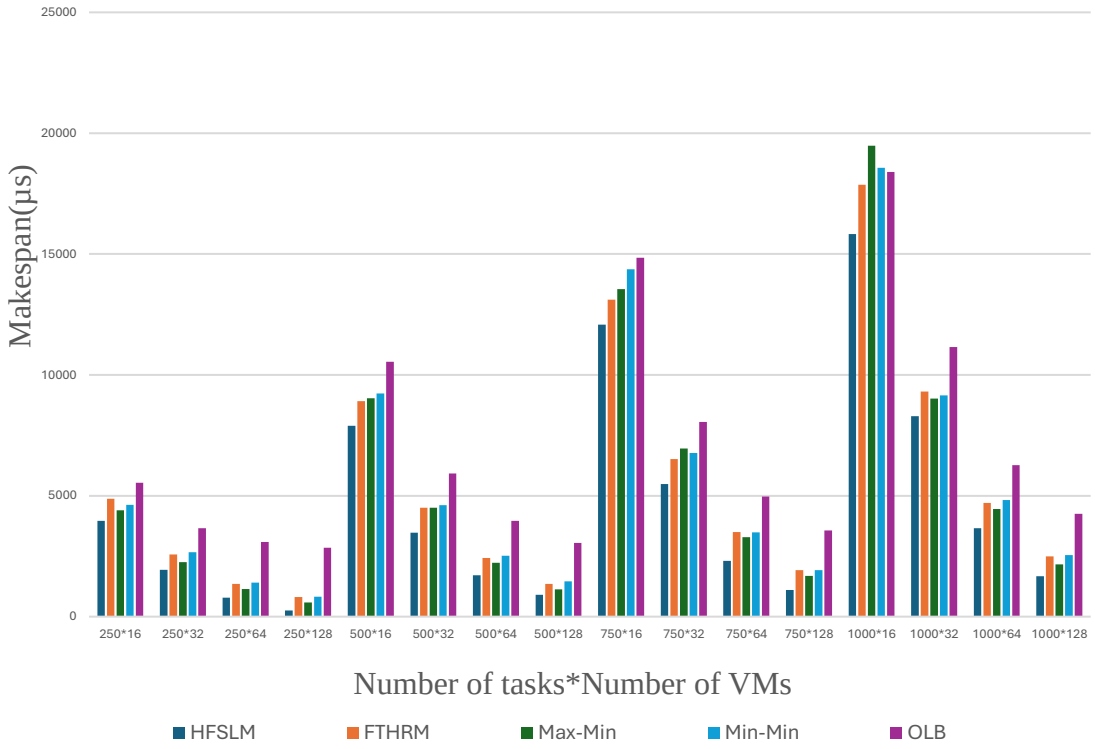


Figure 4.13: Makespan for varying Tasks and VM (HL)

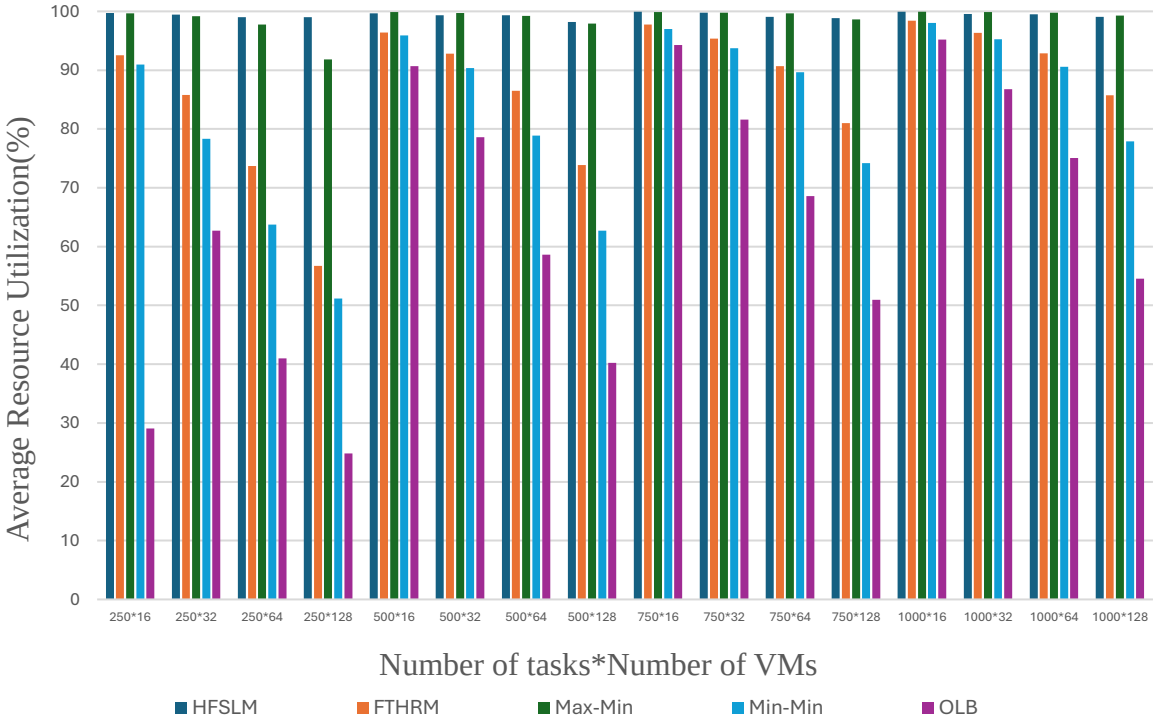


Figure 4.14: Avg. Resource Utilization for Varying Tasks and VM (HL)

4.4.1.3. Low task – High machine heterogeneity (LH)

In low task heterogeneity, the task size ranges from 1 MI to 100 MI, and high machine heterogeneity ranges from 10 MIPS to 100 MIPS. A few observations regarding the considered parameters i.e., makespan and average resource utilization are depicted in Figures 4.15 and 4.16. Further details of the observations are as follows:

- In this particular case, the proposed HFSM cannot beat MAXMIN with respect to the makespan in some scenarios. As depicted in Figure 4.15, the makespan

provided by HFSLM and MAXMIN were equal. However, there are rare cases where HFSLM showed a little better makespan than that of MAXMIN.

- On comparing average resource utilization, HFSLM again efficiently beats all the compared approaches except MAXMIN. For a low task scale, HFSLM beats MAXMIN. However, the performance of HFSLM in the mid-scale goes down than that of MAXMIN but as the task scale goes up, HFSLM shows optimized utilizations.

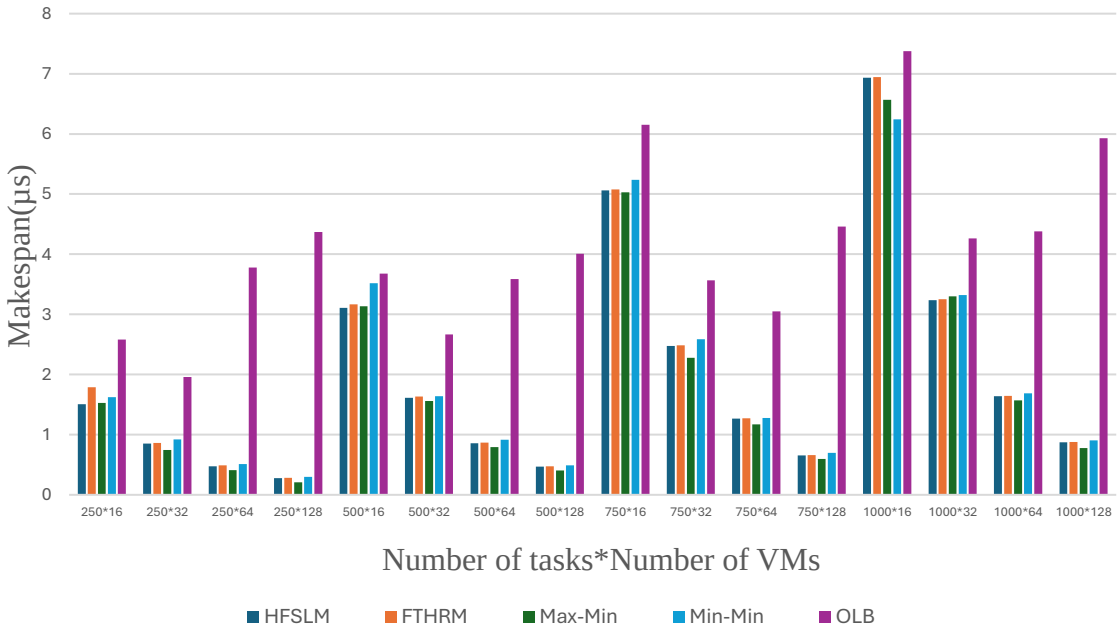


Figure 4.15: Makespan for varying Tasks and VM (LH)

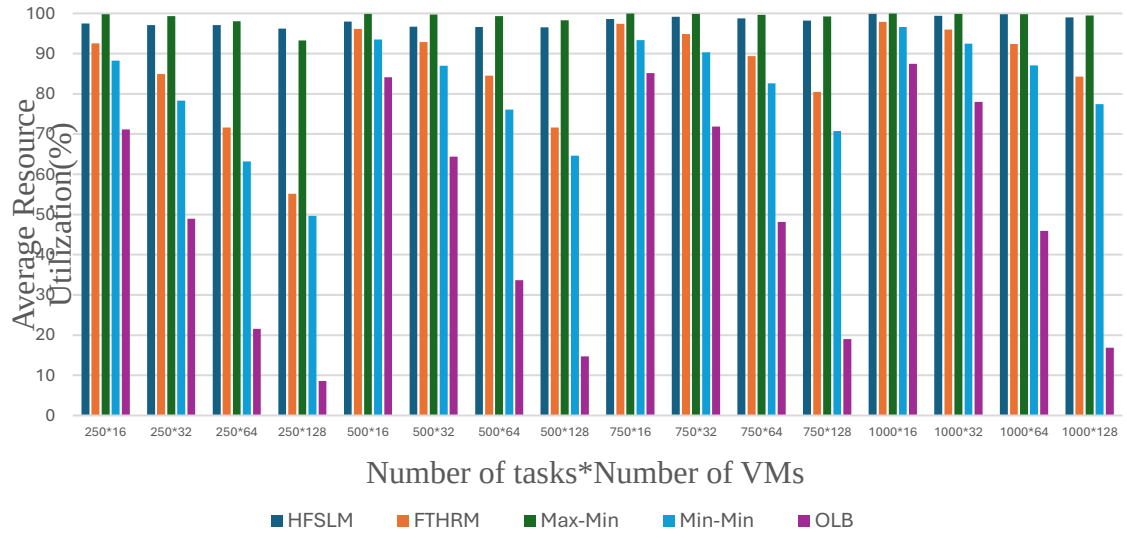


Figure 4.16: Avg. Resource Utilization for varying Tasks and VM (LH)

4.4.1.4. Low task – Low machine heterogeneity (LL)

In low task heterogeneity, the task size ranges from 1 MI to 100 MI, and low machine heterogeneity ranges from 1 MIPS to 10 MIPS. A few observations regarding the considered parameters i.e., makespan and average resource utilization are depicted in Figures 4.17 and 4.18. Further details of the observations are as follows:

- In the case of LL, the proposed HFSLM beats MAXMIN concerning the makespan. As depicted in Figure 4.17, the makespan provided by HFSLM for small task scales goes up, both HFSLM and MAXMIN are in a tie.
- On comparing average resource utilization, HFSLM utilizes the resource efficiently. However, MAXMIN and HFSLM behave almost the same in all task scales.

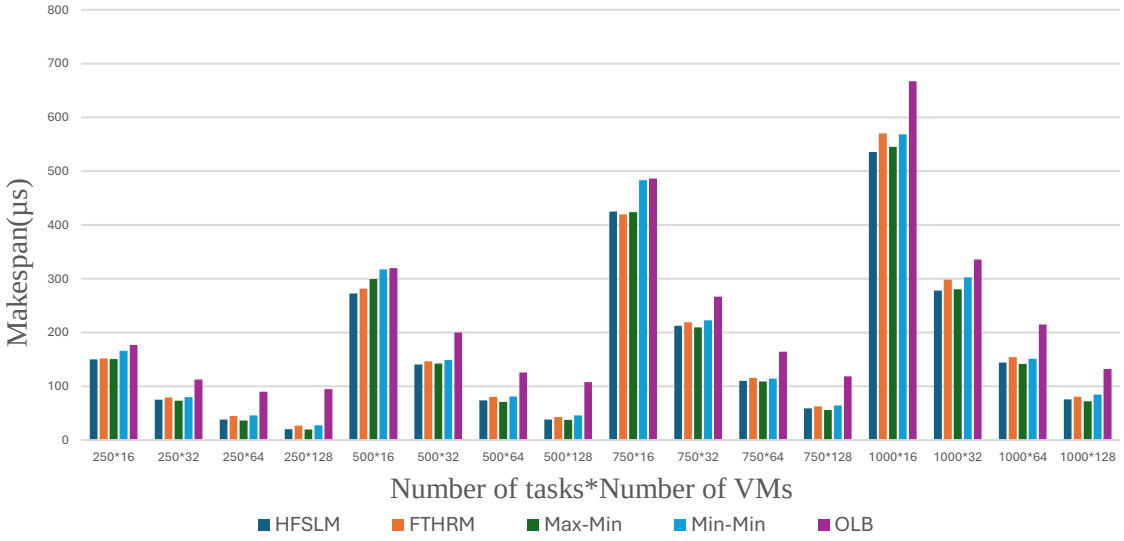


Figure 4.17: Makespan for varying Tasks and VM (LL)

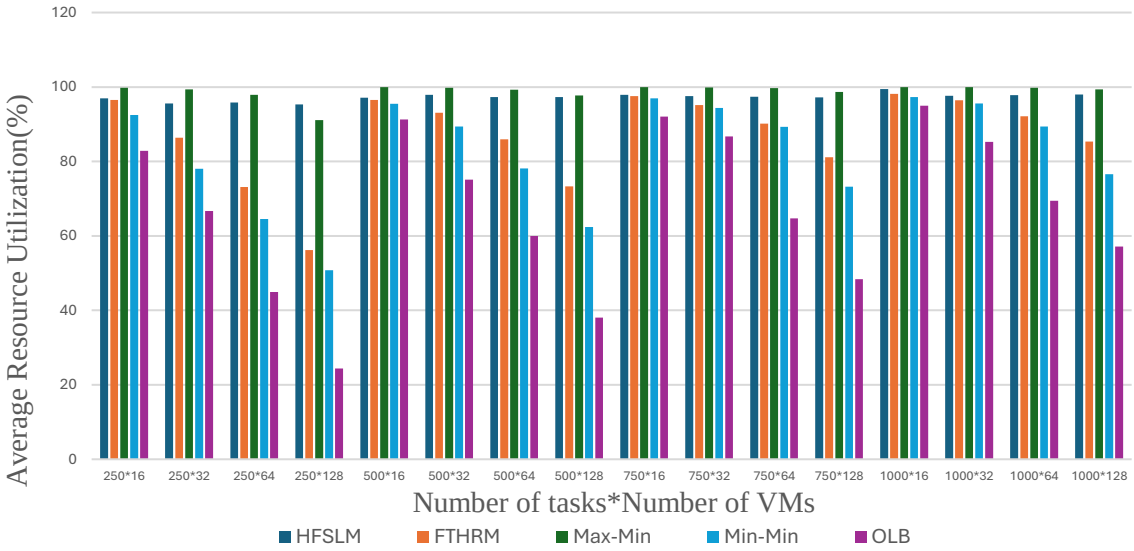


Figure 4.18: Avg. Resource Utilization for varying Tasks and VM (LL)

Observations

The suggested technique outperforms FTHRM in terms of makespan and utilisation, which go from 0.72% to 10.8% and 1.01% to more than 50%, respectively. When compared to MAXMIN, HFSLM exhibits makespan improvements of -3.03% to 8.8% and average resource utilisation gains of -2.15% to 6.7%. While comparing the suggested approach with MINMIN, the model shows an improvement of 0.6% to 19% in makespan and 1.09% to more than 45% in utilization. However, OLB was seen to perform very weakly among all approaches where the suggested model shows improvements of more than 50% in both makespan and utilization than OLB. Furthermore, it was observed that all the models perform almost equal optimization in makespan in LH heterogeneity. However, in that case, also OLB performs weakly among the compared approaches.

4.4.2. Varying heterogeneity over large task scale

The suggested model has also been contrasted with two other popular load-balancing models. i.e., ELISA and MELISA, and was evaluated based on makespan and average resource utilization.

4.4.2.1. Makespan and Average Resource Utilization

However, while comparing these HFSLM with these two models, the model was tested on an extremely large task scale. This is because these two models are more suitable for large task scales [46], [131]. Here, the tasks varied from 10000 to 50000. The average results of the makespan are depicted in Figure 4.19. The results show that the makespan of the proposed HFSLM is better than ELISA and MELISA. However, as the number of tasks increases the average makespan of MELISA becomes optimized.

On comparing average utilization, the three models are compared on the basis of Min, Avg, and Max average resource utilization. In Figure 4.20, it can be noted that there is a significant variation in the range of virtual machine utilization for ELISA and MELISA. However, for HFSLM, the range of variations between minimum, average, and maximum utilization is almost negligible.

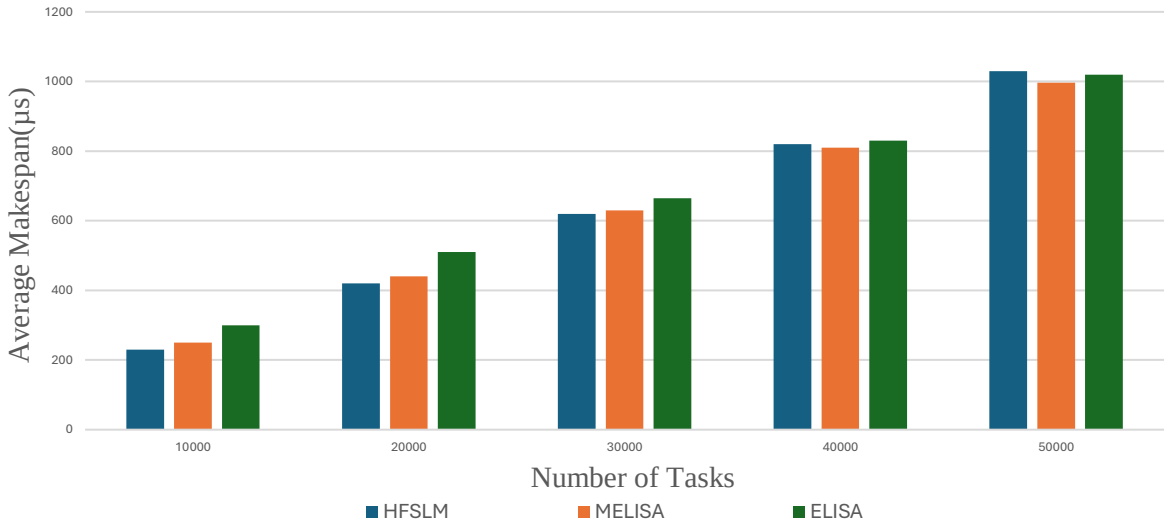


Figure 4.19: Average Makespan for varying Heterogeneity

The proposed model was seen to perform optimally in the case of utilization in all cases of heterogeneity. However, the model could not perform optimally on makespan in a few cases.

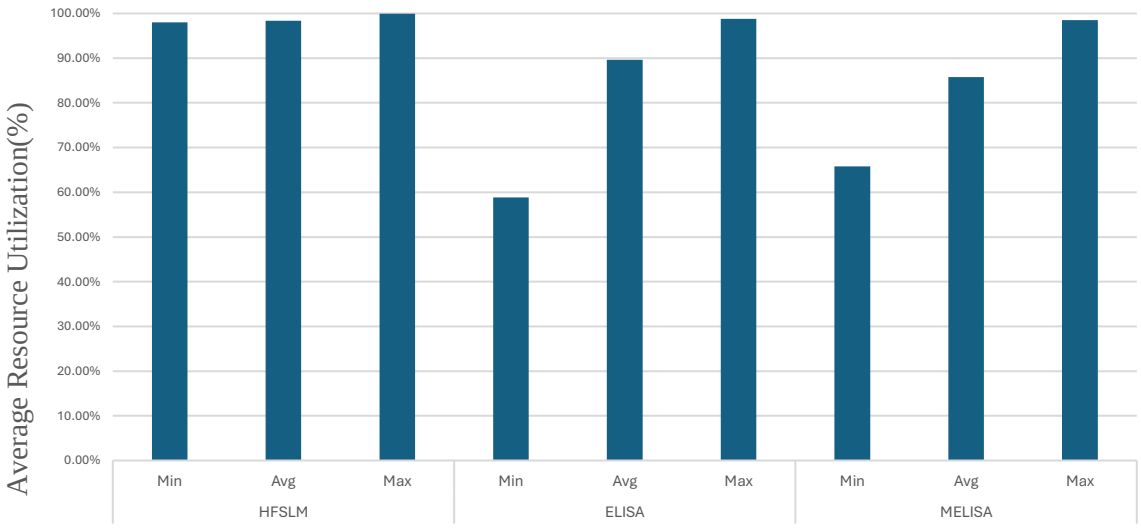


Figure 4.20: Avg. Resource Utilization for varying Heterogeneity

Observations: Additionally, Comparing HFSLM with ELISA and MELISA on a large tasks scale, HFSLM shows improvements from -0.98% to 23.33% and from -3% to 8% on

makespan respectively. Besides, HFSLM shows 1.42% and 1.22% improvements in minimum resource utilization as compared to ELISA and MELISA respectively. On maximum resource utilization, the proposed model shows improvements of 39.1% and 48.8% respectively.

Remarks: The suggested approach outperforms another strategy for QoS parameters. A few reasons are listed below:

- The proposed allocation considers both the upcoming tasks and newly added and deleted VMs. Additionally, optimal load distribution and effective average resource utilization occur simultaneously. As a result, it provides significant enhancement in all considered parameters.
- As can be seen from the overall results the utilization of the proposed approach remains optimized on varying the number of tasks and VMs. This is because of the fact that the proposed allocation strategy focuses on distributing the arriving tasks throughout the available VMs. Moreover, various strategic advancements in the proposed HFSLM play a significant role in the same.
- Furthermore, the proposed model outperforms all the compared approaches in HH and HL cases. It is because in high task heterogeneity the ready time of all the available VMs will always be sorted in other words, whenever we have high task heterogeneity, the ready time of all the VMs in the VM list will always be sorted. The sorted ready time of VMs is the best case for the proposed allocation.

4.4.2.2. Associated Fault Overheads (Makespan and Average Resource Utilization)

The fault always suffers from overheads even if handled. Likewise, the proposed fault tolerance also suffers from overheads. However, the proposed HFSLM reduces the associated fault overheads by balancing the load after handling faults. This achieves optimization in makespan and average resource utilization by reducing the associated overheads. The reductions in associated overhead, concerning both makespan and average resource utilization after load balancing, are illustrated across all four scenarios of task and VM heterogeneities and are depicted in the following Figure 4.21 to Figure 4.28.

High task – High machine heterogeneity (HH)

- The HFSLM model proposed in this study achieves a notable optimization in makespan overhead, reaching up to a significant level of 65.17% improvement by adjusting both tasks and VMs in terms of makespan overhead post to the load balancing, as depicted in Figure 4.21.

- By efficient re-distribution of load among VMs, a decrease of 8.93% in resource utilization overhead is observed following the implementation of load balancing, as illustrated in Figure 4.22 and the reduction of overhead of utilization is depicted in Figure 4.23.

