

FAULT AWARE LOAD BALANCING AND LEARNING BASED RESOURCE ALLOCATION IN CLOUD

Thesis submitted in fulfillment of the requirements for the Degree of

Doctor of Philosophy

By

Punit Gupta

under the supervision of

Prof. Dr. S. P. Ghrera



Department of Computer Science and Engineering

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,
Waknaghat, Solan, H.P**

September 2016

DECLARATION

I hereby declare that the work reported in the Ph.D. thesis entitled “**FAULT AWARE LOAD BALANCING AND LEARNING BASED RESOURCE ALLOCATION IN CLOUD**” submitted at **Jaypee University of Information Technology, Waknaghat, Solan, H.P., India**, is an authentic record of my work carried out under the supervision of **Prof. Dr. S.P. Ghrera**. I have not submitted this work elsewhere for any other degree or diploma. I am fully responsible for the content of my Ph.D Thesis.



Punit Gupta

Department of Computer Science and Engineering.

Jaypee University of Information Technology, Waknaghat, Solan, H.P., India.

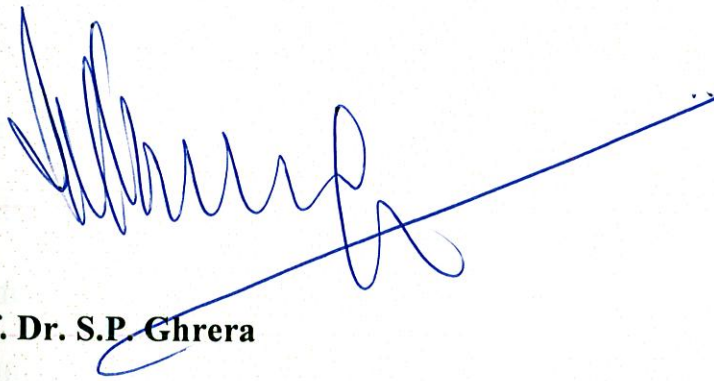
Date

@Copyright :2016 by Punit Gupta

“The copyright of this thesis rests with the author. No quotations from it should be published without the author’s prior written consent and information derived from its should be acknowledged.”

SUPERVISOR'S CERTIFICATE

I hereby declare that the work reported in the Ph.D. thesis entitled "**FAULT AWARE LOAD BALANCING AND LEARNING BASED RESOURCE ALLOCATION IN CLOUD**" submitted by **Punit Gupta** at **Jaypee University of Information Technology, Waknaghat, Solan, H.P., India** is a bonafide record of his original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.



Prof. Dr. S.P. Ghrera

Department of Computer Science and Engineering.

Jaypee University of Information Technology, Waknaghat, Solan, H.P., India.

Date 11th Feb 2017

Acknowledgments

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever. My deepest gratitude is to my advisor, Prof. Dr. S.P. Ghrera. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own and at the same time the guidance to recover when my steps faltered. His patience and support helped me overcome many crisis situations and finish this dissertation.

DPMC members, Dr. Hemraj Saini, Dr. Yashwant, Dr. Rajiv Kumar have been always there to listen and give advices. I am deeply grateful to all of them for the long discussions that helped me sort out the technical details of my work. I am also thankful to Prof. P.K. Gupta for encouraging the use of proper timeline and for carefully reading and commenting on writing this manuscript. I am grateful to my colleges for their encouragement and practical advices. I am also thankful to them for reading my reports, commenting on my views and helping meunderst and enrich my ideas.

My sincere thanks to the undergraduate students and staff of the University, for their various forms of support during my graduate study.

Many friends have helped me stay sane through these difficult years. Their support and care helped me overcome setbacks and stay focused on my graduate study. I greatly value their friendship and I deeply appreciate their belief in me. Most importantly, none of this would have been possible without the love and patience of my family, has been a constant source of love, concern, support and strength all these years.

Finally, I appreciate the financial and academic support of the Jaypee University of Information technology.



Punit Gupta

Dept. of Computer Science

Jaypee University of Information Technology

Waknaghat, Solan, H.P.

Table of Contents

	PAGE NO.
DECLARATION	v
SUPERVISOR'S CERTIFICATE	vi
ACKNOWLEDGEMENT	vii
ABSTRACT	xi
LIST OF ABBREVIATIONS	xii
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
1. INTRODUCTION	1-10
1.1. Characteristics of cloud	2
1.2. Business Models	3
1.3. Issues in Cloud Computing	4
1.3.1. Resource Allocation	4
1.3.2. Load Balancing	5
1.3.3. Migration	6
1.3.4. Power efficient resource allocation and load balancing algorithms	6
1.3.5. Cost efficient resource allocation and load balancing algorithms	6
1.3.6. Fault tolerant algorithms	7
1.3.7. Behavior-based algorithms	7
1.3.8. Trust management	7
1.4. Problem statement	9
1.5. Parameters Used	9
1.6. Performance Parameters	10
1.7. Outline of the Thesis	10
2. LITERATURE REVIEW	12-24
2.1. Resource Allocation	12
2.2. Load Balancing Algorithms	15