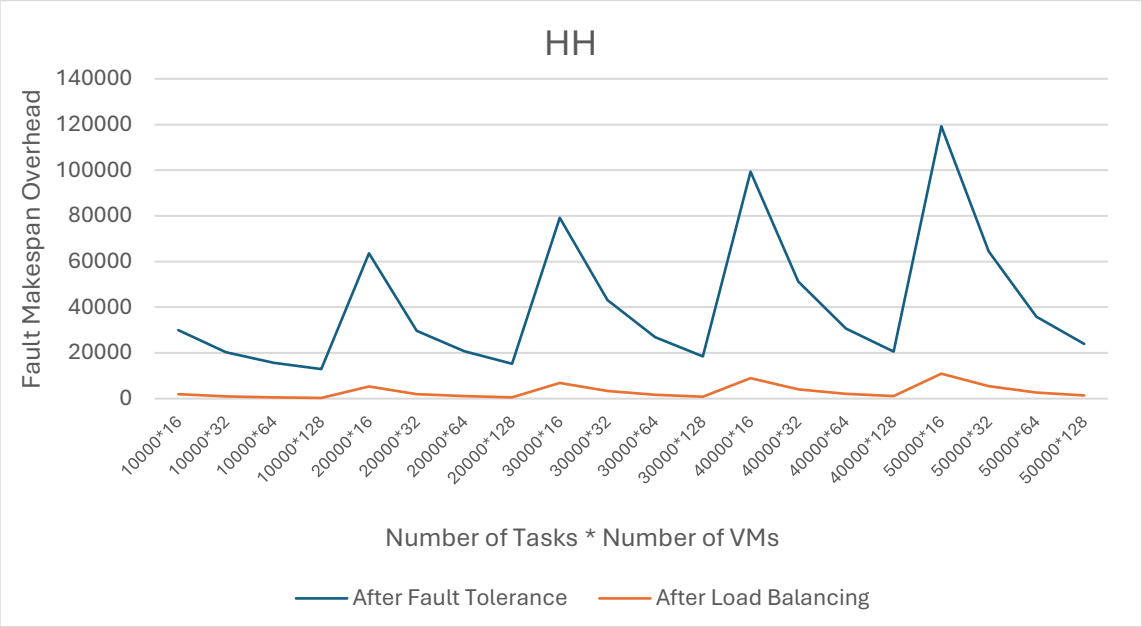


Figure 4.21: Fault Makespan Overhead for varying Tasks and VM (HH)

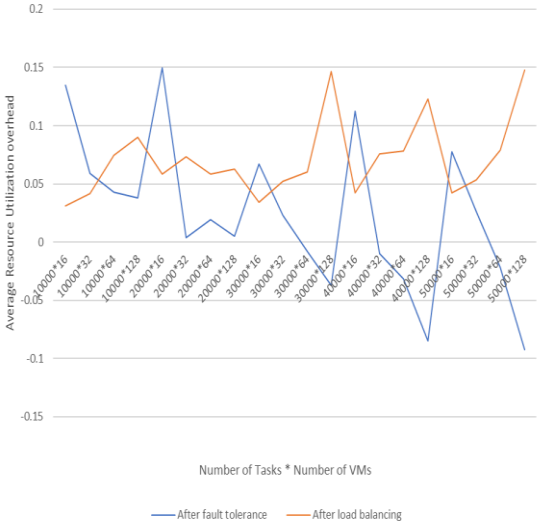


Figure 4.22: Showing Utilization Overhead in HH

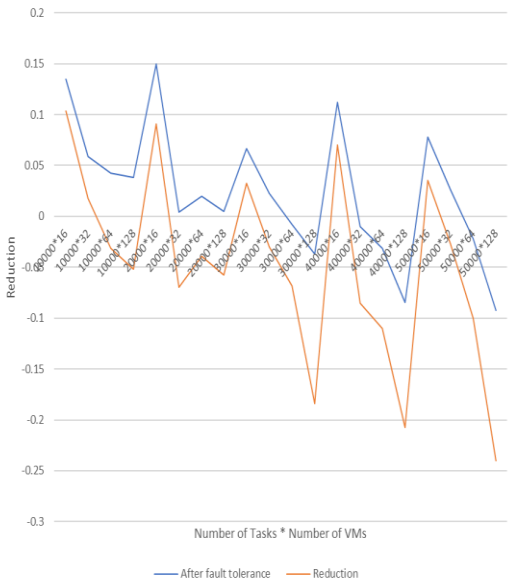


Figure 4.23: Showing Reduction of Utilization Overhead w.r.t. Total Overhead in HH

High task – Low machine heterogeneity (HL)

- The HFSLM demonstrates its efficacy in reducing makespan overhead across various tasks and VMs after load balancing, achieving a substantial decrease of more than 50% in makespan overhead, as depicted in Figure 4.24.

- The overhead related to the resource utilization is shown in Figure 4.25 and the reduction of utilization overhead is seen to be 4.49% after load balancing as depicted in Figure 4.26.

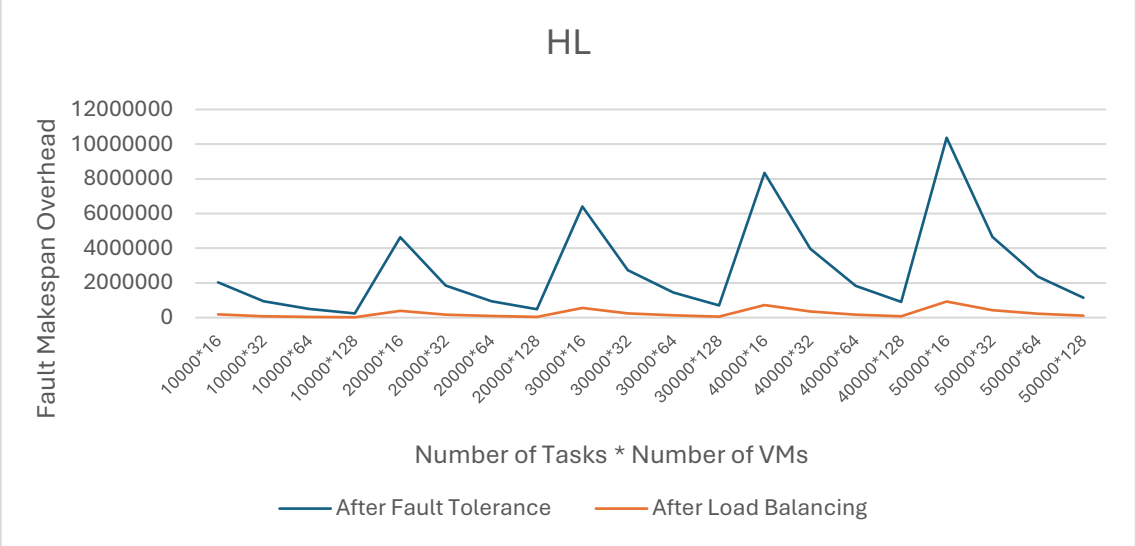


Figure 4.24: Fault Makespan Overhead for varying Tasks and VM (HL)

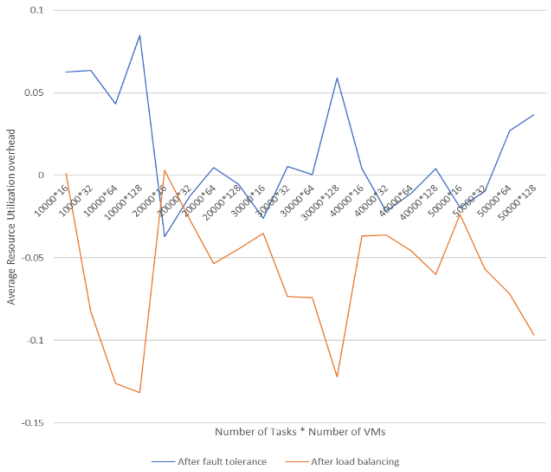


Figure 4.25: Showing Utilization Overhead in HL

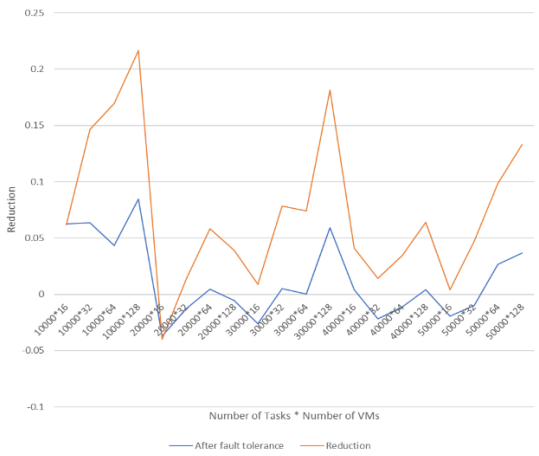


Figure 4.26: Showing Reduction of Utilization Overhead w.r.t. Total Overhead in HL

Low task – High machine heterogeneity (LH)

- The proposed load balancing strategy optimizes the makespan overhead to a very significant level up to 64.06% on widely varying tasks and VMs as can be seen in Figure 4.27.
- Figure 4.28 shows the utilization overhead. Nonetheless, Figure 4.29 demonstrates a significant reduction in resource utilization overhead, attributed to the effective redistribution of workload among virtual machines. We note a reduction of up to 5.15% in utilization overhead after load balancing.

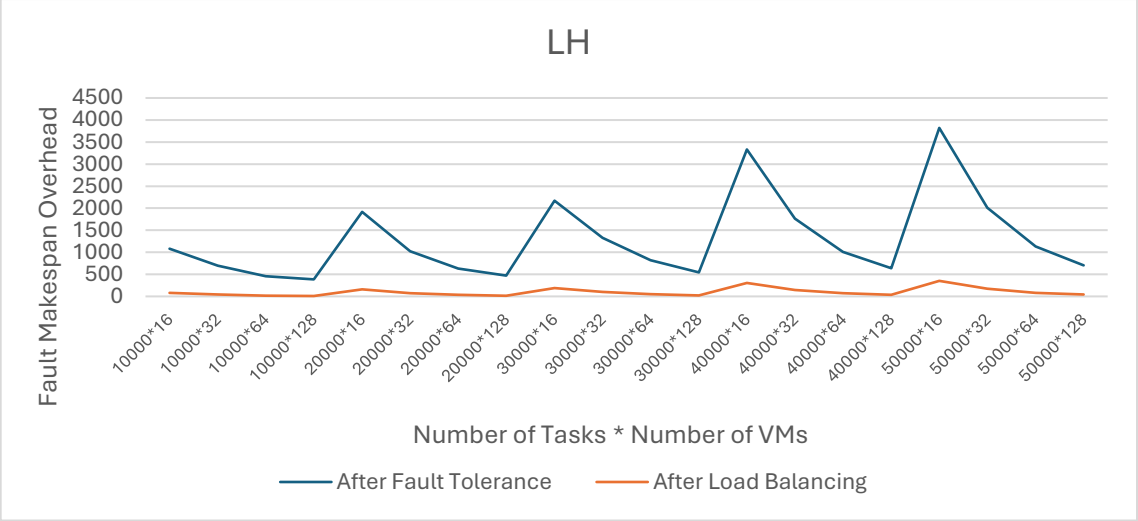


Figure 4.27: Fault Makespan Overhead for varying Tasks and VM (LH)

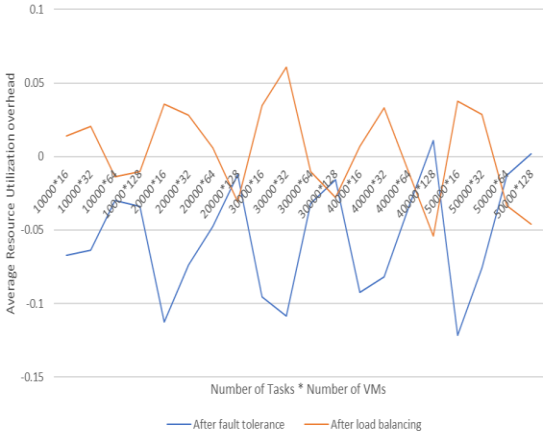


Figure 4.28: Showing Utilization Overhead in LH

Figure 4.29: Showing Reduction of Utilization Overhead w.r.t. Total Overhead in LH

Low task – Low machine heterogeneity (LL)

- The proposed load balancing strategy achieves a remarkable level of reduction in makespan overhead, reaching up to 34.12% across a diverse set of tasks and VM configurations, as illustrated in Figure 4.30.
- As visualized in Figures 4.31 and 4.32, the average resource utilization overhead is slightly decreased in some cases after load balancing particularly in cases of small task range up to 2000 tasks. In this task and machine heterogeneity, we observe the overall -0.75% changes in utilization after load balancing. The reason for this degradation is that in low VM heterogeneity, very few VMs participate in the load-shifting phase of the suggested load-balancing technique which results in less task redistribution.

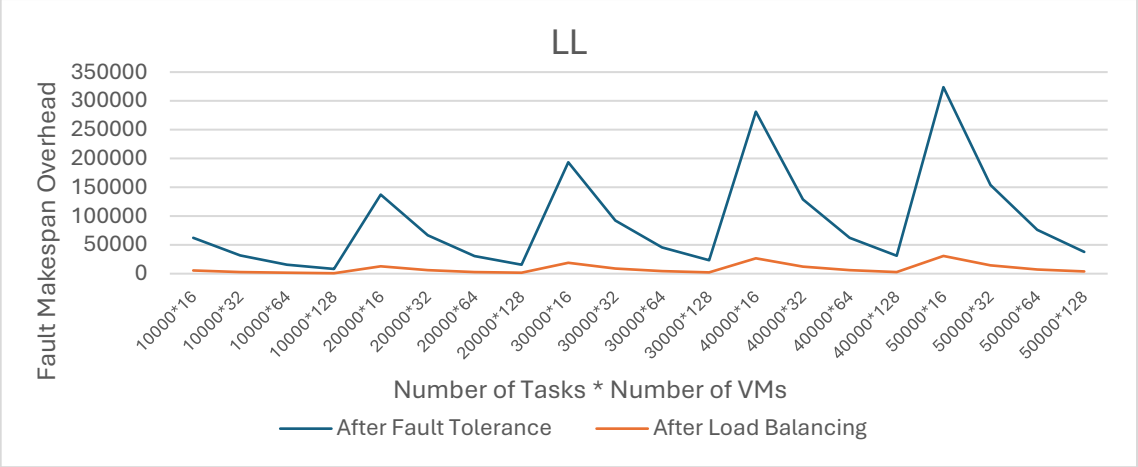


Figure 4.30: Fault Makespan Overhead for varying Tasks and VM (LL)

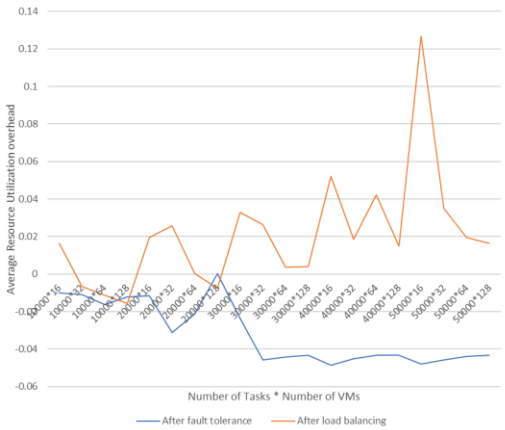


Figure 4.31: Showing Utilization Overhead in LL

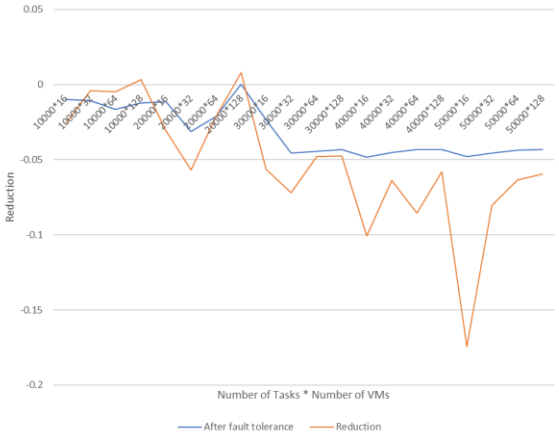


Figure 4.32: Showing Reduction of Utilization Overhead w.r.t. Total Overhead in LL

Observations

From Figures 4.21 to 4.24, we observe that the fault overheads for makespan have been significantly reduced after load balancing. Particularly, as the task number is increasing, the rate of reduction in overheads is also increasing. On the other hand, from Figures 4.25 to 4.28, we observe that the fault overheads for average resource utilization have been reduced thereby increasing the resource utilization after load balancing. Particularly, for HH, the rate of overhead reduction has been seen to be significant. This is because, for both high task and high VM heterogeneity, the proposed load balancing involves high load shifts while redistributing the load. However, for low VM heterogeneity, the proposed load balancing involves fewer load shifts while redistributing the load which leads to less reduction in the fault utilization overhead.

4.5. Summary in Context

In the proposed study, a Hybrid Fault-tolerant Scheduling and Load balancing Model is introduced employing neighboring-based VM to control failure in the cloud system with high computational demands. HFSLM uses a proficient task allocation strategy and distributes the arriving tasks among VMs at the arrival. In case of fault, the model uses the neighboring VMs of the faulty VM as a substitute and allocates an alternate VM to the affected task. Moreover, the proposed model escorts the whole system with an efficient load-balancing algorithm and maintains load equilibrium post-to-fault tolerance. After the implementation of the model in Python, performance evaluation was carried out by comparing HFSLM with FTHRM, MIN-MIN, MAX-MIN, and OLB on a low task scale by varying the task and VM in four different heterogeneities. The evaluations were performed based on makespan and average VM utilization. On very large task scales, the model was also contrasted with two other emerging models i.e., ELISA and MELISA.

The suggested approach outperformed other considered strategies for QoS parameters. A few reasons are listed below:

- *The proposed allocation considers both the upcoming tasks and newly added and deleted VMs. Additionally, optimal load distribution and effective average resource utilization occur simultaneously. As a result, it provides significant enhancement in all considered parameters.*
- *As can be seen from the overall results the utilization of the proposed approach remains optimized on varying the number of tasks and VMs. This is because the proposed allocation strategy focuses on distributing the arriving tasks throughout the available VMs. Moreover, various strategic advancements in the proposed HFSLM play a significant role in the same.*
- *Furthermore, the proposed model outperforms all the compared approaches in HH and HL cases. It is because in high task heterogeneity the ready time of all the available VMs will always be sorted in other words, whenever we have high task heterogeneity, the ready time of all the VMs in the VM list will always be sorted. The sorted ready time of VMs is the best case for the proposed allocation.*
- *The fault overheads with respect to both the makespan and average resource utilization have been reduced significantly in all four task and machine heterogeneities. However, in the case of LL case, the reduction was not found much significant.*

Chapter 5
CRFTS (Clustered and Nearest Neighbor Reservation based Fault Tolerant Scheduling)

Cloud systems supply different kinds of on-demand services in accordance with client needs. As the landscape of cloud computing undergoes continuous development, there is a growing imperative for effective utilization of resources, task scheduling, and fault tolerance mechanisms. To decrease the users' task execution time (shorten the makespan) with reduced operational expenses, to improve the distribution of load, and to boost utilization of resources, proper mapping of user tasks to the available VMs is necessary. This study introduces a unique perspective in tackling these challenges by implementing inventive scheduling strategies along with robust fault tolerance mechanisms in cloud environments.

This chapter introduces the Clustering and Reservation Fault-tolerant Scheduling (CRFTS), which maximizes the system reliability while making it fault-tolerant and optimizing other Quality of Service (QoS) parameters, such as Makespan, Average Resource Utilization, and Reliability. The study optimizes the allocation of tasks to improve the utilization of resources and reduce the time required for their completion. At the same time, the reservation-based fault tolerance framework is presented, emphasizing reactive strategies, thus ensuring continuous service delivery throughout its execution without any interruption. The effectiveness of the suggested model is illustrated through simulations and empirical analyses, highlighting enhancements in QoS parameters while comparing with HEFT [132], E-HEFT [133], and the latest LB-HEFT [134], FTSA-1 [135], and DBSA [56] for various cases/conditions over both tasks and VMs.

The proposed CRFTS involves a novel VM allocation strategy, i.e., a clustered allocation strategy. It initially sorts the tasks and VMs and then divides both tasks and VMs into three clusters namely: low, mid, and high clusters. This clustering makes the allocation more efficient by narrowing the domain of mapping and allocating each task to the most suitable VM. The clustering restricts the domain of both tasks and VMs and thereby prevents the task from getting mapped with the VM which is apparently not appropriate. Furthermore, the proposed model is enforced with an effective fault-tolerant algorithm based on the prior reservation of VMs. The proposed model estimates the AR slot for the tasks and reserves the VM for tasks to guarantee the task execution till completion. The alternative VM is selected based on the previous load of the VM and the clustering approach. While evaluating

the model, CRFTS was evaluated based on parameters like Reliability, Makespan, and Average Resource Utilization on varying the number of tasks and number of VMs.

Some of the recent related fault-tolerant models like HEFT (Heterogeneous Earliest Finish Time), FTSA-1 (Fault Tolerant Scheduling Algorithm), and DBSA (Deadline Based Scheduling Algorithm), etc. were proposed in [38], [40], [45] respectively undoubtedly made noteworthy contributions. However, it is imperative to acknowledge their inherent limitations, creating a compelling need for targeted interventions to propel the field toward further advancements. Among all these proposed approaches, the reliability of DBSA was seen to be the most efficient. An innovative hybrid checkpointing and rollback recovery mechanism is also stated by the author in [25]. In addition, the author claims that the main requirement of today's dispersed environment is the optimized utilization of resources. While the current fault-tolerant methods compromise makespan which eventually results in increased task execution time. Moreover, the diverged environment raises the challenging concern regarding the total reliability of the system while effectively addressing the fault situations. In certain existing fault-tolerant approaches, inefficiencies in resource utilization may arise, resulting in performance degradation and heightened effective rates. This constraint holds particular significance in cloud environments, where the imperative of efficient resource allocation is fundamental in achieving high resource utilization and maintaining optimal service delivery. The cloud may undoubtedly introduce diverged fault scenarios because of its dynamic nature and support of diverse applications. Addressing them becomes a principal requirement for ensuring a seamless and reliable user experience and thereby meeting service level expectations. After the extensive literature review, it was noted that the work on some scheduling parameters like makespan and average utilization of the resources can be addressed more. Here, we are motivated to propose the novel fault-tolerant scheduling model, CRFTS that uses clustering-based allocation and reservation-based fault tolerance. The CRFTS is compared with HEFT, FTSA-1, and DBSA and was evaluated for Reliability and Makespan. This model has improved reliability and surpasses all the compared approaches. Furthermore, the suggested model was compared with three other models i.e., HEFT, E-HEFT (Enhancement of Heterogeneous Earliest Finish Time) [27], and the most recent LB-HEFT (Load balancing- Heterogeneous Earliest Finish Time) [136] for Average Resource Utilization and Makespan. The outcomes demonstrate that CRFTS outperforms the compared state-of-art. Some of the enhancements of the contributed work are listed below:

- Progress in Cloud Reliability

- Novel Task Scheduling
- Reduction of Service Interruptions and Makespan
- Efficient Utilisation of Resources
- Fault-administration
- Scalable and Adaptable Fault Handling.

Additionally, the primary contributions of this chapter can be encapsulated in the following manner:

- In order to find some ideal and optimized task scheduling of user tasks on the accessible VMs, we first introduced the problem of mapping between tasks and VMs.
- Secondly, to handle dynamically failed or affected tasks, we propose a reservation-based fault tolerance and migrate the affected task to some healthy VM.
- To address the two identified challenges, we introduced the CRFTS approach, incorporating two methodologies. Initially, it utilizes the clustered technique for scheduling, and subsequently, for managing faults, the model incorporates the reservation technique.

We established a system model to assess the effectiveness and efficacy of the proposed CRFTS approach by conducting comparisons with five other related approaches. The evaluation considered parameters such as Reliability, Makespan, and Average Resource Utilization during the execution of a set of parallel applications on varying tasks and VM heterogeneity.

5.1. The Proposed Model

In this section, we introduced the fault-tolerant-based scheduling algorithm. The incoming tasks are taken as set T= $\{t_1, t_2, t_3 \ldots t_n\}$ and the VMs are taken as set V= $\{v_1, v_2, v_3 \ldots v_k\}$. This section explains the proposed CRFTS model which uses a clustering approach for allocation and the advance reservation of VMs to handle the failure in a dynamic computationally intensive cloud system. The proposed cluster-based allocation technique can be represented as the bipartite graph between the task set and VM set as shown in Figure 5.2. The System Architecture and Problem Formulation are illustrated in the below subsection.

5.1.1. The System Architecture

The system architecture of LB-CRFTS is comprised of three main layers as shown in Figure 5.1, i.e., Application, Middleware, and Host/VM Layer. Besides, the AR Module is an

important component of system architecture. The application layer and Host/VM layer present and involve the assumptions related to the cloud computing infrastructure included in the model. The considered cloud computing infrastructure involves:

Description of VMs: Describe the VMs used in the model comprising their capacities within the system. The architecture also discusses how VMs are provisioned and administered.

Operational task flow: This defines how tasks are received and managed within the cloud infrastructure, from their arrival till completion. The process includes Task and VM sorting, Task and VM Clustering, Scheduling, Fault handling, and interaction between VMs and other components.

Application Layer: This layer is responsible for the following operations:

- *Interface for the User:* This layer serves as the principal interface between the user and the system. It is accountable for allowing and supervising user tasks and their requirements.
- *Task Management:* After obtaining tasks from the user, task sorting will be performed based on certain criteria of task size.
- *Task Clustering:* The sorted tasks are then processed by the task clustering component within the Application Layer. The task clustering component of the Application Layer is responsible for creating three task clusters, i.e., low, mid, and high clusters.

Host/VM Layer: Likewise, the VMs are sorted and clustered in three clusters, i.e., low, mid, and high clusters in the Host/VM Layer using the VM clustering component.

Middleware: This component is comprised of four main sub-components.

- *Cluster Matching* is responsible for finding the corresponding VM cluster with respect to the task cluster.
- After identifying the corresponding VM cluster, the suitable VM for the task is discovered by *VM Discovery*.
- *VM Mapping* is responsible for matching the selected VM to the task or user requirements.
- If the match is found, the *Schedule Producer* produces the generated schedule for the user tasks to the Application Layer.

Furthermore, the Schedule Producer also communicates the generated schedule to the AR module.

AR Module: When there is a fault or malfunctioning, the AR module kicks into gear, computing the AR slot for the unsuccessful task using its *Time Manager* component. Based on the computed AR slot, the *VM Matching* component identifies the reserved VM for the

affected task and ultimately the AR Module produces the reservation to the Application Layer through its *Reservation Producer* component.

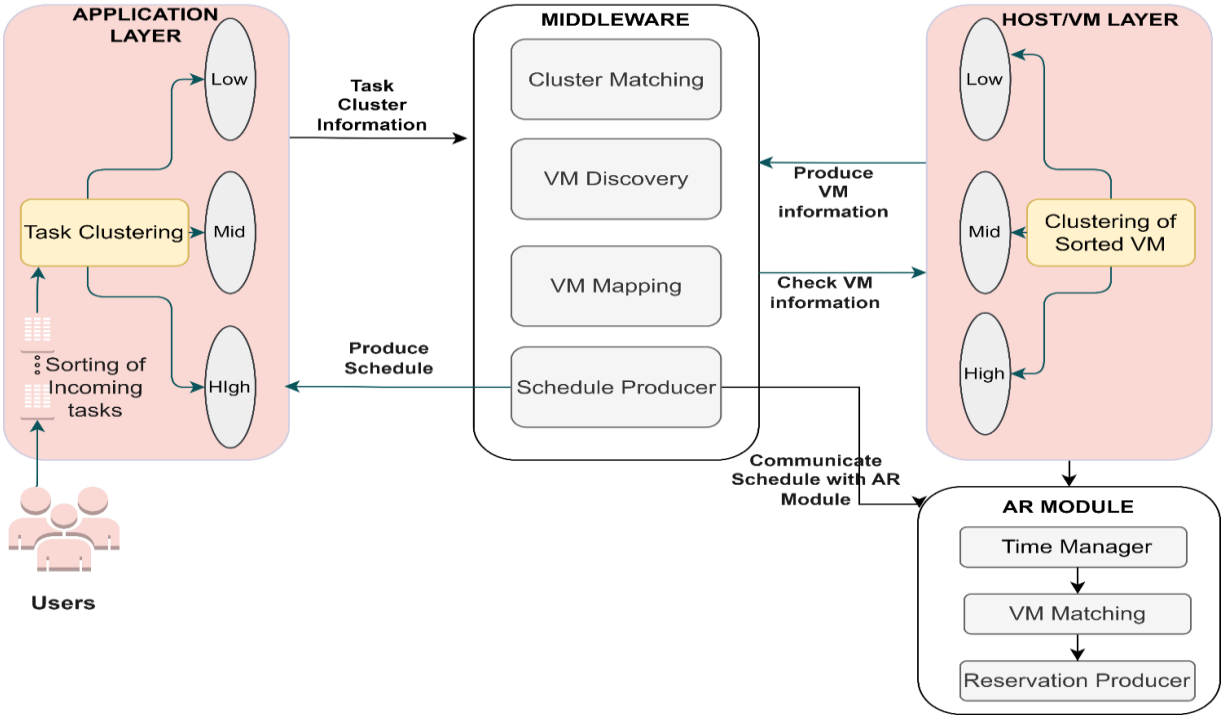


Figure 5.1: System Architecture of CRFTS

5.1.2. System Modelling

In the proposed system, the incoming tasks are taken as set $T=\{t_1, t_2, t_3, \ldots t_n\}$ AND $|T|=n$ are regarded as having varying or identical task weights (w). Some of the additional characteristics of the task are as follows:

- Each task has a time when it starts its execution on any VM, i.e., ST_{ij} (Start Time).
- The completion time of t_i on v_j is termed as FT_{ij} (Finish Time).
- Execution time of t_i i.e., $E(t_i,v_j)$ is the time taken by t_i to complete on v_j.
- Besides, the scheduling is non-preemptive. The task is preempted only in case the corresponding VM has a fault or malfunction.

Likewise, the available VMs with varied speed (s) are represented as a set $V=[v_1, v_2, v_3, \ldots v_j]$ AND $|V|=j$. Moreover, each t_i in T and v_j in V belongs to one of the three clusters i.e., low, medium, and high clusters. The cluster of t_i and v_j depends on the weight and speed of t_i and v_j respectively. Further, the model considers the VM with the following characteristics.

- J is the number of heterogeneous computing VMs that will participate in the mapping of the independently arriving tasks.

- Only compute-intensive tasks are ideally suited for the VMs.
- Each VM has its previous load which is termed the ready time of the VM. Initially, the ready time of all VMs is taken as zero which implies that the VM has not executed any task yet. (*This factor measures the machine's prior workload*).

The properties of the proposed mapping are listed below:

- The task is mapped to only one VM at a time.
- The VM is mapped with more than one t.
- Each mapping is performed in between the corresponding task and VM clusters.

Further, the mapping between set T and set V is considered as many-to-one mapping in a bi-partite graph of T and V as shown in Figure 5.2.

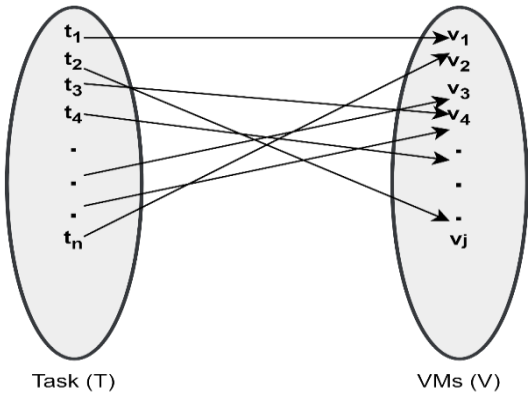


Figure 5.2: Many-to-One Mapping of Tasks and VMs

Besides, the system model consists of two main algorithms—Task allocation, and Fault tolerance.

5.1.3. Problem Formulation

The problem is modelled by effectively mapping (M) the arriving tasks (T) and available VMs (V) with an optimized Makespan. The solution for an efficient "M" is to create an allocation schedule that directs the arrival of tasks to be submitted for processing on VM in a manner that optimizes the optimizing criterion, i.e., OC(M) where "M" is mapping for generating the schedule. The mapping (M) is represented as follows:

$$M = T \rightarrow V$$

TET_j is the total execution time of any VM at any point (p) in time and is calculated as follows:

$$TET_{j,p} = RT_{j,p-1} \quad (1)$$

From here we will define the equations with respect to the given point (p) in time. Because tasks are of different size/weights (W) and VMs possess different speeds (S), therefore $E(t_i, v_j)$ is the execution time of task (t_i) on VM (v_j) and can be calculated as follows:

$$E(t_i, v_j) = \frac{W(t_i)}{S(v_j)} \quad (2)$$

$ST_{ij,p}$ is the Start time of t_i at any point in time and is the time when t_i is assigned to v_j and computed as:

$$ST_{ij,p} = RT_{j,p-1} \quad (3)$$

In the demonstration, ST_{ij} can also be calculated as shown in the following equation:

$$ST_{ij,p} = TET_{j,p-1} \quad (4)$$

Furthermore, FT_{ij} is the time when t_i completes its execution on v_j and is computed as:

$$FT_{ij} = ST_{ij} + E(t_i, v_j) \quad (5)$$

After finishing the current task, the FT_{ij} will become the ST of the next task and is defined as:

$$ST_{i+1,j} = FT_{ij} \quad (6)$$

Besides, after completion of each t_i, RT and TET are updated as shown below:

$$RT_{j,p} = TET_{j,p} = TET_{j,p-1} + E(t_i, v_j) \quad (7)$$

The procedural flow of the model is explained below:

5.1.3.1. Task Allocation

Task allocation starts with the sorting of both tasks and VMs in ascending order of their weight and speed, respectively. After sorting, the allocation is performed in two main phases as described below:

- *Task and VM Clustering:*

In this phase, three different clusters are created from both T and V based on their weight and speed respectively. Clusters are specified and shown in Figure 5.3:

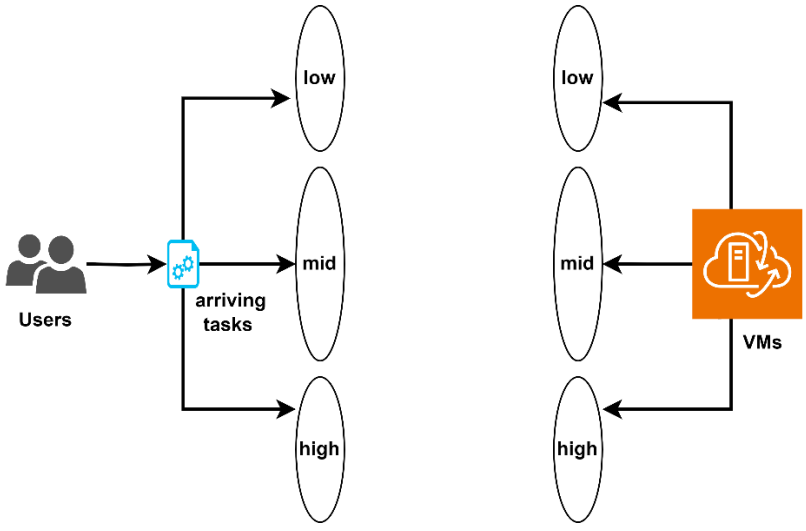


Figure 5.3: Task and VM Clustering

The average phenomenon is used for Task and VM Clustering. Where initially the global average of size and speed for both the task and VM respectively is calculated in equations 8 and 9:

$$Global\ Average(tasks)\ = \frac{\sum_{i=1}^{n} W(t_i)}{n} \quad (8)$$

$$Global\ Average(VMs)\ = \frac{\sum_{i=1}^{j} S(v_i)}{j} \quad (9)$$

The global average separates the set into two halves (Left half and Right half) as shown in Figure 5.4. Similarly, the average of the index for the left half and the right half is calculated and named as left average and right average, respectively.

Further, the cluster ranges are as follows:

- Task Low Cluster [TC^l : 0 <= TC^l <= Left Average],
- Task Mid Cluster [TC^m : (Left Average +1 <= TC^m <= Right Average],
- Task High Cluster [TC^h : (Right Average + 1 < = TC^h < = onwards].

Similarly, VM clusters are specified as :

- VM Low Cluster [VC^l : 1 <= VC^l <= Left Average],
- VM Mid Cluster [VC^m : Left Average +1 <= VC^m <= Right Average)],
- VM High Cluster [VC^h : (Right Average +1 < = VC^h < = onwards].

The main axiom for the classification of VM is that the VMs in high clusters take less time to execute a particular task than subsequent cluster VMs. The clustering of tasks and VMs restricts the domain of allocation and thereby limits the weight and speed variations in the possible mapping. An easy way of understanding the clustering phenomenon is presented in Figure 5.4.

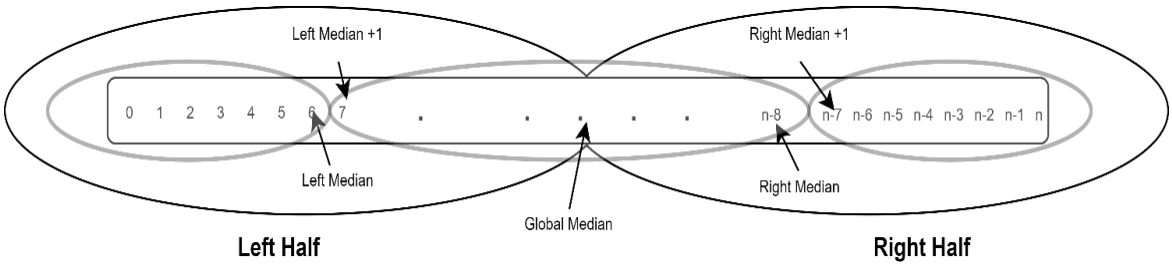


Figure 5.4: Clustering Phenomenon

- *Task to VM Mapping:*

After creating task and VM clusters in the clustering phase, the one-to-one mapping is performed between the corresponding T and V clusters unless all the VMs get their first task to execute. The cluster-wise mapping is performed in two steps:

- Allocate the VM to the tasks in its corresponding cluster in order, until RT (any v_j in the corresponding cluster = 0).
- If the RT of all VMs is greater than 0, allocate the task to the VM in the corresponding cluster with the least RT.

The same is explained and presented in Figure 5.5.

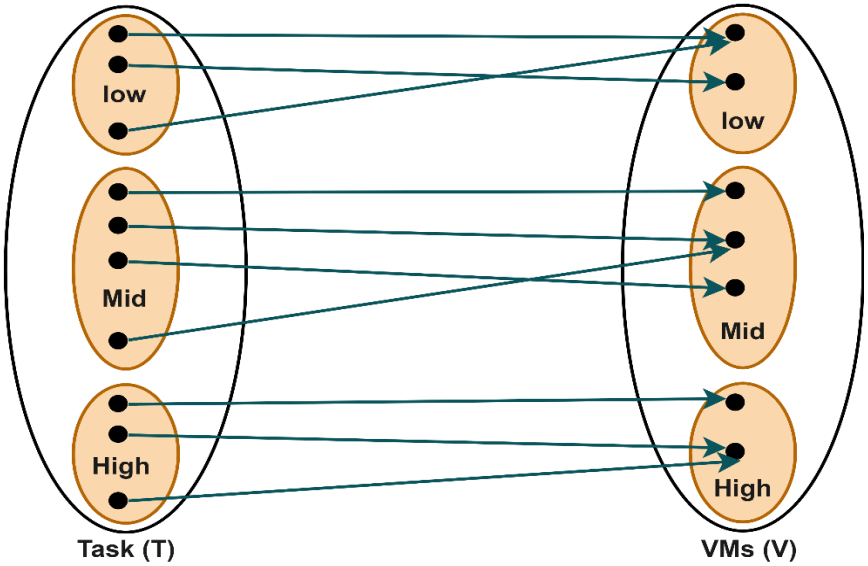


Figure 5.5: Cluster-Wise Mapping

There could be a case to allocate the tasks immediately after sorting. Such a type of allocation is referred to as Cluster-less allocation. The proposed task allocation algorithm is presented in Algorithm 1.

Algorithm 1: Task Allocation Algorithm

Input: (T, RT= 0, S, V, T_size) /*Task and resource components

Output (Makespan, Average Resource Utilization, Reliability,) /*computed by eq. (13, 16, 21) respectively

Phase 1: Task Clustering(T)

1. Initialise the Task Clustering parameters:

Low Task cluster range, medium task cluster range, high task cluster range

2. Compute Global_Avg, Left_Avg, and Right_Avg

For all tasks t_i in T

Int Global_ Avg, left_ Avg, Right_Avg

Do

Global_ Avg = Average of task size

left_ Avg = Average of the left half

Right_ Avg = Average of the right half

low task cluster range = 0 index to left_ Avg

medium task cluster range = left_ Avg +1 to right_ Avg

high task cluster range = right_ Avg +1 onwards

End for

Phase 2: VM Clustering(V)

3. Initialise the VM Clustering parameters:

Low VM cluster range, medium VM cluster range, high VM cluster range

4. Compute Global_ Avg, Left_ Avg, and Right_ Avg

For all v_j in V

Int Global_ Avg, left_ Avg, Right_ Avg

Do

Global_ Avg = Average of VM Speed

left_ Avg = Average of the left half

Right_ Avg = Average of the right half

low VM cluster range = 0 index to left_ Avg

medium VM cluster range = left_ Avg +1 to right_ Avg

high VM cluster range = right_ Avg +1 onwards

End for

Phase 3: Task and VM Mapping

Input: (RT, Task and VM Clusters)

Output: Task to VM Mapping

5. **For** all clusters in Clusters

For all this in the cluster

Allocated [i] = false

If Allocated [i]= false

While ∀ v_js, (RT (v_j) = 0))

Do

Allocate t_i to VM in the order

Allocated [i]= True

Allocate tasks to the VM with the least RT

Allocated [i]= True

End for

End for

The RT_j and TET_j are dynamically updated as in eq. 7 after the completion of each task. This implies that after a task gets completed on the VM with the recent least RT, the RT for that VM is recalculated to consider and reflect the additional load. The sorting of both VMs and Tasks before allocation and iterative updating of RT facilitates more adequate load distribution across all VMs.

5.1.3.2. Fault Tolerance

In this phase, VMs are monitored for faults in that case, advance reservation is enabled and an alternative VM is provided for the affected task. The model supports fault tolerance by resource reservation technique to offer a backup VM for the impacted task in the event of VM failure. Eq. (10) given below is used to determine the AR slot:

$$AR_{ij} = FT_{ij} - ST_{ij} \tag{10}$$

NOT expected_performance_metrics (vj) function is operated for discovering failed VMs and unexecuted tasks as discussed in Chapter 4.

If any VM leaves the system or fails at any point in time, and an alternative VM is not there, the associated tasks will suffer premature termination. Let's suppose f VMs are failing and these 'f' VMs are taken as the separate set of failed VMs (V_f) and are defined as $V_f = \{v_f : v_f \in V \text{ and } |V_f| = f\}$. This failing of VMs will affect the corresponding tasks and lead to the unsuccessful execution of these tasks. Let 'u' the number of unsuccessful tasks. Similarly, these 'u' unsuccessful tasks are taken as a separate set of unsuccessful tasks (T_u) and are defined as $T_u = \{t_u: t_u \in T \text{ and } |T_u| = u \ \& \ u<=|T|\}$. Now, these unexecuted tasks need to be reassigned to some alternative VM to complete their execution which is done by advance reservation technique. To ensure the uninterrupted operation of T_u, the task set t_u must now be redistributed from v_f to other relevant healthy v_h in V_h.

Where, V_h: $v_h \mid v_h \in V \text{ AND } v_h \notin V_f$.

The model reserves the nearest neighboring VM of the corresponding failed task as an alternative VM.

After every redistribution of task t_u from T_u to v_j in V, the TET_j is updated as shown below:

$$TET_{j,p} = TET_{j,p-1} - E(t_u, v_f) \tag{11}$$

$$TET_{j,p} = TET_{j,p-1} + E(t_u, v_j) \; where \; v_j \in V \; AND \; v_j \notin V_f \tag{12}$$

All tasks in T_u are migrated to another VM (V_j) from (V_f) for the calculated AR slot. The initial sorting helps the model to achieve a one-to-one suitable order. Thereafter, clustering before allocation restricts the domain of allocation which helps the model to assign the most suitable VM to the task. Lastly, the task from each cluster is allocated to the corresponding

VM clusters only which helps the model to avoid under and overutilization of the VM up to a certain degree. However, still there are chances of over and underutilization of VMs which will be handled by integrating the proposed approach with an efficient load-balancing technique that is in consideration for future work which continuously monitors the under and over utilising VMs and shifts the load between them.

The proposed fault tolerance algorithm is presented in Algorithm 2.

Algorithm 2: Nearest Neighboring Fault-tolerance Algorithm

Load ARM (t_i, v_j, AR, Status[T]) /*Advance Reservation Matrix initialize all slots as zero

For all t_is in T

Compute ST_{ij} and FT_{ij} using eq. 3 and 5

Compute AR slot using eq. 10

Identify V_f and T_u as per Algorithm 2 in Chapter 4

Identify healthy VMs

$V_h = v_i | v_i \in V$ AND $v_i \notin V_f$

Reserve nearest neighbor VM

for each t_u in T_u:

while Status(t_u) = 1:

for each $t_f.v_i$ in V:

for each $t_f.v_i$ in V:

$R = t_u.v_{i+1}$

$L = t_u.v_{i-1}$

if (R, L $\in V_h$ && RT(R) < RT(L)&& R $\in V_h$):

Select R (right neighbor) as an alternative VM for t_u for AR_{ij}

// Reserve the time slot for the selected task

else if (R, L $\in V_h$ && RT(R) > RT(L)&& L $\in V_h$):

Select L (left neighbor) as an alternative VM for t_u for AR_{ij}

else if (L $\in$ V && R $\notin$ V&& L $\in V_h$) //First VM fails

Select L (left neighbor) as an alternative VM for t_u for AR_{ij}

else if (R $\in$ V && L $\notin$ V&& R $\in V_h$) //Last VM fails

Select R (right neighbor) as an alternative VM for t_u for AR_{ij}

else R++, L- -

endfor

endfor

6 Update TET_j using eq. 11 and 12

7 Update ARM with parameters (t_i, v_j, AR, Status==1 means VM reserved)

End for

Figure 5.6 presents an apparent flowing depiction of the model in a flowchart.

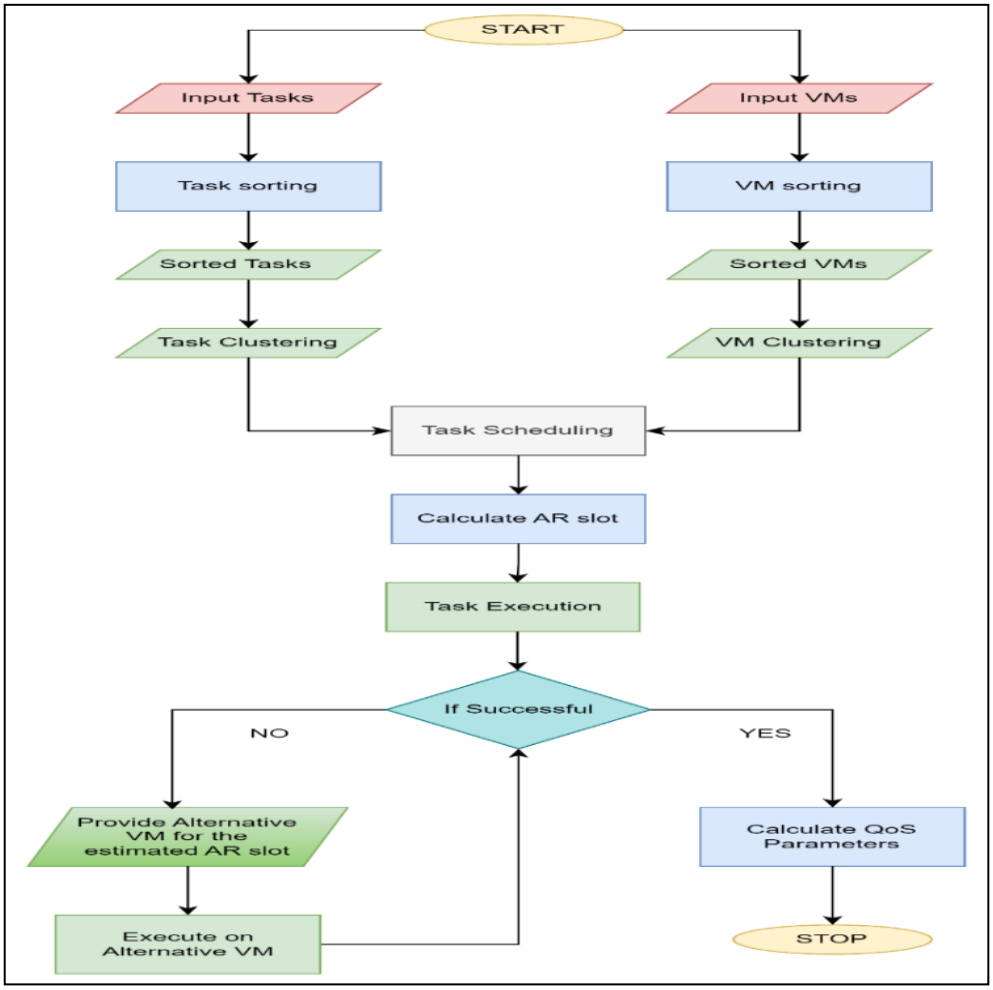


Figure 5.6: Flowchart of CRFTS

5.2. Illustrative Example

This section presents the illustration of the model by comparing the proposed clustered allocations with the normal allocation without clustering. The comparison has been carried out by demonstrating the example in this section. Four VMs and an instance with nine separate independent activities or tasks have been taken as shown in Table 5.1 to

demonstrate the functionality of the proposed model. Here, MI stands for Million Instructions and MIPS for Million Instructions Per Second.

Table 5.1: Considered instance of tasks and VMs

Task (t_i)	Weight (t_i)	V_j	Speed(V_j)
t_1	120 MI		
t_2	260 MI		
t_3	480 MI	V_1	30 MIPS
t_4	86 MI	V_2	40 MIPS
t_5	100 MI	V_3	10 MIPS
t_6	220 MI	V_4	20 MIPS
t_7	450 MI		
t_8	280 MI		
t_9	350 MI		

Note: The ready time of all VMs is initially taken as zero

First, the allocation is done by only sorting the tasks and VMs without clustering. The example is demonstrated for both cluster-less allocation and clustered allocation to show how clustered allocation offers optimized QoS as compared to cluster-less allocation. The beginning and end of the execution of t_i on v_j are marked by the times called STij and FTij, respectively. Additionally, the E (t_i, v_j) is added to the ST_{ij} to compute FT_{ij}.

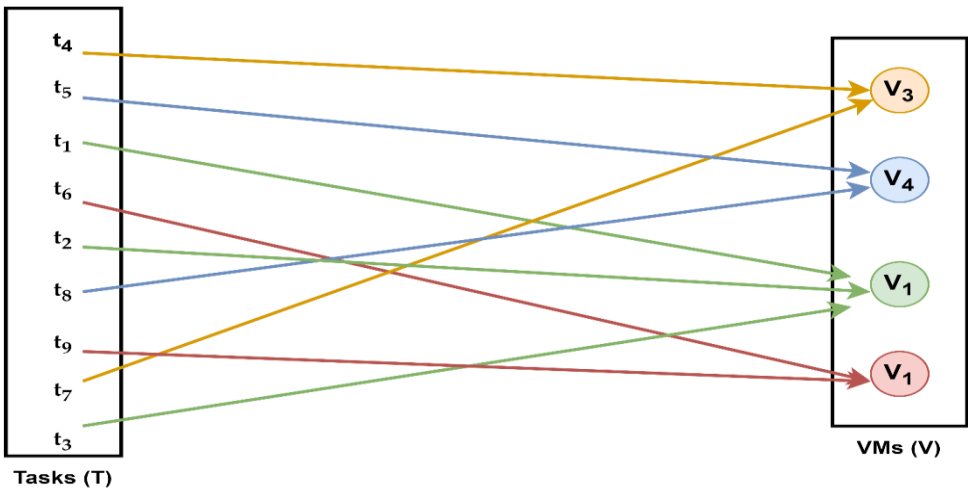


Figure 5.7: Mapping of Cluster-Less Allocation

5.2.1. Cluster-less Task Allocation

Initially, before the allocation, both tasks and VMs are sorted based on the weight of the tasks and the speed of the VMs. The undertaken instance after sorting is presented in Table

5.2 while the cluster-less mapping is shown in Figure 5.7. It shows the task to VM mapping by only the sorting of tasks and VMs without clustering them.

Table 5.2: An instance of tasks and VMs after sorting

Task (t_i)	**Weight (t_i)**	**V_j**	**Speed(V_j)**
t_4	86 MI		
t_5	100 MI		
t_1	120 MI	V_3	10 MIPS
t_6	220 MI	V_4	20 MIPS
t_2	260 MI	V_1	30 MIPS
t_8	280 MI	V_2	40 MIPS
t_9	350 MI		
t_7	450 MI		
t_3	480 MI		

After sorting, the allocation begins by mapping the tasks and VMs in the same sorting order until the RT of any VM is 0. Once the RT of all VMs becomes greater than 0, thereafter, tasks are allocated to the VM having the least RT. Here, t_4 is allocated to v_3, t_5 is allocated to v_4 so on. Here, the RT of each VM which is initially zero will become the ST of the corresponding task allocated to that VM. Therefore, ST_{43}, ST_{54}, ST_{11} , ST_{62} is zero. After t_4 , t_5 , t_1 and t_6 completes its execution on the assigned VMs, the FT_{43} will be calculated by adding the $E(t_4,v_3)$ to the ST_{43} and $E(t_4,v_3) = \frac{86}{10} = 8.6$. Therefore, FT_{43} will be 0 + 8.6 = 8.6. Similarly, FT_{54}, FT_{11}, and FT_{62} will be 5, 4, and 5.5 respectively as shown in Figure 5.8. Here, TET_j and RT_j will be updated and are taken as FT of the respective tasks. Now, the next task will be allocated to the VM having the least RT and FT_{ij} will become ST for t_{i+1} assigned to v_j. Hence, 8.6 will be ST_{73}, 5 will be ST_{84}, 4 will be ST_{21} and 5.5 will be ST_{92} as shown in Figure 5.8. Similarly, each task is allocated to the VM with the least RT and both RT and TET will be updated after every completion of the task. After all the tasks are completed the final TET of v_1, v_2, v_3, and v_4 will be 28.6, 14.25, 53.6, and 19 respectively as shown in Figure 5.8.

Once the execution of all tasks is completed, the Makespan and Average Resource Utilization are calculated from eq. 13 and 16 respectively.

Makespan will be calculated as Max (28.6, 14.25, 53.6, 19) as in eq. (13)

Makespan = 53.6

Average Resource Utilization will be calculated as:

$$UT = \frac{28.6 + 14.25 + 53.6 + 19}{4 * 53.6} = \frac{115.45}{214.4} = 53.8\% \quad \text{as in eq. (16)}$$

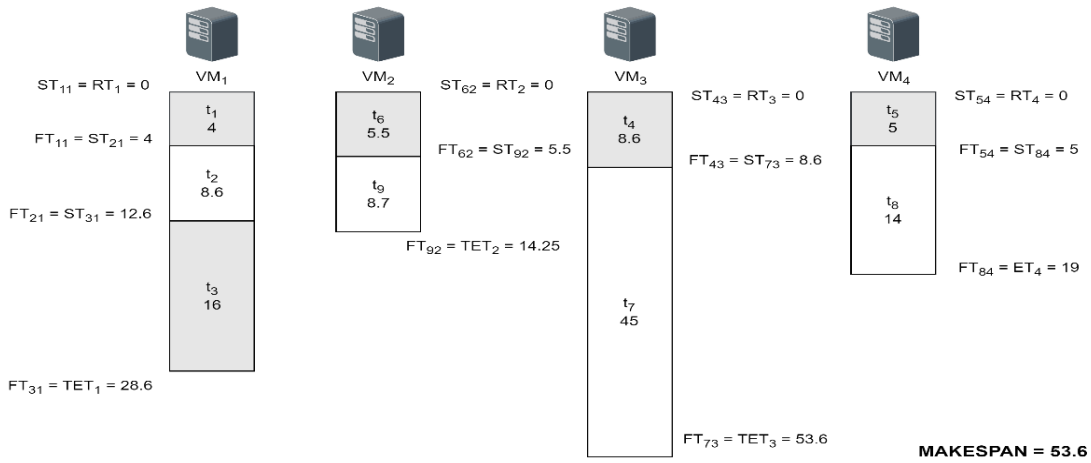


Figure 5.8: Demonstration of Cluster-Less Allocation

5.2.2. Proposed Clustered Task Allocation

The proposed clustering allocation is demonstrated in these phases.

5.2.2.1. Task and VM Clustering Phase

The clustered allocation narrows the domain of mapping by separating the tasks and VMs into three different clusters after sorting as explained in Section 3.3.1.1. Following the clustering, the task allocation is performed between the respective clusters only. The mapping of clustered allocation of the undertaken example is shown in Figure 5.9.

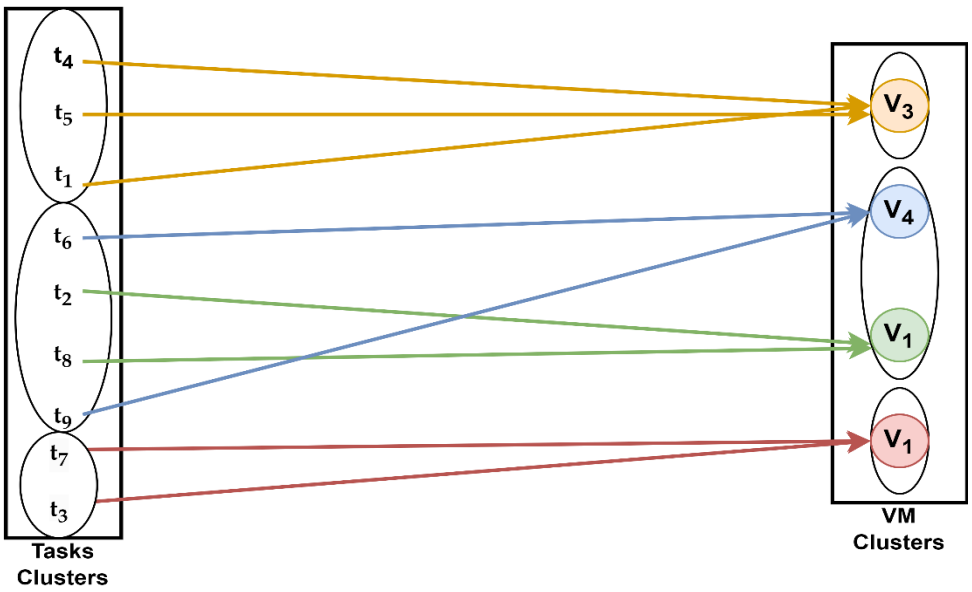


Figure 5.9: Mapping of Clustered Allocation

5.2.2.2. Task and VM Mapping Phase

After clustering, the allocation begins in the same sorting order until the RT of any VM is 0 but only in the respective clusters. Here, t_4 is allocated to v_3. Once, the RT of all VMs in the particular cluster becomes greater than 0, then the next task is allocated to the VM of the same cluster having the least RT. Likewise, t_5 and t_1 are allocated to v_3. Where, the ST_{43}, ST_{53}, and ST_{13} will be 0, 8.6, and 18.6 respectively. In the mid cluster, t_6 is allocated to v_4, and t_2 is allocated to v_1 both having RT equal to zero. Once t_6 and t_2 are completed, $RT(v_4) = 11$ and $RT(v_1) = 8.6$ then t_8 will be allocated to the VM with the least RT in the mid cluster only. i.e., v_1. After all the tasks are completed the final TET of v_1, v_2, v_3, and v_4 will be 17.93, 30, 30.6, and 28.5 respectively as shown in Figure 5.10.

In this example, the makespan will be calculated as Max (17.93, 27.62, 30.6, 28.5) as in eq. (13)

$$\text{Makespan} = 30.6$$

In this example, Average Resource Utilization will be calculated as:

$$UT = \frac{17.93 + 27.62 + 30.6 + 28.5}{4 * 30.6} = \frac{104.2}{122.4} = 84.9\% \text{ as in eq. (16)}$$

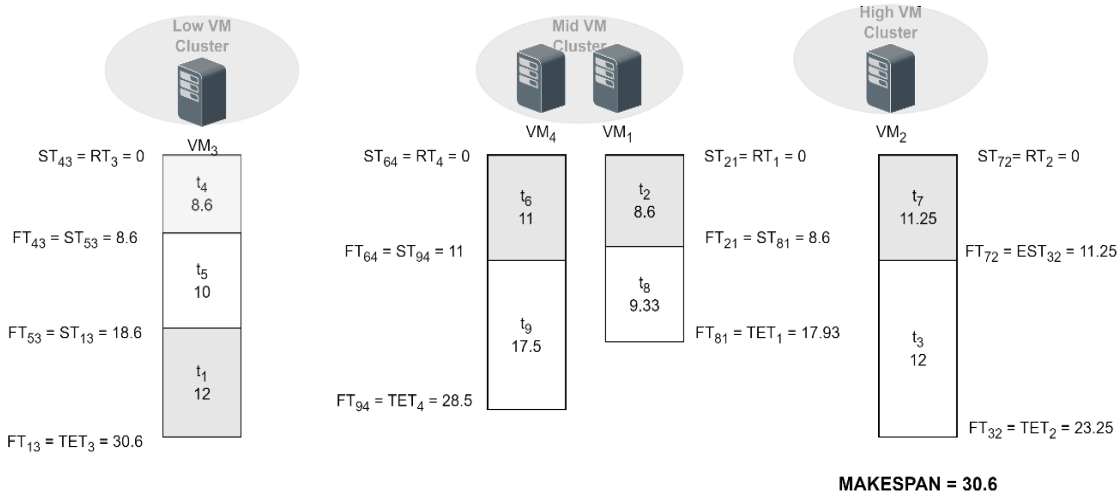


Figure 5.10: Demonstration of Clustered Allocation

On comparing the makespan and average utilization with the cluster-less allocation, the makespan was found to be 53.6 in the case of cluster-less allocation. However, in clustered allocation, the makespan was optimized to 30.6. Similarly, average resource utilization was found to be 53.8% in cluster-less allocation while the average utilization of clustered scheduling was found to be 84.9%. Therefore, the proposed clustered allocation strategy

was found to be more optimized concerning both makespan and resource utilization than that of cluster-less allocation.

5.3. Performance Metrics

After the successful task execution, the reliability of the system is checked. Additionally, the Makespan is taken as the highest or maximum among all TET_j and can be expressed as eq. (13) [137].

$$\text{Makespan} = \max(FT_{ij}), \forall V_j \tag{13}$$

Progress percentage of Makespan (Pp_m): It is the percentage of progress in makespan offered in proposed CRFTS over the other existing approaches i.e., HEFT, E-HEFT, and LB-HEFT approaches, and is calculated in eq. (14) [138].

$$Pp_m\ (\%) = \frac{M\ (Compared\ Approach) - M(Proposed\ Approach)}{M(Compared\ Approach)} * 100 \tag{14}$$

Average of (Pp_m): To calculate the deviation from the desired rate in percentage, divide the sum of every Pp_i for each tested VM by their respective tested numbers as shown in eq. (15) [139].

$$\text{Average of } (Pp_m) = \frac{\sum_{i=1}^{t} Ppm\ (each\ tested\ VM\ number)}{t} \tag{15}$$

The Average VM utilization of the system is calculated in eq. (16) as [140].

$$UT = \frac{\sum_{1}^{k}(FT - p(tf \in Vf, Vj \in Vf))}{k * \text{Makespan}}\ \forall Vj \tag{16}$$

Progress percentage of UT (Pp_u): It is the ratio defining the progress percentage of average utilization of the proposed CRFTS and other compared approaches and is calculated in eq. (17) as [141].

$$Pp_u\ (\%) = -\left(\frac{UT\ (Compared\ Approach) - UT(Proposed\ Approach)}{UT(Compared\ Approach)}\right) * 100 \tag{17}$$

Average of (Pp_m): To calculate the deviation from the desired rate in percentage, divide the sum of every Pp_u for each tested VM by their respective tested numbers as shown in eq. (18) [142].

$$\text{Average of } (Pp_u) = \frac{\sum_{i=1}^{t} Ppu\ (each\ tested\ VM\ number)}{t} \tag{18}$$

The reliability of the model is derived in terms of the Mean Time Between Failure (MTBF) and Failure rate (ʎ).

MTBF is the average time between two consecutive failures and is presented in eq. (19) [104]:

$$\text{MTBF} = \frac{Flowtime}{|T_u|} \tag{19}$$

Where, $|T_u|$ = Fault (%age) * $|T_n|$

Where, T_n and T_u are the total number of tasks and total unexecuted tasks, respectively, at any point in time

Failure rate (ʎ) is the reverse of MTBF and a measure of a system's effectiveness; the equation for ʎ is calculated as in eq. (20) [104] :

$$ʎ = \frac{1}{MTBF} \tag{20}$$

Reliability is calculated in terms of the percentage and is the ratio between the failed tasks and total tasks. Reliability is considered a maximization problem. It is defined as the percentage of tasks that are successfully completed out of all incoming tasks and is calculated in eq. (21):

$$\text{Reliability} = \frac{|T_n| - |T_u|}{|T_n|} * 100 \tag{21}$$

The Model readily provides the reserved VM as an alternative VM for the impacted task to ensure the successful execution of every task. Thereby, confirming maximum reliability in the event of more than 50% of faulty VMs.

5.4. Results and Observations

The results obtained from this work are detailed in this section. These results are derived from the algorithms outlined in the previous sections of this chapter and are presented in a structured manner to ensure clarity and coherence.

5.4.1. Experimental Setup

An analysis of the performance of the proposed CRFTS algorithm has been conducted in small and large task scales. The proposed model was compared based on Makespan and Average Resource Utilization while competing with HEFT, E-HEFT, and the latest LB-HEFT algorithm on a small task scale. The suggested CRFTS algorithm has been implemented in a heterogeneous environment evaluated on 5, 10, 20, and 40 VMs for varying the tasks from 25 to 1000. However, for large task scales, the task ranges are taken from 25 to 4096 and the model was cultivated with the existing HEFT, FTSA-1, and DBSA algorithms in terms of Makespan and Average Resource Utilization in the case of 100 VMs. Furthermore, the model was evaluated in terms of Reliability while comparing with HEFT,

FTSA-1, and DBSA on a small task scale taking the VM number as 20. Additionally, the size of task heterogeneity ranges from 1 to 100 MI while the speed of VM heterogeneity ranges from 1 to 10 MIPS. Table 5.3 presents the experimental environment for better understanding. The results show that the recommended model performs better than the comparison techniques.

Table 5.3: Experimental Environment and Parameters

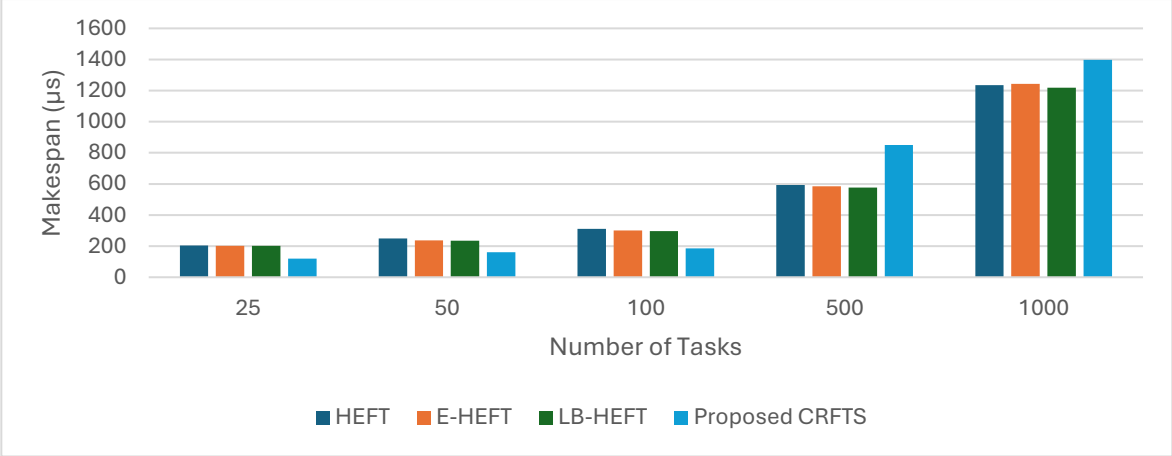
<table>
<tr><th>Parameters Optimized</th><th>Experimental Setups / Compared Approaches</th><th colspan="2">Input parameter</th><th>Range</th></tr>
<tr><td>Reliability</td><td>HEFT, DBSA, FTSA-1</td><td rowspan="2">Small Task Scale</td><td>Task Range</td><td>25 to 1000</td></tr>
<tr><td>Makespan, Average Resource Utilization</td><td>HEFT, E-HEFT, LB-HEFT</td><td>VM Range</td><td>5 to 40</td></tr>
<tr><td rowspan="2">Makespan, Average Resource Utilization</td><td rowspan="2">HEFT, DBSA, FTSA-1</td><td rowspan="2">Large Task Scale</td><td>Task Range</td><td>512 to 4096</td></tr>
<tr><td>VM Number</td><td>100</td></tr>
<tr><td colspan="2" rowspan="2">Task and VM Heterogeneity</td><td>Task Heterogeneity Range</td><td colspan="2">1 MI to 100 MI</td></tr>
<tr><td>VM Heterogeneity Range</td><td colspan="2">1 MIPS to 10 MIPS</td></tr>
</table>

5.4.2. Small Task Scale

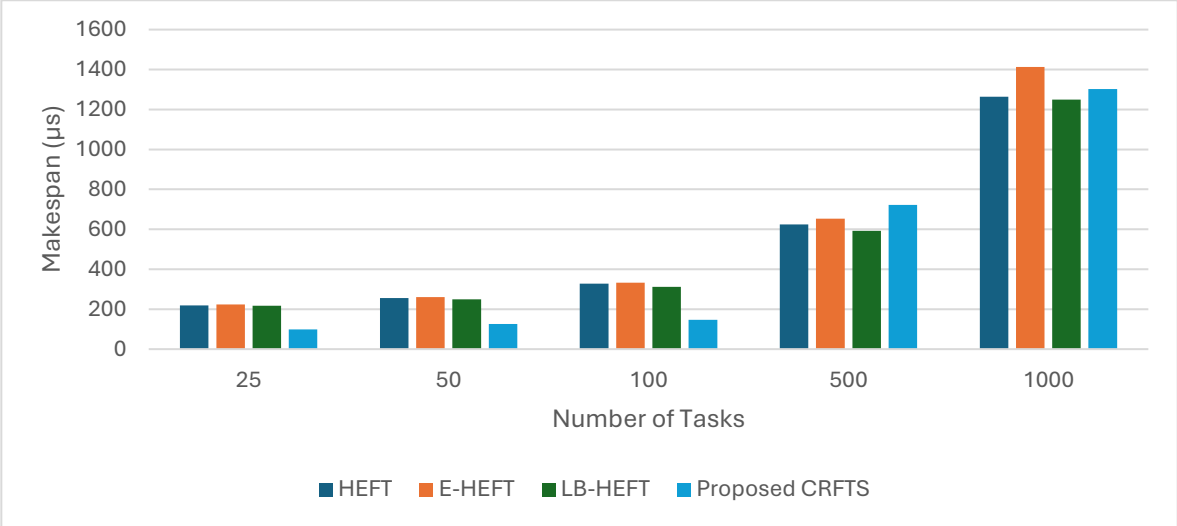
It is used to implement the suggested CRFTS algorithm in a heterogeneous environment evaluated on 5, 10, 20, and 40 VMs for varying the tasks from 25 to 1000.

5.4.2.1. Makespan

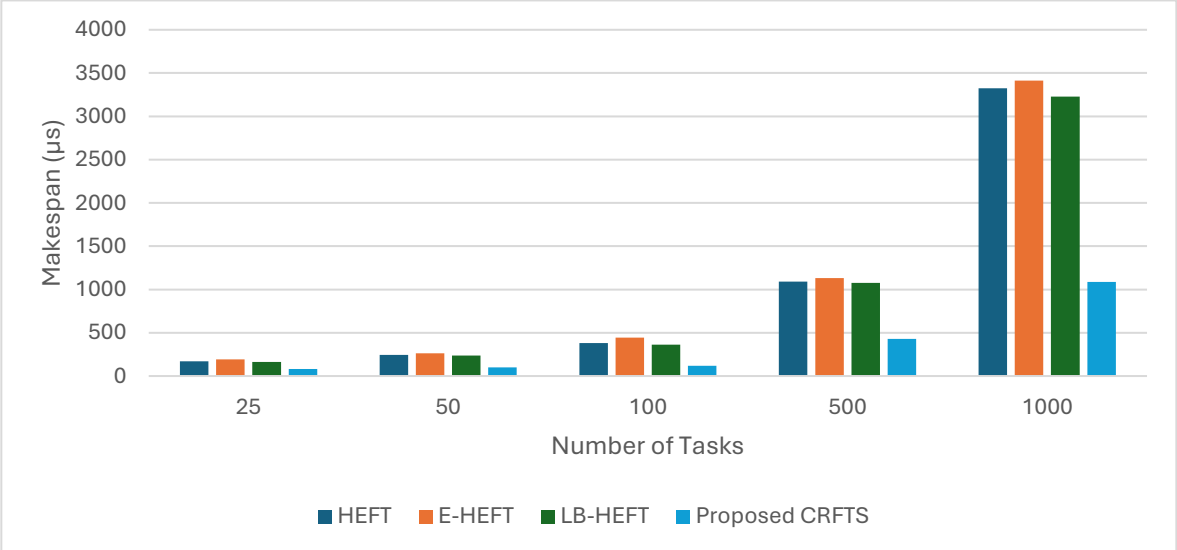
In Figure 5.11 (a-d), the implementation results of our suggested CRFTS are shown in comparison to HEFT, E-HEFT, and LB-HEFT concerning makespan using task instances of 25, 50,100, 500, and 1000 by considering 5, 10, 20, and 40 VMs, respectively. CRFTS surpasses the other compared algorithms when it comes to the time needed to finish the task for any number of VMs, according to the comparative results in Figure 5.11 (a-d). This is because, while mapping, the CRFTS utilizes a clustering approach that restricts the mapping domain of both tasks and VMs and thereby selects an appropriate VM for the task.



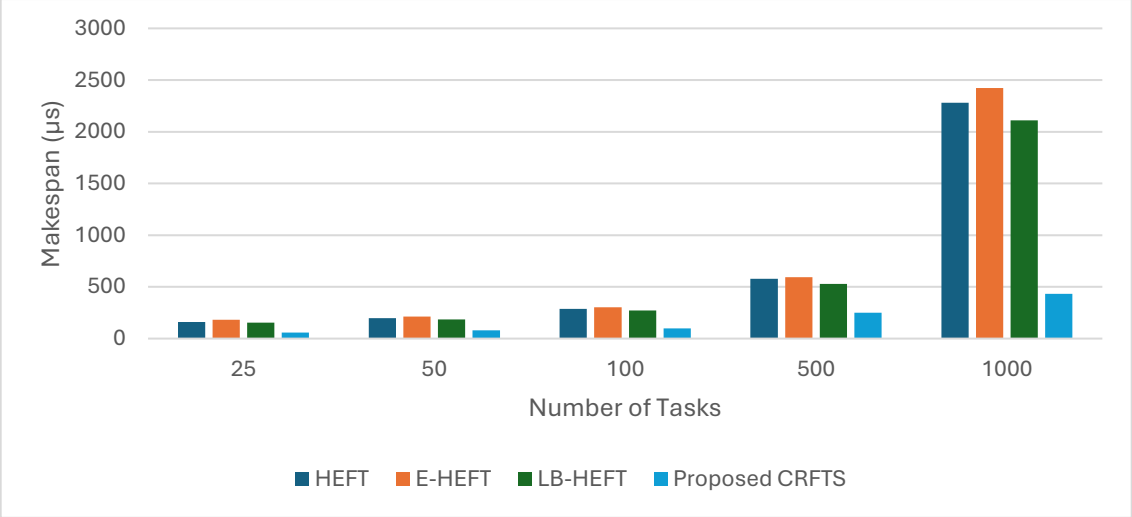
(a) 5 VMs



(b) 10 VMs



(c) 20 VMs



(d) 40 VMs

Figure 5.11: Makespan of Compared Approaches on 5, 10, 20, and 40 VMs (a-d)

Observations

Out of all methods, the E-HEFT algorithm offers the highest makespan. This may be due to the Matching Game theory that is used in E-HEFT to assign tasks to the appropriate VMs. However, the matching game theory selections take all tasks and VMs into consideration, which requires a lengthy selection process and reduces makespan. The results demonstrate that the proposed CRFTS algorithm outperforms the other algorithms. Specifically, as the quantity of VMs increases to 40, the laid-out algorithm outperforms the HEFT, E-HEFT, and LB-HEFT algorithms in terms of makespan for any number of tasks and VMs as seen in Figure 5.11 (c and d).

Pp_m over other compared approaches

Further, the results of comparing the proposed CRFTS algorithm's average enhancements (average progress percentage (Pp_m)) to the compared approaches with respect to the makespan in percentage computed as in eq. 14 and 15 employing 5, 10, 20, and 40 VMs using various task sizes with the HEFT, E-HEFT, and LB-HEFT existing algorithms are shown in Table 5.4 for comparison.

Table 5.4: Pp_m Related to the Proposed CRFTS

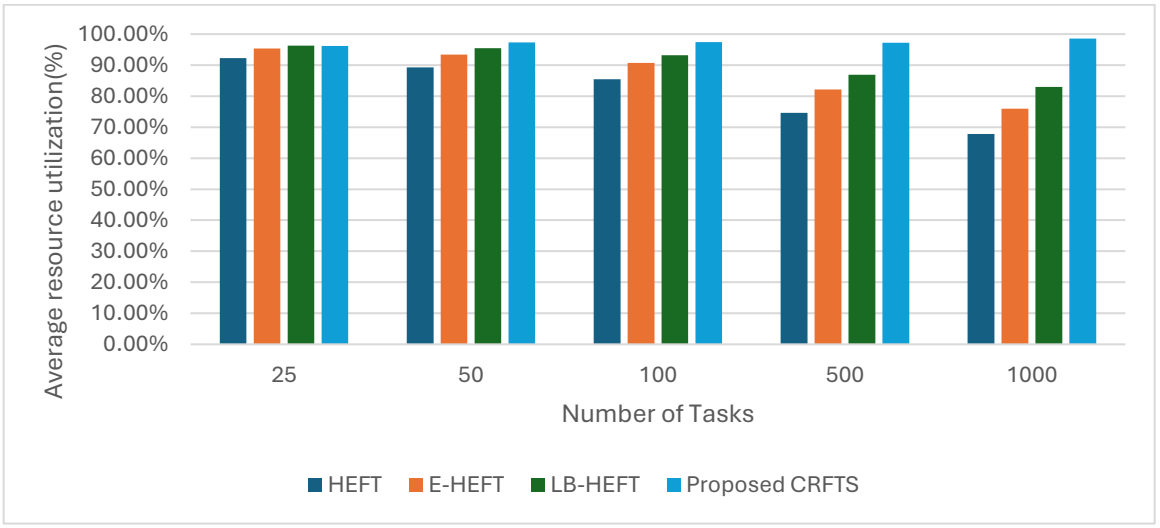
Tested VM numbers	Pp_m over HEFT	Pp_m over E-HEFT	Pp_m over LB-HEFT
5	12.06%	2.96%	9.42%
10	28.55%	32.2%	35%
20	61.72%	64.79%	60.50%
40	65.22%	67.42%	62.91%

The final average progress percentage shows that the optimization of the makespan is increasing as the number of tasks and VMs are increasing which is expected because as the

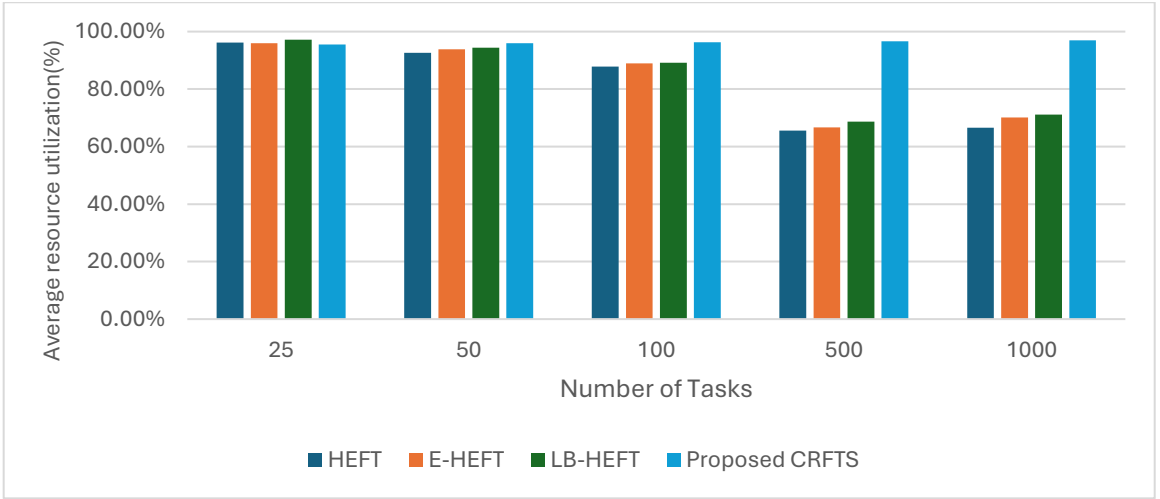
number of task and VMs are increasing, the clustering gets optimized thereby optimizing the makespan.

5.4.2.2. Average Resource Utilization

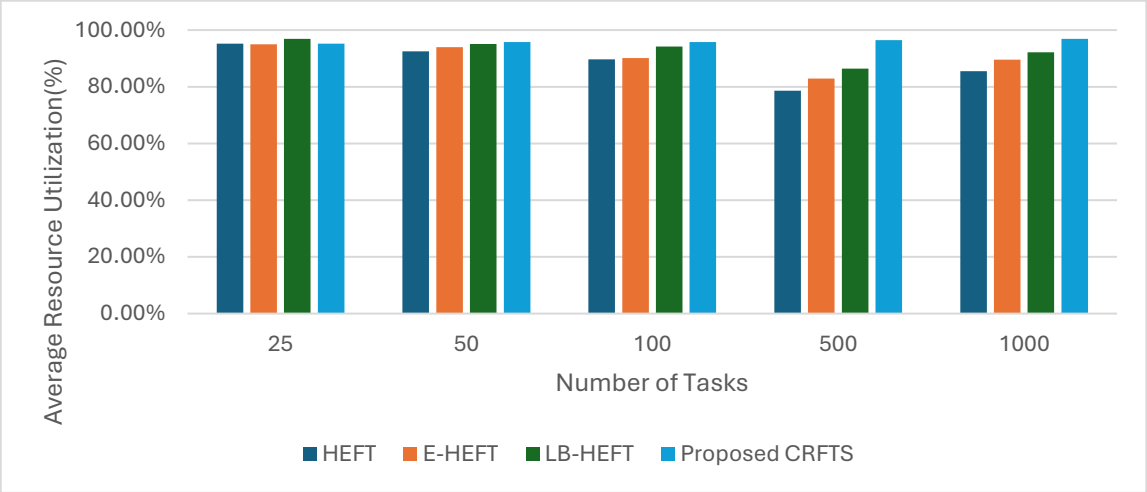
The performance of CRFTS has improved resource utilization per VM compared to other methods, as shown by the findings in Figure 5.12 (a-d). As the number of virtual computers rises, this tendency continues. However, as the number of tasks rises, the utilization shown by the CRFTS increases dramatically. Besides, with an increase in task number, the utilization of the other three models decreases. As can be seen from Figure 5.12 (b), the proposed CRFTS offers very optimal utilization, especially in the case of high task numbers and all other three comparing models perform very low utilization of resources. Moreover, all the comparing approaches work well for the low range of task numbers. However, as the task number increases, the utilization of the comparing models decreases. Unlikely the proposed model shows a constant growth in the utilization with an increase in task number.



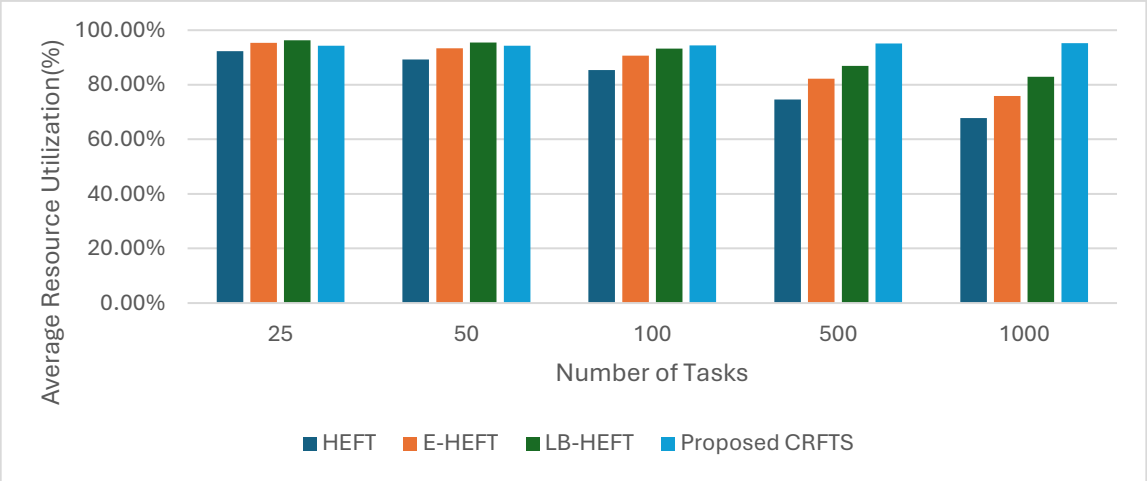
(a) 5 VMs



(b) 10 VMs



(c) 20VMs



(d) 40 VMs

Figure 5.12: Average Resource Utilization of Compared Approaches on 5, 10, 20, and 40 VM (a-d)

Observation

CRFTS was shown to be heavily utilized across all VM ranges, particularly for more than 100 tasks. Furthermore, the utilization of CRFTS is found to be approximately constant on varying tasks or VMs. This is because of sorting done before allocation. Moreover, CRFTS is found optimal than the compared approaches in all three considered parameters. Additionally, HEFT was shown to have the lowest average utilization among the other techniques. Moreover, LB-HEFT was found to be the second-best model in the case of utilization after CRFTS. However, when the number of tasks is less than 50, LB-HEFT performs somewhat better than CRFTS. As compared with HEFT, the proposed approach shows improvements in utilization from 3% to 33%. CRFTS shows improvements of 2% to 30% as compared to E-HEFT. This may be because, in this approach, the maximum number

of servings per machine must be considered while choosing the suitable machine for each job. While comparing CRFTS with LB-HEFT, the suggested model has shown improvements of 0% to 30%.

Pp_u over other compared approaches

Additionally, the results of comparing the proposed CRFTS algorithm's average enhancements (Pp_u) to the compared approaches with respect to the average resource utilization in percentage computed as in eq. 17 and 18 employing 5, 10, 20, and 40 VMs using various task sizes with the HEFT, E-HEFT, and LB-HEFT existing algorithms are shown in Table 5.5 for comparison.

Table 5.5: Pp_u related to the proposed CRFTS

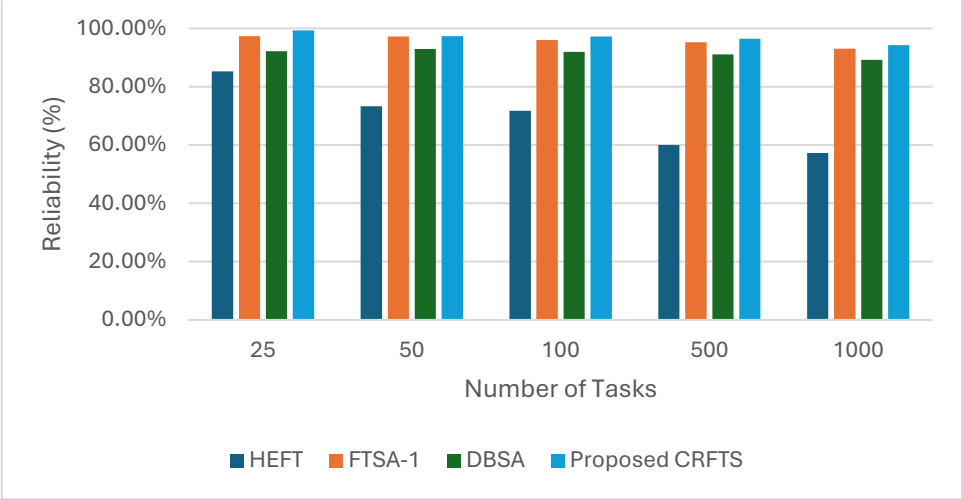
Tested VM numbers	Pp_u over HEFT	Pp_u over E-HEFT	Pp_u over LB-HEFT
5	20.53%	22.11%	7.38%
10	30.42%	18.86%	16.94%
20	9.26%	6.60%	4.37%
40	17.26%	10.01%	5.40%

The final average progress percentage shows that improvements in average resource utilization in HEFT are increasing as the number of VMs is increasing. However, the average progress percentage of CRFTS is significant in E-HEFT and LB-HEFT.

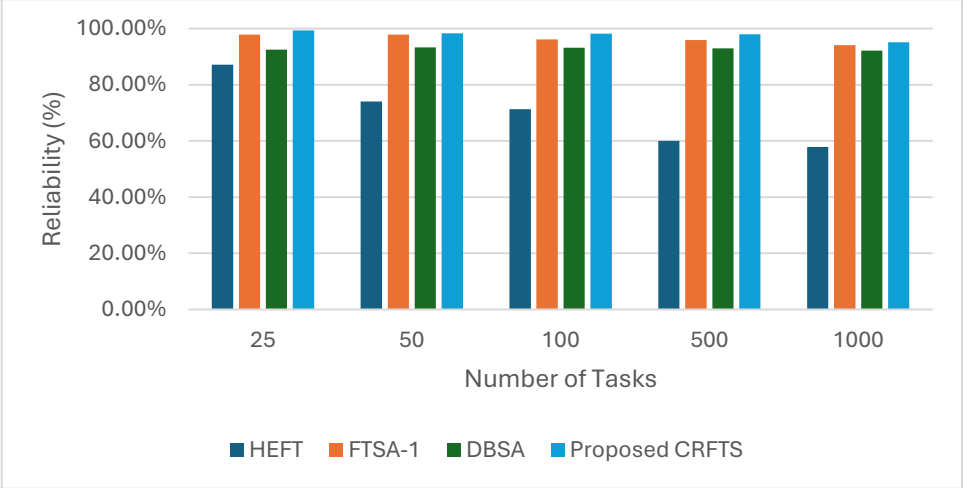
5.4.2.3. Reliability

The model is compared with other fault-tolerant models like HEFT, DBSA, and FTSA-1 and was evaluated based on reliability where the number of tasks varied from 25 to 1000, as can be seen in Figure 5.13. The performance of CRFTS has improved reliability as compared to other methods, as shown by the findings in Figure 5.13 (a-d). As the number of virtual computers rises, the reliability increases. However, as the number of tasks rises, the reliability in most of the cases decreases. As can be seen from Figure 5.13 (a-d), the proposed CRFTS offers increased reliability. However, when the number of VMs is 40, FTSA-1, DBSA and the proposed CRFTS performed optimally. Besides, the tendency of increased reliability offered by CRFTS continues as shown in Figure 5.13 (d).

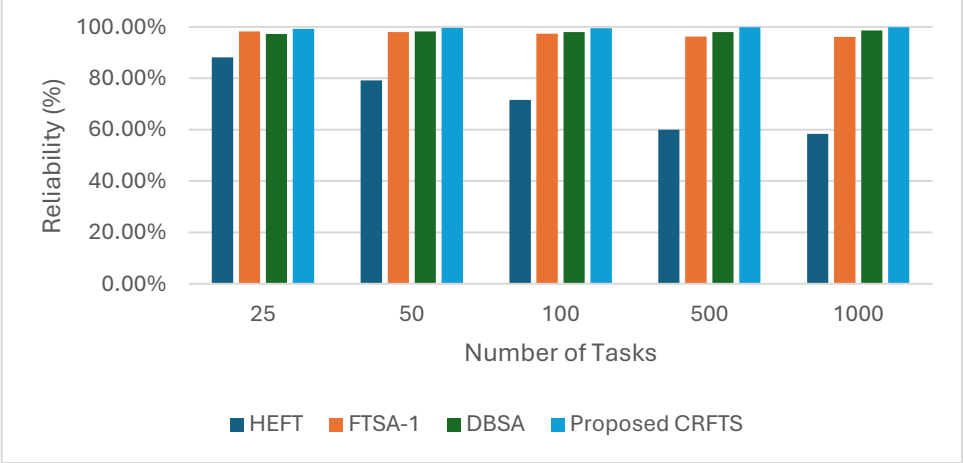
The minimum reliability of HEFT, FTSA-1, and DBSA is 57.24%, 93.10%, and 89.20% respectively. At the same time, the reliability of the proposed CRFTS ranges from 99.3% to 99.9% on varying both VM and task numbers.



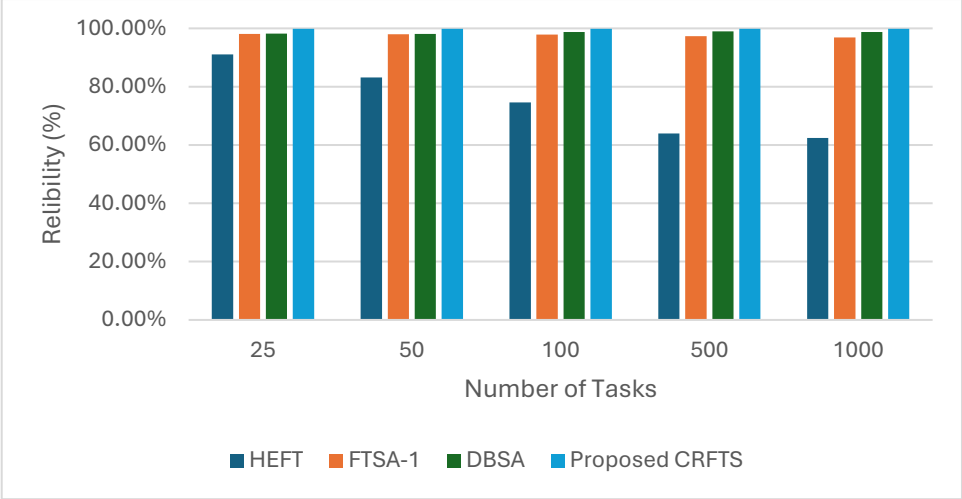
(a) 5 VMs



(b) 10 VMs



(c) 20 VMs



(d) 40 VMs

Figure 5.13: Reliability of Compared Approaches

Observations:

Additionally, with the increase in task number, the reliability of HEFT and FTSA-1 decreases while for DBSA and CRFTS, the reliability increases with the increase in task number as can be seen in Figure 5.13. However, CRFTS shows better reliability than that of DBSA. It was discovered that HEFT has extremely low reliability as compared to other approaches. It may be because HEFT does not offer any fault handling mechanism and thereby does not provide guaranteed task completion in case of faulty. Moreover, HEFT always assigns tasks to the processor with the Earliest Finish time (EFT) without taking load balancing across processors into account. Furthermore, the CRFTS shows an increase of 12.71% to 71.06%, 1.12% to 3.95%, and 1.31% to 6.54% than HEFT, FTSA-1, and DBSA, respectively. The improvements in the reliability of the proposed CRFTS are because of the advance reservation employed for handling the faults.

5.4.3. Large Task Scale

Here, the task ranges are taken from 25 to 4096 and the model is cultivated with the existing HEFT, FTSA-1, and DBSA algorithms in terms of Makespan and Average Resource Utilization for 100 VMs.

5.4.3.1. Makespan:

Furthermore, Figure 5.14 displays the implementation outcomes of our suggested CRFTS on comparison with HEFT, FTSA-1, and DBSA algorithms concerning makespan on varying the number of tasks from 512 to 4096. The CRFTS model improves the other comparing algorithms in terms of makespan to complete the workflow for any range of task numbers, according to the comparative results in Figure 5.14.

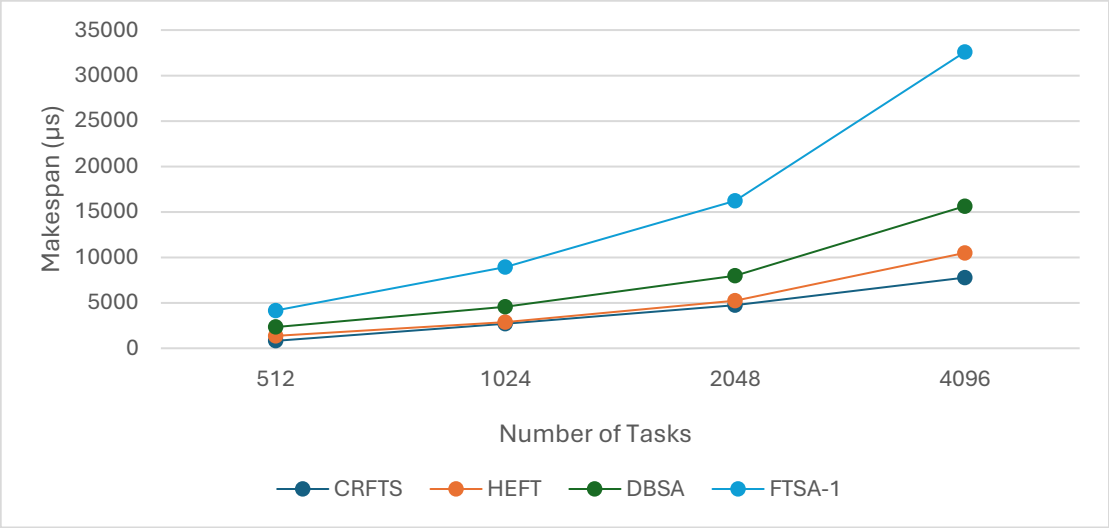


Figure 5.14: Makespan of Compared Approaches on 100 VMs

Observations:

Out of all the approaches, HEFT offers an optimized makespan after the proposed CRFTS. However, FTSA-1 provides the highest makespan among all the approaches. It may be because FTSA-1 is focusing more on reliability than that of makespan. According to the results related to makespan, CRFTS shows 40%, 6.32%, 9.358%, and 25.870% improvements over HEFT in 512, 1024, 2048, and 4096 tasks respectively. Comparing CRFTS with DBSA and FTSA-1, the proposed CRFTS shows more than 40% improvements in varying the number of jobs, respectively.

5.4.3.2. Average Resource Utilization:

Figure 5.15 displays the implementation outcomes of our suggested CRFTS in comparison with HEFT, FTSA-1, and DBSA algorithms concerning average resource utilization on varying the number of jobs from 512 to 4096. The CRFTS offers improved average resource utilization than other compared algorithms to complete the workflow for any range of task numbers, according to the comparative results in Figure 5.15.

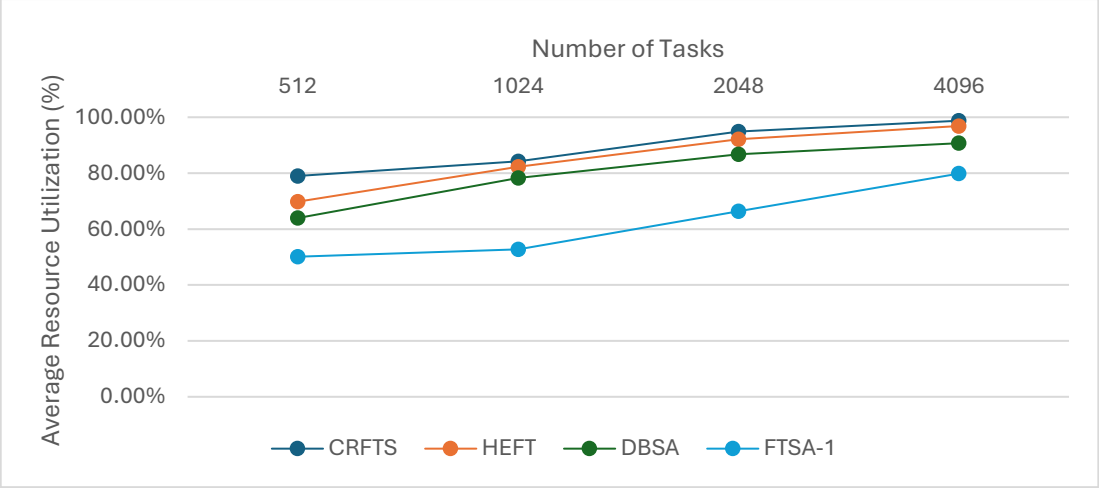


Figure 5.15: Average Resource Utilization of Compared Approaches on 100 VMs

Observations:

The average resource utilization provided by FTSA-1 was found to be very low compared to the other approaches. The utilization offered by HEFT is second optimal after CRFTS. CRFTS was found optimum as compared to other approaches. According to the results concerning to average resource utilization, CRFTS shows 14.38624%, 2.415644%, 3.169663%, and 2.083104% improvements over HEFT, 36.55696%, 37.33824%, 30.07273%, and 19.15755% improvements DBSA and 19.05063%, 7.11148%, 8.54854%, and 8.130822% improvements over FTSA-1 in 512, 1024, 2048, and 4096 number of tasks respectively.

5.5. Summary in Context

To get around the overheads of the compared approaches, the work in this article introduces the novel clustering approach for task scheduling namely CRFTS. The proposed CRFTS allocates the tasks in such a way that makespan is minimized and utilization is maximized. Out of all compared approaches, the optimal makespan is provided by the proposed CRFTS because of the clustering approach used for task allocation. However, the longest makespan is provided by the E-HEFT method. This is because each task is mapped by the E-HEFT algorithm table using the optimal VM rules over matching game theory. Additionally, CRFTS shows the increased resource utilization as the number of tasks is increasing. Conversely, the utilization of all other compared approaches decreases as an increase in the number of tasks. On the other hand, the LB-HEFT method performs better than HEFT in terms of resource utilization for considered VMs, as shown by the comparative findings in Fig. 6 (a-d). This is because, in contrast to the HEFT algorithm, which always selects the device with the earliest finish time without taking the number of tasks on the machine into

account, the LB-HEFT algorithm depends on the number of tasks assigned to each machine, which influences consumption of the devices at a rate close to one another. Furthermore, the progress percentage of both makespan (Pp_m) and utilization (Pp_u) revealed by LB-HEFT is significantly related to the other compared algorithms and was also computed. The model is also enhanced on other parameters like reliability and average resource utilization. The reliability of the model was enhanced by making the proposed scheduling approach fault-tolerant. The fault tolerance is incorporated in the model by using the reservation technique where the VM is bound to the task for the pre-estimated time slot known as the reservation slot. This bounding of VMs to the task manages the failure of any cloud-based VM. The model was compared with 5 existing models i.e., HEFT, E-HEFT, LB-HEFT, FTSA-1, and DBSA by varying the number of VMs and tasks. The evaluations are carried out for small and large task scales. The outcomes demonstrate that the suggested model outperforms all the considered methods. More particularly, as the number of tasks is increasing the model shows the constant growth in the considered parameters.

Chapter 6
Conclusion and Future Directions

6.1. Conclusion

This research has navigated the complex environment of fault-tolerant resource provisioning in cloud computing, aiming to strike a delicate balance between performance optimization and user satisfaction. The journey originated with a systematic discovery of existing fault-tolerant techniques, highlighting their role in enhancing the cloud computing landscape. QoS constraints emerged as central elements influencing the equilibrium between resource utilization and SLAs. The succeeding chapters revealed a description of innovation and observed exploration. Existing approaches were inspected, and in response to the discovered gaps, novel frameworks and models were introduced. The RFRTS model tackled task scheduling and reservation-based fault tolerance, while the HFSLM model developed resource allocation, achieving a balanced synergy between user satisfaction and performance metrics. Moreover, HFSLM is designed to satisfy the load distribution equilibrium post-to-fault tolerance. This equilibrium is maintained by integrating the load balancing with fault tolerance. CRFTS strategy further emphasized the potential for significant reductions in Makespan and improved Resource Utilization. Producing these contributions, the findings establish noticeable improvements in the effectiveness of cloud computing settings. The proposed contexts have demonstrated not only theoretical possibilities but also realistic feasibility, as demonstrated by empirical measurements. These outcomes establish the efficiency of strategies in resource utilization, enhancing task execution times, and improved reliability measures.

- Chapter 1 serves as the entry point into an extensive exploration of cloud computing.

It begins by offering a comprehensive introduction to the complex landscape of cloud computing, laying the foundation for a detailed exploration of its broad-ranging impacts.

A pivotal focus of the chapter is the insightful dissection of methodologies and strategies intricately woven into the fabric of cloud computing to achieve resource provisioning. By delving into these approaches, the chapter not only elucidates their theoretical underpinnings but also discerns their practical applications, shedding light on their pivotal role in optimizing cloud performance and satisfying the SLAs.

In conclusion, Chapter 1 serves as a comprehensive introduction to the intricacies of cloud computing and lays a robust groundwork for the succeeding chapters. By weaving together theoretical insights, practical applications, and a keen awareness of environmental

implications, this chapter not only informs but also piques curiosity, setting the stage for a thorough exploration of fault-tolerant resource provisioning in cloud computing.

- In Chapter 2, diverse models for analyzing the faults, and rectifying these faults by implementing fault-tolerance integrated with scheduling and load-balancing strategies in cloud environments are comprehensively surveyed. The main emergent and developing methods regarding fault tolerance in the cloud environment are categorized into Proactive, Reactive, and Resilient. In resilient approaches, the revolutionary technologies AI/ML are considered and are observed to be more efficient than proactive and reactive techniques. It is because the reactive and proactive techniques normally employ the traditional procedures like, checkpoint restart, replication, migration, etc, which have limitations as these procedures could find it difficult to adjust dynamically or regulate to shifting demands, which could result in inefficiencies during times of high consumption.

After reviewing the literature, the below-mentioned conclusions can be drawn:

  - Checkpoint, restarting, and replication were found to be the frequently used methods to address the faults in the cloud.
  - Scholars and researchers are more concerned with determining crash defects than hardware faults such as transient, intermittent, or permanent faults.
  - When it comes to the implementation tool for evaluating the presented algorithms, research is mostly using the Cloudsim tool.
  - Proactive approaches have been used more frequently than reactive and resilient.
  - Researchers are more motivated toward response time and less towards makespan, adaptability, accuracy, and crashes.
  - Since the resilient approach utilized machine learning and artificial intelligence to predict and handle faults; therefore, it is the forthcoming effort of fault tolerance in the cloud.

- Chapter 3 proposes explorations of fault tolerance and task scheduling. The RFRTS model is proposed in this chapter which is concluded below:

Response Ranked Task Scheduling and Advance Reservation:

Initially, the proposed ranked-based scheduling approach is used for task allocation, and later reservation-based reactive fault tolerance method is suggested for a cloud system. To achieve the highest level of cloud computing infrastructure reliability, the suggested technique considers CPU faults and the VM reservation will ensure the assignment of an

alternative VM to the affected task. The proposed fault-tolerant approach has been compared with three existing reliable fault-tolerant approaches namely multi-objective scheduling algorithm with Fuzzy Resource utilization (FR-MOS), Cost-effective Workflow Scheduling Algorithm (CWS), and Fault-tolerant Cost-effective Workflow Scheduling Algorithm (FCWS) based on reliability. The outcomes unequivocally show that our suggested RFRTS algorithm surpasses the current FR-MOS, CWS, and FCWS considering reliability in all the states.

Since system reliability is one of the major issues in cloud systems, focusing on "execution till completion" is a crucial factor in enhancing reliability, therefore fault tolerance is necessary to achieve. The research suggests a method for task ranking by considering task lengths and task wait times. Besides, the algorithm implies an allocation strategy based on the determined rank value. Also, we provide a fault-tolerance strategy in which VM reservations are made based on a pre-calculated reservation window. The paper's focus is on the system's reliability. The major drawback of the suggested allocation is that it could not minimize the makespan. However, it will unquestionably improve the task response times by focusing on the wait time of the tasks. The study's future demand for working with the suggested ranked scheduling technique, where the VMs will also be ranked to work over further optimizations of makespan and resource utilisation. The model will be extended by accompanying load-balancing techniques for further optimization of the environment.

The proposed RFRTS was evaluated on reliability by comparing it with other reliable existing models namely, FCWS, FR-MOS, and CWS. We selected five distinct states of task numbers with varying lengths for the simulation we created: Small(S)[n = 50 approx], Medium(M)[n=100 approx], Medium large(M-L)[n=200 approx], Large(L)[n = 400 approx], Extra large(E-L)[n=600 approx].

In S, the minimum improvement by the model was seen to be 0.30% while the maximum improvement was seen to be at 2.25%. In M, the minimum improvement by the model was seen to be 1.32% while the maximum improvement was seen to be 2.36%. In M-L, the minimum improvement by the model was seen to be 1.53% while the maximum improvement was seen to be 2.37%. In L, the minimum improvement by the model was seen to be 1.65% while the maximum improvement was seen to be 2.18%. In E-L, the minimum improvement by the model was seen to be 1.26% while the maximum improvement was seen to be 2.45%.

- In Chapter 4, a Hybrid Fault-tolerant Scheduling and Load balancing Model is introduced employing neighboring-based VM to control failure in the cloud system with high computational demands. HFSLM uses a proficient task allocation strategy and distributes the arriving tasks among VMs at the arrival. In case of fault, the model uses the neighboring VMs of the faulty VM as a substitute and allocates an alternate VM to the affected task. Moreover, the proposed model escorts the whole system with an efficient load-balancing algorithm and maintains load equilibrium post-to-fault tolerance. After the implementation of the model in Python, performance evaluation was carried out by comparing HFSLM with FTHRM, MIN-MIN, MAX-MIN, and OLB on a low task scale by varying the task and VM in four different heterogeneities. The evaluations were performed based on makespan and average VM utilization. On very large task scales, the model was also contrasted with two other emerging models i.e., ELISA and MELISA.

The evaluation has been done by adjusting the number of tasks and VMs, size of tasks, and capacities of VMs in four different heterogeneities given by [35] i.e., HH, HL, LH, LL. For all these four cases the working efficiency of the proposed model and compared strategies have been analyzed and depicted graphically in the given figures. In comparison, the tasks have been taken on a small scale varying from 250 to 1000. On the other hand, the VMs have varied from 16 to 128.

The suggested technique outperforms FTHRM in terms of makespan and utilization, which go from 0.72% to 10.8% and 1.01% to more than 50%, respectively. When compared to MAX-MIN, HFSLM exhibits makespan improvements of -3.03% to 8.8% and average resource utilization gains of -2.15% to 6.7%. While comparing the suggested approach with MIN-MIN, the model shows an improvement of 0.6% to 19% in makespan and 1.09% to more than 45% in utilization. However, OLB was seen to perform very weakly among all approaches where the suggested model shows improvements of more than 50% in both makespan and utilization than OLB. Furthermore, it was observed that all the models perform almost equal optimization in makespan in LH heterogeneity. However, in that case, also OLB performs weakly among the compared approaches.

Additionally, Comparing HFSLM with ELISA and MELISA on a large tasks scale, HFSLM shows improvements from -0.98% to 23.33% and from -3% to 8% on makespan respectively. Besides, HFSLM shows 1.42% and 1.22% improvements in minimum resource utilization as compared to ELISA and MELISA respectively. On maximum resource utilization, the proposed model shows improvements of 39.1% and 48.8% respectively.

The suggested approach outperformed other considered strategies for QoS parameters. A few reasons are listed below:

- The proposed allocation considers both the upcoming tasks and newly added and deleted VMs. Additionally, optimal load distribution and effective average resource utilization occur simultaneously. As a result, it provides significant enhancement in all considered parameters.
- As can be seen from the overall results the utilization of the proposed approach remains optimized on varying the number of tasks and VMs. This is because the proposed allocation strategy focuses on distributing the arriving tasks throughout the available VMs. Moreover, various strategic advancements in the proposed HFSLM play a significant role in the same.
- Furthermore, the proposed model outperforms all the compared approaches in HH and HL cases. It is because in high task heterogeneity the ready time of all the available VMs will always be sorted in other words, whenever we have high task heterogeneity, the ready time of all the VMs in the VM list will always be sorted. The sorted ready time of VMs is the best case for the proposed allocation.

- In Chapter 5, Clustering and Reservation Fault-tolerant Scheduling (CRFTS) is introduced, which maximizes the system reliability while making it fault-tolerant and optimizing other Quality of Service (QoS) parameters, such as Makespan, Average Resource Utilization, and Reliability. The study optimizes the allocation of tasks to improve the utilization of resources and reduce the time required for their completion. At the same time, the reservation-based fault tolerance framework is presented, emphasizing reactive strategies, thus ensuring continuous service delivery throughout its execution without any interruption. The effectiveness of the suggested model is illustrated through simulations and empirical analyses, highlighting enhancements in QoS parameters while comparing with HEFT, FTSA-1, DBSA, E-HEFT, and the latest LB-HEFT for various cases/conditions over both tasks and VMs. An analysis of the performance of the proposed CRFTS algorithm has been conducted in small and large task scales. The proposed model was compared based on Makespan and Average Resource Utilization while competing with HEFT, E-HEFT, and the latest LB-HEFT algorithm and based on Reliability while competing with HEFT, FTSA-1, and DBSA on a small task scale. On a large task scale, CRFTS competed with HEFT, DBSA, and FTSA-1 based on makespan and average resource utilization. The conclusions drawn from the evaluation are listed below:

- In the case of makespan, CRFTS achieved maximum progress of 65.22%, 67.42%, and 62.91% on a small task scale while comparing with HEFT, E-HEFT, and the latest LB-HEFT respectively.
- In the case of average resource utilization, CRFTS achieved maximum progress of 30.42%, 22.11%, and 16.94% in a small task scale while comparing with HEFT, E-HEFT, and the latest LB-HEFT respectively.
- Furthermore, the CRFTS shows an increase of 12.71% to 71.06%, 1.12% to 3.95%, and 1.31% to 6.54% than HEFT, FTSA-1, and DBSA, respectively.
- Additionally, CRFTS achieved maximum progress of 65.22%, 67.42%, and 62.91% on a small task scale while comparing with HEFT, E-HEFT, and the latest LB-HEFT respectively.
- In large task scale, HEFT offers an optimized makespan after the proposed CRFTS. However, FTSA-1 provides the highest makespan among all the approaches. It may be because FTSA-1 is focusing more on reliability than that of makespan. According to the results related to makespan, CRFTS shows 40%, 6.32%, 9.358%, and 25.870% improvements over HEFT in 512, 1024, 2048, and 4096 tasks respectively. Comparing CRFTS with DBSA and FTSA-1, the proposed CRFTS shows more than 40% improvements in varying the number of jobs, respectively.
- In large task scales, the average resource utilization provided by FTSA-1 was found to be very low compared to the other approaches. The utilization offered by HEFT is second optimal after CRFTS. CRFTS was found optimum as compared to other approaches. According to the results concerning average resource utilization, CRFTS shows 14.38624%, 2.415644%, 3.169663%, and 2.083104% improvements over HEFT, 36.55696%, 37.33824%, 30.07273%, and 19.15755% improvements DBSA and 19.05063%, 7.11148%, 8.54854%, and 8.130822% improvements over FTSA-1 in 512, 1024, 2048, and 4096 number of tasks respectively.

- Chapter 6 acts as the pivotal nexus, synthesizing the results derived from the contributions and key findings presented across this thesis. Providing a comprehensive conclusion, it encapsulates the core of the research and articulates recommendations for potential future avenues of exploration.

6.2. Forthcoming Research Directions and Open Issues

It can be examined from the reviewed state-of-art that some important QoS parameters, except Response Time, are not being focused on. Other parameters, such as makespan, turnaround time, waiting time, flowtime, resource utilization, and accuracy, also need to be considered. Furthermore, various other faults, like byzantine and system crashes, etc., are also not examined much in hybrid fault tolerance algorithms. Therefore, it is necessary to enhance the performance of these hybrid fault tolerance algorithms by contemplating these limitations in forthcoming research. Moreover, researchers should focus on some of the below-mentioned aspects to overcome the limitations of existing techniques.

- Focus more on resilient fault tolerance.
- Focus on the computational cost along with fault tolerance.
- Identify and predict the faults accurately.
- Resolve faults with load balancing and scheduling.
- Fault handling with optimization of other QoS parameters.

6.2.1. Future works

After careful consideration and assessment, it is concluded that several research fields might be followed to raise the performance of cloud computing and boost the optimization of QoS parameters of cloud systems. They are listed below:

1. The researchers can make the scheduling efficient for better makespan and average resource utilization.

2. The assessed state-of-the-art shows that, except for response time, certain crucial QoS criteria are not being prioritized. It is also necessary to take into account additional factors including turnaround time, waiting time, flow time, resource utilization, and accuracy.

3. To improve task execution time and scheduling, a large body of research is focused on discovering resource and workload identification criteria. For workloads to be adaptive, scalable, and optimal, under and overusing resources should be avoided.

4. A sender-initiated load balancing mechanism that assists in uniform load distribution among dispersed nodes is necessary for task relocation.

5. Reservation can be used for fault tolerance as suggested in [72] for ensuring complete execution of tasks where the resources are reserved well in advance and may be used in case of faults.

6. It is essential to concentrate on limiting the penalty while taking into account system failures to attain QoS optimization-based allocation.

7. Only a few scheduling methods include the availability parameter, and it's highly dependent on VM failure and changes in the impact rate of users, therefore, to decrease VM failure, it is important to take this parameter into further consideration in later algorithms.
8. The penalties on account of faults can be minimized by accompanying the models with efficient load-balancing techniques.
9. It is clear from examining several methods that a task scheduling algorithm by itself cannot address all the issues. Most algorithms base their work scheduling on a few factors. One method, for instance, only considers the response time and execution time parameters and overlooks other QoS principles like the execution cost, dependability, utilization, etc. Therefore, by including more standards, an improved scheduling algorithm that can produce better results may be developed.
10. Future studies should consider the factors of scalability, elasticity, and other fault overheads which are the properties of the system to fit in a situation.

6.2.2. Methodical Roadmap for Open Challenges

A structured strategy or roadmap presented in Figure 6.1 that incorporates prioritization based on influence and feasibility is needed to address the scheduling and load balancing with fault tolerance challenges.

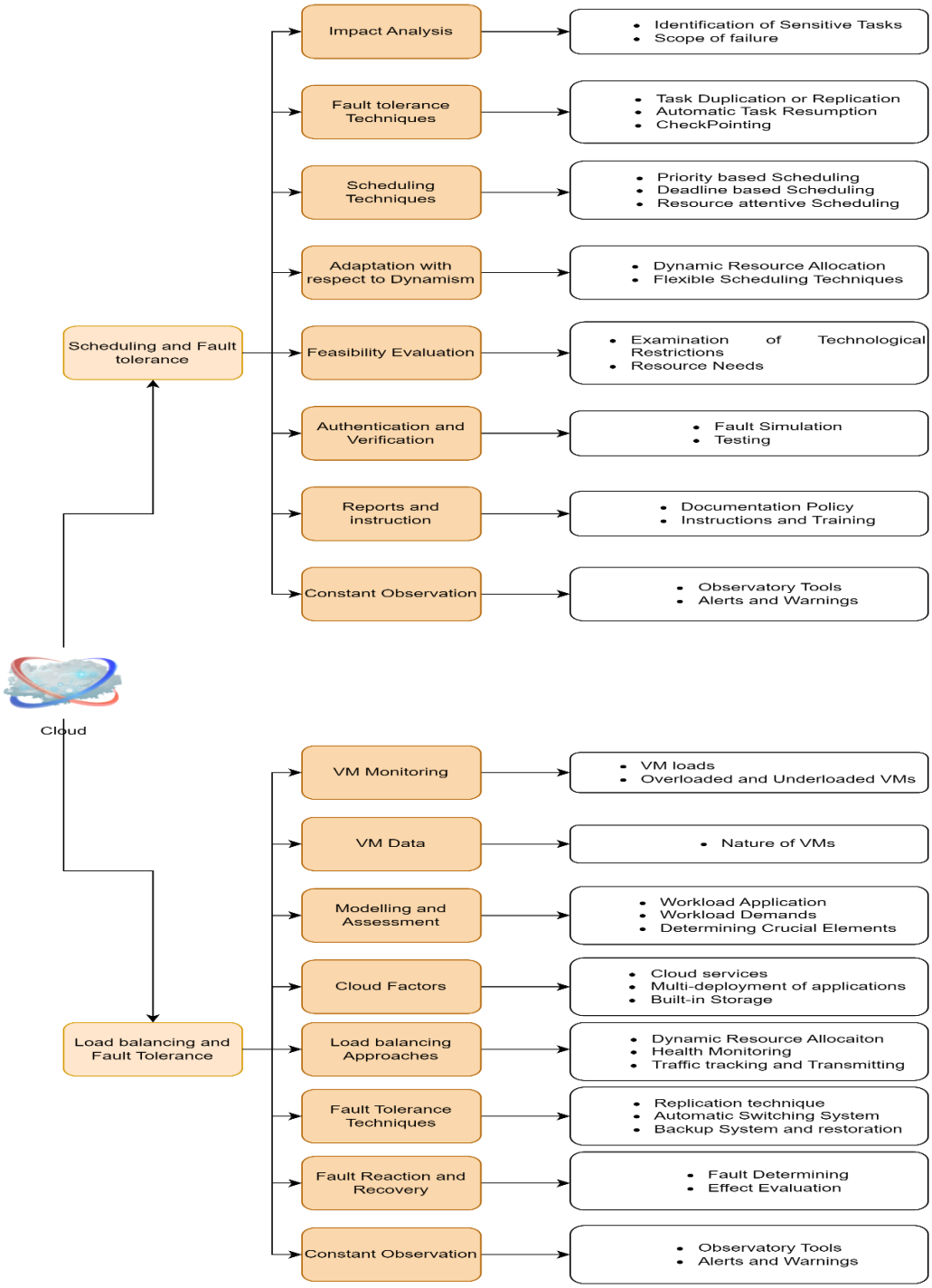


Figure 6.1. Showing the Proposed Structured Roadmap to Address the Cloud Challenges

References:

[1] R. Prasad and V. Rohokale, *Cyber Security: The Lifeline of Information and Communication Technology*. in Springer Series in Wireless Technology. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-31703-4.

[2] D. Lowe and B. Galhotra, "An Overview of Pricing Models for Using Cloud Services with analysis on Pay-Per-Use Model," *IJET*, vol. 7, no. 3.12, p. 248, Jul. 2018, doi: 10.14419/ijet.v7i3.12.16035.

[3] I. Odun-Ayo, M. Ananya, F. Agono, and R. Goddy-Worlu, "Cloud Computing Architecture: A Critical Analysis," in *2018 18th International Conference on Computational Science and Applications (ICCSA)*, Melbourne, Australia, Australia: IEEE, Jul. 2018, pp. 1–7. doi: 10.1109/ICCSA.2018.8439638.

[4] M. A. Mukwevho and T. Celik, "Toward a Smart Cloud: A Review of Fault-Tolerance Methods in Cloud Systems," *IEEE Trans. Serv. Comput.*, vol. 14, no. 2, pp. 589–605, Mar. 2021, doi: 10.1109/TSC.2018.2816644.

[5] O. Alzakholi, L. Haji, H. Shukur, R. Zebari, S. Abas, and M. Sadeeq, "Comparison Among Cloud Technologies and Cloud Performance," *JASTT*, vol. 1, no. 1, pp. 40–47, Apr. 2020, doi: 10.38094/jastt1219.

[6] S. Smys, R. Bestak, and Á. Rocha, Eds., *Inventive Computation Technologies*, vol. 98. in Lecture Notes in Networks and Systems, vol. 98. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-33846-6.

[7] U. Samal and A. Kumar, "A software reliability model incorporating fault removal efficiency and it's release policy," *Comput Stat*, vol. 39, no. 6, pp. 3137–3155, Sep. 2024, doi: 10.1007/s00180-023-01430-9.

[8] U. Samal and A. Kumar, "Metrics and trends: a bibliometric approach to software reliability growth models," *Total Quality Management & Business Excellence*, vol. 35, no. 11–12, pp. 1274–1295, Aug. 2024, doi: 10.1080/14783363.2024.2366510.

[9] U. Samal and A. Kumar, "A Neural Network Approach for Software Reliability Prediction," *Int. J. Rel. Qual. Saf. Eng.*, vol. 31, no. 03, p. 2450009, Jun. 2024, doi: 10.1142/S0218539324500098.

[10] S. Kumar and D. A. S. Kushwaha, "Future of Fault Tolerance in Cloud Computing," vol. 22, no. 17, 2019.

[11] V. Gupta, B. P. Kaur, and S. Jangra, "An efficient method for fault tolerance in cloud environment using encryption and classification," *Soft Comput*, vol. 23, no. 24, pp. 13591–13602, Dec. 2019, doi: 10.1007/s00500-019-03896-6.

[12] Sir Syed University of Engineering and Technology Karachi, Pakistan *et al.*, "A Systematic Survey of Simulation Tools for Cloud and Mobile Cloud Computing Paradigm," *JISR-C*, vol. 20, no. 1, Jun. 2022, doi: 10.31645/JISRC.22.20.1.10.

[13] M. A. Shahid, M. M. Alam, and M. M. Su'ud, "A Systematic Parameter Analysis of Cloud Simulation Tools in Cloud Computing Environments," *Applied Sciences*, vol. 13, no. 15, p. 8785, Jul. 2023, doi: 10.3390/app13158785.

[14] H. Arabnejad, C. Pahl, G. Estrada, A. Samir, and F. Fowley, "A Fuzzy Load Balancer for Adaptive Fault Tolerance Management in Cloud Platforms," in *Service-Oriented and Cloud Computing*, vol. 10465, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds., in Lecture Notes in Computer Science, vol. 10465. , Cham: Springer International Publishing, 2017, pp. 109–124. doi: 10.1007/978-3-319-67262-5_9.

[15] M. K. Edemo, "DEVELOPING FAULT TOLERANCE ARCHITECTURE FOR REAL-TIME SYSTEMS OF CLOUD COMPUTING".

[16] T. Zaidi, "Modeling for Fault Tolerance in Cloud Computing Environment," *Journal of Computer Sciences and Applications*.

[17] S. M. Abdulhamid, M. S. Abd Latiff, S. H. H. Madni, and M. Abdullahi, "Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm," *Neural Comput & Applic*, vol. 29, no. 1, pp. 279–293, Jan. 2018, doi: 10.1007/s00521-016-2448-8.

[18] S. Sengupta and A. Negi, "Comparative Analysis of Contrast Enhancement Techniques for MRI Images," in *Proceeding of the International Conference on Computer Networks, Big Data and IoT (ICCBI - 2019)*, vol. 49, A. P. Pandian, R. Palanisamy, and K. Ntalianis, Eds., in Lecture Notes on Data Engineering and Communications Technologies, vol. 49. , Cham: Springer International Publishing, 2020, pp. 290–296. doi: 10.1007/978-3-030-43192-1_33.

[19] A. Ganesh, M. Sandhya, and S. Shankar, "A study on fault tolerance methods in Cloud Computing," in *2014 IEEE International Advance Computing Conference (IACC)*, Gurgaon, India: IEEE, Feb. 2014, pp. 844–849. doi: 10.1109/IAdCC.2014.6779432.

[20] P. Gupta and P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-37319-1.

[21] P. KumarPatra, H. Singh, and G. Singh, “Fault Tolerance Techniques and Comparative Implementation in Cloud Computing,” *IJCA*, vol. 64, no. 14, pp. 37–41, Feb. 2013, doi: 10.5120/10705-5643.

[22] M. Hasan and M. S. Goraya, “Fault tolerance in cloud computing environment: A systematic survey,” *Computers in Industry*, vol. 99, pp. 156–172, Aug. 2018, doi: 10.1016/j.compind.2018.03.027.

[23] G. Singh and S. Kinger, “A Survey On Fault Tolerance Techniques And Methods In Cloud Computing,” *International Journal of Engineering Research*, vol. 2, no. 6, 2013.

[24] M. Khaldi, M. Rebbah, B. Meftah, and O. Smail, “Fault tolerance for a scientific workflow system in a Cloud computing environment,” *International Journal of Computers and Applications*, vol. 42, no. 7, pp. 705–714, Oct. 2020, doi: 10.1080/1206212X.2019.1647651.

[25] Z. Amin, H. Singh, and N. Sethi, “Review on Fault Tolerance Techniques in Cloud Computing,” *IJCA*, vol. 116, no. 18, pp. 11–17, Apr. 2015, doi: 10.5120/20435-2768.

[26] S. Prathiba and S. Sowvarnica, “Survey of failures and fault tolerance in cloud,” in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, Chennai, India: IEEE, Feb. 2017, pp. 169–172. doi: 10.1109/ICCCT2.2017.7972271.

[27] Z. Xia, Y. Zhu, X. Sun, Z. Qin, and K. Ren, “Towards Privacy-Preserving Content-Based Image Retrieval in Cloud Computing,” *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 276–286, Jan. 2018, doi: 10.1109/TCC.2015.2491933.

[28] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, “Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends,” *Swarm and Evolutionary Computation*, vol. 62, p. 100841, Apr. 2021, doi: 10.1016/j.swevo.2021.100841.

[29] M. A. Shahid, N. Islam, M. M. Alam, M. S. Mazliham, and S. Musa, “Towards Resilient Method: An exhaustive survey of fault tolerance methods in the cloud computing environment,” *Computer Science Review*, vol. 40, p. 100398, May 2021, doi: 10.1016/j.cosrev.2021.100398.

[30] G. P. Sarmila, N. Gnanambigai, and P. Dinadayalan, “Survey on fault tolerant — Load balancing algorithmsin cloud computing,” in *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, India: IEEE, Feb. 2015, pp. 1715–1720. doi: 10.1109/ECS.2015.7124879.

[31] K. Kotecha, V. Piuri, H. N. Shah, and R. Patel, Eds., *Data Science and Intelligent Applications: Proceedings of ICDSIA 2020*, vol. 52. in Lecture Notes on Data Engineering and Communications Technologies, vol. 52. Singapore: Springer Singapore, 2021. doi: 10.1007/978-981-15-4474-3.

[32] M. Dhingra and N. Gupta, "ALGORITHMS TO ENHANCE THE RELIABILITY OF VIRTUAL NODES USING ADAPTIVE FAULT TOLERANCE TECHNIQUES," *COMPUTER SCIENCE*, 2019.

[33] S. Singh and I. Chana, "Resource provisioning and scheduling in clouds: QoS perspective," *J Supercomput*, vol. 72, no. 3, pp. 926–960, Mar. 2016, doi: 10.1007/s11227-016-1626-x.

[34] S. Umar Mushtaq, S. Sheikh, and S. M. Idrees, "Next-Gen Cloud Efficiency: Fault-Tolerant Task Scheduling With Neighboring Reservations for Improved Resource Utilization," *IEEE Access*, vol. 12, pp. 75920–75940, 2024, doi: 10.1109/ACCESS.2024.3404643.

[35] T. D. Braun *et al.*, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, Jun. 2001, doi: 10.1006/jpdc.2000.1714.

[36] N. M. Reda, A. Tawfik, M. A. Marzok, and S. M. Khamis, "Sort-Mid tasks scheduling algorithm in grid computing," *Journal of Advanced Research*, vol. 6, no. 6, pp. 987–993, Nov. 2015, doi: 10.1016/j.jare.2014.11.010.

[37] M. K. Gokhroo, M. C. Govil, and E. S. Pilli, "Detecting and mitigating faults in cloud computing environment," in *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*, Ghaziabad: IEEE, Feb. 2017, pp. 1–9. doi: 10.1109/CIACT.2017.7977362.

[38] T. J. Charity and G. C. Hua, "Resource reliability using fault tolerance in cloud computing," in *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Dehradun, India: IEEE, Oct. 2016, pp. 65–71. doi: 10.1109/NGCT.2016.7877391.

[39] A. S. M. Noor, N. F. M. Zian, N. H. A. Rahim, R. Mamat, and W. N. A. W. Azman, "Novelty circular neighboring technique using reactive fault tolerance method," *IJECE*, vol. 9, no. 6, p. 5211, Dec. 2019, doi: 10.11591/ijece.v9i6.pp5211-5217.

[40] G. Vallee *et al.*, “A Framework for Proactive Fault Tolerance,” in *2008 Third International Conference on Availability, Reliability and Security*, IEEE, Mar. 2008, pp. 659–664. doi: 10.1109/ARES.2008.171.

[41] C. Engelmann, G. R. Vallee, T. Naughton, and S. L. Scott, “Proactive Fault Tolerance Using Preemptive Migration,” in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Weimar, Germany: IEEE, 2009, pp. 252–257. doi: 10.1109/PDP.2009.31.

[42] A. Ragmani, A. Elomri, N. Abghour, K. Moussaid, M. Rida, and E. Badidi, “Adaptive fault-tolerant model for improving cloud computing performance using artificial neural network,” *Procedia Computer Science*, vol. 170, pp. 929–934, 2020, doi: 10.1016/j.procs.2020.03.106.

[43] S. M. Hosseini and M. G. Arani, “Fault-Tolerance Techniques in Cloud Storage: A Survey,” *IJDTA*, vol. 8, no. 4, pp. 183–190, Aug. 2015, doi: 10.14257/ijdta.2015.8.4.19.

[44] S. K. Battula, S. Garg, J. Montgomery, and B. Kang, “An Efficient Resource Monitoring Service for Fog Computing Environments,” *IEEE Trans. Serv. Comput.*, vol. 13, no. 4, pp. 709–722, Jul. 2020, doi: 10.1109/TSC.2019.2962682.

[45] M. Nazari Cheraghlou, A. Khadem-Zadeh, and M. Haghparast, “A survey of fault tolerance architecture in cloud computing,” *Journal of Network and Computer Applications*, vol. 61, pp. 81–92, Feb. 2016, doi: 10.1016/j.jnca.2015.10.004.

[46] R. Shah, B. Veeravalli, and M. Misra, “On the Design of Adaptive and Decentralized Load Balancing Algorithms with Load Estimation for Computational Grid Environments,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1675–1686, Dec. 2007, doi: 10.1109/TPDS.2007.1115.

[47] B. Mohammed, “A FRAMEWORK FOR EFFICIENT MANAGEMENT OF FAULT TOLERANCE IN CLOUD DATA CENTRES AND HIGH- PERFORMANCE COMPUTING SYSTEMS”.

[48] M. Nazari Cheraghlou, A. Khademzadeh, and M. Haghparast, “New Fuzzy-Based Fault Tolerance Evaluation Framework for Cloud Computing,” *J Netw Syst Manage*, vol. 27, no. 4, pp. 930–948, Oct. 2019, doi: 10.1007/s10922-019-09491-2.

[49] M. R. Chinnaiah and N. Niranjan, “Fault tolerant software systems using software configurations for cloud computing,” *J Cloud Comp*, vol. 7, no. 1, p. 3, Dec. 2018, doi: 10.1186/s13677-018-0104-9.

[50] J. A. Liakath, P. Krishnadoss, and G. Natesan, “DCCWOA: A multi-heuristic fault tolerant scheduling technique for cloud computing environment,” *Peer-to-Peer Netw. Appl.*, vol. 16, no. 2, pp. 785–802, Mar. 2023, doi: 10.1007/s12083-022-01445-x.

[51] H. Xu, S. Xu, W. Wei, and N. Guo, “Fault tolerance and quality of service aware virtual machine scheduling algorithm in cloud data centers,” *J Supercomput*, vol. 79, no. 3, pp. 2603–2625, Feb. 2023, doi: 10.1007/s11227-022-04760-5.

[52] Dr. G. S. R. Dr.G.Sasikanth Reddy, “Fault Tolerance- Challenges, Techniques and Implementation in Cloud Computing,” *jst*, vol. 7, no. 10, pp. 30–34, Dec. 2022, doi: 10.46243/jst.2022.v7.i010.pp30-34.

[53] S. Goutam and A. K. Yadav, “Preemptable priority based dynamic resource allocation in cloud computing with fault tolerance,” in *2015 International Conference on Communication Networks (ICCN)*, Gwalior, India: IEEE, Nov. 2015, pp. 278–285. doi: 10.1109/ICCN.2015.54.

[54] B. Mohammed, I. Awan, H. Ugail, and M. Younas, “Failure prediction using machine learning in a virtualised HPC system and application,” *Cluster Comput*, vol. 22, no. 2, pp. 471–485, Jun. 2019, doi: 10.1007/s10586-019-02917-1.

[55] S. M. Abdulhamid and M. S. A. Latiff, “A checkpointed league championship algorithm-based cloud scheduling scheme with secure fault tolerance responsiveness,” *Applied Soft Computing*, vol. 61, pp. 670–680, Dec. 2017, doi: 10.1016/j.asoc.2017.08.048.

[56] J. Liu, M. Wei, W. Hu, X. Xu, and A. Ouyang, “Task scheduling with fault-tolerance in real-time heterogeneous systems,” *Journal of Systems Architecture*, vol. 90, pp. 23–33, Oct. 2018, doi: 10.1016/j.sysarc.2018.08.007.

[57] D. L. P. Saikia and Y. L. Devi, “FAULT TOLEREANE TECHNIQUES AND ALGORITHMS IN CLOUD COMPUTING,” vol. 4.

[58] S. M. A. Ataallah, S. M. Nassar, and E. E. Hemayed, “Fault tolerance in cloud computing - survey,” in *2015 11th International Computer Engineering Conference (ICENCO)*, Cairo, Egypt: IEEE, Dec. 2015, pp. 241–245. doi: 10.1109/ICENCO.2015.7416355.

[59] Jeongmin Park, Giljong Yoo, and Eunseok Lee, “Proactive self-healing system based on multi-agent technologies,” in *Third ACIS Int’l Conference on Software Engineering Research, Management and Applications (SERA’05)*, Mount Pleasant, MI, USA: IEEE, 2005, pp. 256–263. doi: 10.1109/SERA.2005.55.

[60] J. C. Patni, M. S. Aswal, O. P. Pal, and A. Gupta, “Load balancing strategies for Grid computing,” in *2011 3rd International Conference on Electronics Computer Technology*, Kanyakumari, India: IEEE, Apr. 2011, pp. 239–243. doi: 10.1109/ICECTECH.2011.5941745.

[61] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, “Grid load balancing using intelligent agents,” *Future Generation Computer Systems*, vol. 21, no. 1, pp. 135–149, Jan. 2005, doi: 10.1016/j.future.2004.09.032.

[62] J. Balasangameshwara and N. Raju, “A hybrid policy for fault tolerant load balancing in grid computing environments,” *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 412–422, Jan. 2012, doi: 10.1016/j.jnca.2011.09.005.

[63] S. Talwani and I. Chana, “Fault tolerance techniques for scientific applications in cloud,” in *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, Noida, India: IEEE, Aug. 2017, pp. 1–5. doi: 10.1109/TEL-NET.2017.8343578.

[64] D. A. Shafiq, N. Z. Jhanjhi, and A. Abdullah, “Load balancing techniques in cloud computing environment: A review,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3910–3933, Jul. 2022, doi: 10.1016/j.jksuci.2021.02.007.

[65] P. Kumari and P. Kaur, “A survey of fault tolerance in cloud computing,” *Journal of King Saud University - Computer and Information Sciences*, vol. 33, no. 10, pp. 1159–1176, Dec. 2021, doi: 10.1016/j.jksuci.2018.09.021.

[66] S. Bharany *et al.*, “Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy,” *Sustainable Energy Technologies and Assessments*, vol. 53, p. 102613, Oct. 2022, doi: 10.1016/j.seta.2022.102613.

[67] A. Kumar and P. Chawla, “A Systematic Literature Review on Load Balancing Algorithms of Virtual Machines in a Cloud Computing Environment,” *SSRN Journal*, 2020, doi: 10.2139/ssrn.3564355.

[68] K. B. Et. Al., “Load balancing in Cloud Computing: Issues and Challenges,” *TURCOMAT*, vol. 12, no. 2, pp. 3224–3231, Apr. 2021, doi: 10.17762/turcomat.v12i2.2380.

[69] U. Samal and A. Kumar, “Enhancing Software Reliability Forecasting Through a Hybrid ARIMA-ANN Model,” *Arab J Sci Eng*, vol. 49, no. 5, pp. 7571–7584, May 2024, doi: 10.1007/s13369-023-08486-1.

[70] R. Indhumathi, K. Amuthabala, G. Kiruthiga, N. Yuvaraj, and A. Pandey, “Design of Task Scheduling and Fault Tolerance Mechanism Based on GWO Algorithm for Attaining Better QoS in Cloud System,” *Wireless Pers Commun*, vol. 128, no. 4, pp. 2811–2829, Feb. 2023, doi: 10.1007/s11277-022-10072-x.

[71] L. Zhu, K. Huang, Y. Hu, and X. Tai, “A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata,” *IEEE Access*, vol. 9, pp. 81236–81252, 2021, doi: 10.1109/ACCESS.2021.3078773.

[72] S. Sheikh, A. Nagaraju, and M. Shahid, “A fault-tolerant hybrid resource allocation model for dynamic computational grid,” *Journal of Computational Science*, vol. 48, p. 101268, Jan. 2021, doi: 10.1016/j.jocs.2020.101268.

[73] Z. Momenzadeh and F. Safi-Esfahani, “Workflow scheduling applying adaptable and dynamic fragmentation (WSADF) based on runtime conditions in cloud computing,” *Future Generation Computer Systems*, vol. 90, pp. 327–346, Jan. 2019, doi: 10.1016/j.future.2018.07.041.

[74] F. Al-Turjman, *The Cloud in IoT-enabled spaces*. Boca Raton, FL: CRC Press, 2020.

[75] S. Meng, Q. Li, T. Wu, W. Huang, J. Zhang, and W. Li, “A fault-tolerant dynamic scheduling method on hierarchical mobile edge cloud computing,” *Computational Intelligence*, vol. 35, no. 3, pp. 577–598, Aug. 2019, doi: 10.1111/coin.12219.

[76] J. Thaman and M. Singh, “Cost-effective task scheduling using hybrid approach in cloud,” *IJGUC*, vol. 8, no. 3, p. 241, 2017, doi: 10.1504/IJGUC.2017.087813.

[77] S. Sheikh, M. Sharma, and A. Singh, *Recent Advances in Computing Sciences: Proceedings of RACS 2022*, 1st ed. London: CRC Press, 2023. doi: 10.1201/9781003405573.

[78] S. Antony, S. Antony, A. S. A. Beegom, and M. S. Rajasree, “Task Scheduling Algorithm with Fault Tolerance for Cloud,” in *2012 International Conference on Computing Sciences*, Phagwara, India: IEEE, Sep. 2012, pp. 180–182. doi: 10.1109/ICCS.2012.71.

[79] S. U. Mushtaq, S. Sheikh, and A. Nain, “The Response Rank based Fault-tolerant Task Scheduling for Cloud System,” in *Proceedings of the 2023 1st International Conference on Advanced Informatics and Intelligent Information Systems (ICAI3S 2023)*, vol. 181, A. Putro Suryotomo and H. Cahya Rustamaji, Eds., in Advances in Intelligent Systems Research, vol. 181. , Dordrecht: Atlantis Press International BV, 2024, pp. 37–48. doi: 10.2991/978-94-6463-366-5_5.

[80] M. A. Shahid, M. M. Alam, and M. M. Su'ud, "Achieving Reliability in Cloud Computing by a Novel Hybrid Approach," *Sensors*, vol. 23, no. 4, p. 1965, Feb. 2023, doi: 10.3390/s23041965.

[81] G. Chen, H. Jin, D. Zou, B. B. Zhou, W. Qiang, and G. Hu, "SHelp: Automatic Self-Healing for Multiple Application Instances in a Virtual Machine Environment," in *2010 IEEE International Conference on Cluster Computing*, Heraklion, Greece: IEEE, Sep. 2010, pp. 97–106. doi: 10.1109/CLUSTER.2010.18.

[82] S. Sidiroglou, O. Laadan, C. Perez, N. Viennot, J. Nieh, and A. D. Keromytis, "ASSURE: automatic software self-healing using rescue points," *SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 37–48, Mar. 2009, doi: 10.1145/2528521.1508250.

[83] I. P. Egwutuoha, S. Chen, D. Levy, B. Selic, and R. Calvo, "A Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the Cloud," in *2012 Second International Conference on Cloud and Green Computing*, Xiangtan, Hunan, China: IEEE, Nov. 2012, pp. 268–273. doi: 10.1109/CGC.2012.22.

[84] R. A. A.Vinnarasi, "Temperature Monitoring with the Linux Kernel on a Multi Core Processor," *IJIRSET*, vol. 04, no. 03, pp. 876–883, Mar. 2015, doi: 10.15680/IJIRSET.2015.0403011.

[85] K. Toshniwal and J. M. Conrad, "A web-based sensor monitoring system on a Linux-based single board computer platform," in *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, Concord, NC, USA: IEEE, Mar. 2010, pp. 371–374. doi: 10.1109/SECON.2010.5453851.

[86] D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and M. Scarpa, "Workload-Based Software Rejuvenation in Cloud Systems," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1072–1085, Jun. 2013, doi: 10.1109/TC.2013.30.

[87] J. Liu, J. Zhou, and R. Buyya, "Software Rejuvenation Based Fault Tolerance Scheme for Cloud Applications," in *2015 IEEE 8th International Conference on Cloud Computing*, New York City, NY, USA: IEEE, Jun. 2015, pp. 1115–1118. doi: 10.1109/CLOUD.2015.164.

[88] D. Sun, G. Zhang, C. Wu, K. Li, and W. Zheng, "Building a fault tolerant framework with deadline guarantee in big data stream computing environments," *Journal of Computer and System Sciences*, vol. 89, pp. 4–23, Nov. 2017, doi: 10.1016/j.jcss.2016.10.010.

[89] S. Malik and F. Huet, “Adaptive Fault Tolerance in Real Time Cloud Computing,” in *2011 IEEE World Congress on Services*, Washington, DC, USA: IEEE, Jul. 2011, pp. 280–287. doi: 10.1109/SERVICES.2011.108.

[90] Y. Zhang, Z. Zheng, and M. R. Lyu, “BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing,” in *2011 IEEE 4th International Conference on Cloud Computing*, Washington, DC, USA: IEEE, Jul. 2011, pp. 444–451. doi: 10.1109/CLOUD.2011.16.

[91] M. Hasan and M. S. Goraya, “Fault tolerance in cloud computing environment: A systematic survey,” *Computers in Industry*, vol. 99, pp. 156–172, Aug. 2018, doi: 10.1016/j.compind.2018.03.027.

[92] 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 Trans. Comput.*, vol. 64, no. 9, pp. 2545–2558, Sep. 2015, doi: 10.1109/TC.2014.2366751.

[93] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su’ud, and S. Musa, “A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach,” *IEEE Access*, vol. 8, pp. 130500–130526, 2020, doi: 10.1109/ACCESS.2020.3009184.

[94] S. Bharany *et al.*, “Energy-Efficient Clustering Scheme for Flying Ad-Hoc Networks Using an Optimized LEACH Protocol,” *Energies*, vol. 14, no. 19, p. 6016, Sep. 2021, doi: 10.3390/en14196016.

[95] F. Safara, A. Souri, T. Baker, I. Al Ridhawi, and M. Aloqaily, “PriNergy: a priority-based energy-efficient routing method for IoT systems,” *J Supercomput*, vol. 76, no. 11, pp. 8609–8626, Nov. 2020, doi: 10.1007/s11227-020-03147-8.

[96] A. N. Asadi, M. A. Azgomi, and R. Entezari-Maleki, “Analytical evaluation of resource allocation algorithms and process migration methods in virtualized systems,” *Sustainable Computing: Informatics and Systems*, vol. 25, p. 100370, Mar. 2020, doi: 10.1016/j.suscom.2019.100370.

[97] H. Yuan, H. Liu, J. Bi, and M. Zhou, “Revenue and Energy Cost-Optimized Biobjective Task Scheduling for Green Cloud Data Centers,” *IEEE Trans. Automat. Sci. Eng.*, vol. 18, no. 2, pp. 817–830, Apr. 2021, doi: 10.1109/TASE.2020.2971512.

[98] T. Welsh and E. Benkhelifa, “On Resilience in Cloud Computing: A Survey of Techniques across the Cloud Domain,” *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–36, May 2021, doi: 10.1145/3388922.

[99] S. Abapour, M. Nazari-Heris, B. Mohammadi-Ivatloo, and M. Tarafdar Hagh, “Game Theory Approaches for the Solution of Power System Problems: A Comprehensive Review,” *Arch Computat Methods Eng*, vol. 27, no. 1, pp. 81–103, Jan. 2020, doi: 10.1007/s11831-018-9299-7.

[100] K. Wang, J. Wu, X. Zheng, A. Jolfaei, J. Li, and D. Yu, “Leveraging Energy Function Virtualization With Game Theory for Fault-Tolerant Smart Grid,” *IEEE Trans. Ind. Inf.*, vol. 17, no. 1, pp. 678–687, Jan. 2021, doi: 10.1109/TII.2020.2971584.

[101] M. Asim Shahid, M. M. Alam, and M. Mohd Su’ud, “Improved accuracy and less fault prediction errors via modified sequential minimal optimization algorithm,” *PLoS ONE*, vol. 18, no. 4, p. e0284209, Apr. 2023, doi: 10.1371/journal.pone.0284209.

[102] R. Verma and S. Chandra, “HBI-LB: A Dependable Fault-Tolerant Load Balancing Approach for Fog based Internet-of-Things Environment,” *J Supercomput*, vol. 79, no. 4, pp. 3731–3749, Mar. 2023, doi: 10.1007/s11227-022-04797-6.

[103] T. Tamilvizhi and B. Parvathavarthini, “A novel method for adaptive fault tolerance during load balancing in cloud computing,” *Cluster Comput*, vol. 22, no. S5, pp. 10425–10438, Sep. 2019, doi: 10.1007/s10586-017-1038-6.

[104] S. M. A. Attallah, M. B. Fayek, S. M. Nassar, and E. E. Hemayed, “Proactive load balancing fault tolerance algorithm in cloud computing,” *Concurrency and Computation*, vol. 33, no. 10, p. e6172, May 2021, doi: 10.1002/cpe.6172.

[105] T. Mohmmed and N. Abdalrahman, “A Load Balancing with Fault Tolerance Algorithm for Cloud Computing,” in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, Khartoum, Sudan: IEEE, Feb. 2021, pp. 1–6. doi: 10.1109/ICCCEEE49695.2021.9429597.

[106] M. R. Sumalatha, C. Selvakumar, T. Priya, R. T. Azariah, and P. M. Manohar, “CLBC - Cost effective load balanced resource allocation for partitioned cloud system,” in *2014 International Conference on Recent Trends in Information Technology*, Chennai, India: IEEE, Apr. 2014, pp. 1–5. doi: 10.1109/ICRTIT.2014.6996174.

[107] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su’ud, and S. Musa, “A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach,” *IEEE Access*, vol. 8, pp. 130500–130526, 2020, doi: 10.1109/ACCESS.2020.3009184.

[108] B. Mohammed, M. Kiran, I.-U. Awan, and K. M. Maiyama, “An Integrated Virtualized Strategy for Fault Tolerance in Cloud Computing Environment,” in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted*

Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse: IEEE, Jul. 2016, pp. 542–549. doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0094.

[109] R. Geist, R. Reynolds, and J. Westall, "Selection of a checkpoint interval in a critical-task environment," *IEEE Trans. Rel.*, vol. 37, no. 4, pp. 395–400, Oct. 1988, doi: 10.1109/24.9847.

[110] P. Das and P. M. Khilar, "VFT: A virtualization and fault tolerance approach for cloud computing," in *2013 IEEE CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGIES*, Thuckalay, Tamil Nadu, India: IEEE, Apr. 2013, pp. 473–478. doi: 10.1109/CICT.2013.6558142.

[111] A. Semmoud, M. Hakem, B. Benmammar, and J. Charr, "Load balancing in cloud computing environments based on adaptive starvation threshold," *Concurrency and Computation*, vol. 32, no. 11, p. e5652, Jun. 2020, doi: 10.1002/cpe.5652.

[112] S. Dam, G. Mandal, K. Dasgupta, and P. Dutta, "Genetic algorithm and gravitational emulation based hybrid load balancing strategy in cloud computing," in *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, Hooghly, India: IEEE, Feb. 2015, pp. 1–7. doi: 10.1109/C3IT.2015.7060176.

[113] S. T. Mamta Khanchi*, "An Efficient Algorithm For Load Balancing In Cloud Computing," Jun. 2016, doi: 10.5281/ZENODO.55545.

[114] S. Dam, G. Mandal, K. Dasgupta, and P. Dutta, "An Ant Colony Based Load Balancing Strategy in Cloud Computing," in *Advanced Computing, Networking and Informatics- Volume 2*, vol. 28, M. Kumar Kundu, D. P. Mohapatra, A. Konar, and A. Chakraborty, Eds., in Smart Innovation, Systems and Technologies, vol. 28. , Cham: Springer International Publishing, 2014, pp. 403–413. doi: 10.1007/978-3-319-07350-7_45.

[115] A. N. Singh and S. Prakash, "WAMLB: Weighted Active Monitoring Load Balancing in Cloud Computing," in *Big Data Analytics*, vol. 654, V. B. Aggarwal, V. Bhatnagar, and D. K. Mishra, Eds., in Advances in Intelligent Systems and Computing, vol. 654. , Singapore: Springer Singapore, 2018, pp. 677–685. doi: 10.1007/978-981-10-6620-7_65.

[116] S. Ghosh and C. Banerjee, "Priority based Modified Throttled Algorithm in Cloud Computing," in *2016 International Conference on Inventive Computation Technologies*

(ICICT), Coimbatore, India: IEEE, Aug. 2016, pp. 1–6. doi: 10.1109/INVENTIVE.2016.7830175.

[117] S. Subalakshmi and N. Malarvizhi, "Enhanced Hybrid Approach for Load Balancing Algorithms in Cloud Computing," vol. 2, no. 2.

[118] A. N. Aliyu and P. B. Souley, "Performance Analysis of a Hybrid Approach to Enhance Load Balancing in a Heterogeneous Cloud Environment," *IJASRE*, vol. 5, no. 7, pp. 246–257, 2019, doi: 10.31695/IJASRE.2019.33430.

[119] M. N and P. A, "An Efficient Improved Weighted Round Robin Load Balancing Algorithm in Cloud Computing," *IJET*, vol. 7, no. 3.1, p. 110, Aug. 2018, doi: 10.14419/ijet.v7i3.1.16810.

[120] R. A. Haidri, C. P. Katti, and P. C. Saxena, "A load balancing strategy for Cloud Computing environment," in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*, Ajmer: IEEE, Jul. 2014, pp. 636–641. doi: 10.1109/ICSPCT.2014.6884914.

[121] V. L. Padma Latha, N. Sudhakar Reddy, and A. Suresh Babu, "RETRACTED: Optimizing Scalability and Availability of Cloud Based Software Services Using Modified Scale Rate Limiting Algorithm," *Theoretical Computer Science*, vol. 943, p. 230, Jan. 2023, doi: 10.1016/j.tcs.2022.07.019.

[122] S. Yuan, S. Das, R. Ramesh, and C. Qiao, "Availability-Aware Virtual Resource Provisioning for Infrastructure Service Agreements in the Cloud," *Inf Syst Front*, vol. 25, no. 4, pp. 1495–1512, Aug. 2023, doi: 10.1007/s10796-022-10302-4.

[123] C. Wang, Z. Fu, and G. Cui, "A neural-network-based approach for diagnosing hardware faults in cloud systems," *Advances in Mechanical Engineering*, vol. 11, no. 2, p. 1687814018819236, Feb. 2019, doi: 10.1177/1687814018819236.

[124] M. A. Shahid, M. M. Alam, and M. M. Su'ud, "Performance Evaluation of Load-Balancing Algorithms with Different Service Broker Policies for Cloud Computing," *Applied Sciences*, vol. 13, no. 3, p. 1586, Jan. 2023, doi: 10.3390/app13031586.

[125] M. Farid, R. Latip, M. Hussin, and N. A. W. Abdul Hamid, "Scheduling Scientific Workflow Using Multi-Objective Algorithm With Fuzzy Resource Utilization in Multi-Cloud Environment," *IEEE Access*, vol. 8, pp. 24309–24322, 2020, doi: 10.1109/ACCESS.2020.2970475.

[126] S. Kianpisheh, N. M. Charkari, and M. Kargahi, "Reliability-driven scheduling of time/cost-constrained grid workflows," *Future Generation Computer Systems*, vol. 55, pp. 1–16, Feb. 2016, doi: 10.1016/j.future.2015.07.014.

[127] X. Tang, “Reliability-Aware Cost-Efficient Scientific Workflows Scheduling Strategy on Multi-Cloud Systems,” *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2909–2919, Oct. 2022, doi: 10.1109/TCC.2021.3057422.

[128] Y. Liu, Z. Wang, and D. Zhou, “Resilient Actuator Fault Estimation for Discrete-Time Complex Networks: A Distributed Approach,” *IEEE Trans. Automat. Contr.*, vol. 66, no. 9, pp. 4214–4221, Sep. 2021, doi: 10.1109/TAC.2020.3033710.

[129] A. Sheeba and B. Uma Maheswari, “An efficient fault tolerance scheme based enhanced firefly optimization for virtual machine placement in cloud computing,” *Concurrency and Computation*, vol. 35, no. 7, p. e7610, Mar. 2023, doi: 10.1002/cpe.7610.

[130] G. Chen, N. Guan, K. Huang, and W. Yi, “Fault-tolerant real-time tasks scheduling with dynamic fault handling,” *Journal of Systems Architecture*, vol. 102, p. 101688, Jan. 2020, doi: 10.1016/j.sysarc.2019.101688.

[131] L. Anand, D. Ghose, and V. Mani, “ELISA: An estimated load information scheduling algorithm for distributed computing systems,” *Computers & Mathematics with Applications*, vol. 37, no. 8, pp. 57–85, Apr. 1999, doi: 10.1016/S0898-1221(99)00101-7.

[132] H. Topcuoglu, S. Hariri, and Min-You Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002, doi: 10.1109/71.993206.

[133] Y. Samadi, M. Zbakh, and C. Tadonki, “E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing,” in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, Orleans: IEEE, Jul. 2018, pp. 601–609. doi: 10.1109/HPCS.2018.00100.

[134] H. Mahmoud, M. Thabet, M. H. Khafagy, and F. A. Omara, “An efficient load balancing technique for task scheduling in heterogeneous cloud environment,” *Cluster Comput*, vol. 24, no. 4, pp. 3405–3419, Dec. 2021, doi: 10.1007/s10586-021-03334-z.

[135] A. Benoit, M. Hakem, and Y. Robert, “Fault tolerant scheduling of precedence task graphs on heterogeneous platforms,” in *2008 IEEE International Symposium on Parallel and Distributed Processing*, Miami, FL, USA: IEEE, Apr. 2008, pp. 1–8. doi: 10.1109/IPDPS.2008.4536133.

[136] Y. M., “A Survey of Cloud Computing Fault Tolerance: Techniques and Implementation,” *IJCA*, vol. 138, no. 13, pp. 34–38, Mar. 2016, doi: 10.5120/ijca2016909055.

[137] M. Sharma, M. Sharma, S. Sharma, and A. Kumar, “Flow Shop Scheduling Problem of Minimizing Makespan with Bounded Processing Parameters,” in *Soft Computing for Problem Solving 2019*, vol. 1138, A. K. Nagar, K. Deep, J. C. Bansal, and K. N. Das, Eds., in Advances in Intelligent Systems and Computing, vol. 1138. , Singapore: Springer Singapore, 2020, pp. 171–183. doi: 10.1007/978-981-15-3290-0_14.

[138] M. B. Shareh, S. H. Bargh, A. A. R. Hosseinabadi, and A. Slowik, “An improved bat optimization algorithm to solve the tasks scheduling problem in open shop,” *Neural Comput & Applic*, vol. 33, no. 5, pp. 1559–1573, Mar. 2021, doi: 10.1007/s00521-020-05055-7.

[139] J.-P. Arnaout, “A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times,” *Ann Oper Res*, vol. 285, no. 1–2, pp. 273–293, Feb. 2020, doi: 10.1007/s10479-019-03138-w.

[140] A. Al-Rahayfeh, S. Atiewi, A. Abuhussein, and M. Almiani, “Novel Approach to Task Scheduling and Load Balancing Using the Dominant Sequence Clustering and Mean Shift Clustering Algorithms,” *Future Internet*, vol. 11, no. 5, p. 109, May 2019, doi: 10.3390/fi11050109.

[141] N. Garg, D. Singh, and M. S. Goraya, “Energy and resource efficient workflow scheduling in a virtualized cloud environment,” *Cluster Comput*, vol. 24, no. 2, pp. 767–797, Jun. 2021, doi: 10.1007/s10586-020-03149-4.

[142] R. Sookhtsaraei, M. Iraji, J. Artin, and M. S. Iraji, “Increasing the quality of services and resource utilization in vehicular cloud computing using best host selection methods,” *Cluster Comput*, vol. 24, no. 2, pp. 819–835, Jun. 2021, doi: 10.1007/s10586-020-03159-2.

Publications:

- Mushtaq, Sheikh Umar, Sophiya Sheikh, and Sheikh Mohammad Idrees. "*Next-Gen Cloud Efficiency: Fault-Tolerant Task Scheduling with Neighboring Reservations for Improved Cloud Resource Utilization*." *IEEE Access* (2024). ***(Impact Factor: 3.9)***
  https://ieeexplore.ieee.org/document/10537159

- Published review paper entitled "*In-depth analysis of fault tolerant approaches integrated with load balancing and task scheduling*" in Peer to Peer Networking and Applications. **(*Impact Factor: 3.3*)**
  https://link.springer.com/article/10.1007/s12083-024-01798-5
- Mushtaq, Sheikh Umar, Sophiya Sheikh, and Ajay Nain. "The Response Rank based Fault-tolerant Task Scheduling for Cloud System." *2023 1st International Conference on Advanced Informatics and Intelligent Information Systems (ICAI3S 2023)*. Atlantis Press, 2024.
- Mushtaq, Sheikh Umar, and Sophiya Sheikh. "A fault-tolerant resource reservation model in cloud computing." *Recent Advances in Computing Sciences*. CRC Press, 2023. 295-301.

Conferences Presented:

- Participated in the International conference titled "1st International Conference on Recent Advances in Computing Sciences (RACS-2022) (4th – 5th Nov 2022), organized by the School of Computer Application at Lovely Professional University, Punjab.
- Participated in 1st International Conference on Futuristic Computation Technique: Approaches, Implementations and Applications (ICFCT-2022) (16th – 17th Dec 2022), organized by Panipat Institute of Engineering and Technology, Haryana.
- Participated in 1st International Conference on Advanced Informatics and Intelligent Information Systems (ICAI3S 2023) (29th – 30th Nov 2023) Organised by Department of Informatic UPN "Veteran" Yogyakarta, Indonesia.

Patent Publications:

- Published Patent entitled "**Novel Neighbouring and Reservation based fault tolerant Dynamic Scheduling in Cloud Environments**"
  Application Number: 202311069888
  Inventor: Sheikh Umar Mushtaq, Sophiya Sheikh
  Status: Under Examination for Grant
  Publication Date: 24/11/2023

- Published Patent entitled "**QoS aware Clustered Task Scheduling for Cloud environment**"
  Application Number: 202311079300
  Inventor: Sheikh Umar Mushtaq, Sophiya Sheikh
  Status: Published
  Publication Date: 29/12/2023

- Published Patent entitled "**An Efficient Load-Balancing System for Optimizing QoS in Cloud Computing Environments**"
  Application Number: 202411052036
  Inventor: Sheikh Umar Mushtaq, Sophiya Sheikh
  Status: Published
  Publication Date: 26/07/2024

Copyrights:

- Copyright entitled "**An Analysis of Scheduling Techniques with Fault Tolerance and Load Balancing in Cloud Computing"**
- Copyright entitled "**A Structured Roadmap to Address Reliability and QoS Challenges in Cloud"**

Pipeline Work:

- Extended version of **CRFTS**
- Copyright Submitted entitled "**Organization to in-depth analysis of hybrid models of fault tolerance and load balancing in the cloud**"