2.3. Power Efficient Algorithms	16
2.4. Cost Efficient Algorithms	19
2.5. Behavior based algorithm	21
2.6. Trust Models	22
3. FAULT AND DEADLINE AWARE LOAD BALANCING	25-50
3.1. Approach 1: Fault and Load Aware Load Balancing in Cloud Storage	27-36
3.1.1. Problem Statement	26
3.1.2. Problem Approach	27
3.1.3. Proposed Model	30
3.1.4. Experiment and Result	33
3.2. Approach 2: Deadline Aware Load Balancing of Distributed Servers in Distributed	36-45
3.2.1. Proposed Approach	36
3.2.2. Proposed Model	39
3.2.3. Experiment and Result	42
3.3 Approach 3: Load Balancing Algorithm for Hybrid Cloud IaaS	45-50
3.2.1. Proposed Approach	46
3.3.2. Experiment and Results	48
3.4. Conclusion	50
4. LEARNING BASED FAULT WARE RESOURCE ALLOCATION	51-91
4.1. Approach 1: Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure	52-61
4.1.1. Proposed algorithm	52
4.1.2. Experiment and Results	57
4.2. Approach 2: Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure	62-73
4.2.1. Proposed Algorithm	62
4.2.2. Experiment and Results	67
4.3. Approach 3: Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure	74-84
4.3.1. Proposed Algorithm	74

4.3.2. Experiment and Results	80
4.4. Approach 4: Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure	85-89
4.4.1 Proposed Algorithm	85
4.4.2 Experiment and Result	87
4.5. Comparative Analysis of Learning Based Algorithms	89
4.6. Conclusion	91
5. FAULT AWARE POWER EFFICIENT SCHEDULING	92-103
5.1. Approach 1: Power and Fault Aware Reliable Resource Allocation for Cloud Infrastructure	93-97
5.1.1. Proposed algorithm	94
5.1.2. Experimental and Results	95
5.2. Approach 2: Trust and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization	98-103
5.2.1. Proposed Algorithm	100
5.2.2. Experiment and Results	101
5.3. Conclusion	103
6. CONCLUSION AND FUTURE SCOPE	104-105
6.1. Conclusion	104
6.2. Future Scope	105
7. REFERENCES	106-115
8. LIST OF PUBLICATIONS	116-117

List of Tables

Table No.	Title	Page No.
Table 3.1	Experimental parameters used for simulation environment	35
Table 3.2	Experimental parameters used for simulation environment	45
Table 3.3	Servers Parameters	52
Table 4.1	Experimental parameters used for simulation environment	62
Table 4.2	Experimental parameters used for simulation environment	73
Table 4.3	Experimental parameters used for simulation environment	86
Table 4.4	Server fault rate	93
Table 4.5	Request failure count	94
Table 4.6	Request completion count	94
Table 5.1	Experimental parameters used for simulation environment	102
Table 5.2	Experimental parameters used for simulation environment	109

List of Figures

Figure No.	Title	Page No.
Figure No. 1.1	Cloud system characteristics and properties	2
Figure No. 2.1	Taxonomy for resource allocation in cloud	15
Figure No. 2.2	Energy aware allocation taxonomy	19
Figure No. 2.3	The trust cloud framework.	23
Figure No. 2.4	The logical structure diagram of TCMCS.	24
Figure No. 3.1 (a)	Problem statement for load balancing (a) at time $t=0$, servers receive equal amount of client requests.	29
Figure No. 3.1 (b)	Problem statement for load balancing (b) at time $t=2$, scenario of servers after processing the receive requests.	29
Figure No. 3.2	Organization of distribute storage servers.	30
Figure No. 3.3 (a)	Proposed load balancing algorithm.	33
Figure No. 3.3 (b)	Find a neighbor server algorithm	34
Figure No. 3.4	Proposed algorithms flow diagram	34
Figure No. 3.5	Number of request completed	36
Figure No. 3.6	Number of sent requests vs. No. Of failed requests.	37
Figure No. 3.7	Overall response time.	37
Figure No. 3.8	Average utilization of system	38
Figure No. 3.9	Organization of distribute data servers.	40
Figure No. 3.10 (a)	Load balancing algorithm.	43
Figure No. 3.10 (b)	Find a neighboring server algorithm	43
Figure No. 3.11	Proposed algorithms flow diagram	44
Figure No. 3.12	Comparison between no. Of sent requests vs. No. Of completed requests.	45
Figure No. 3.13	Comparison between no. Of sent requests vs. No. Of postponed requests.	46
Figure No. 3.14	Overall response time.	46
Figure No. 3.15	Average utilization of system	47
Figure No. 3.16	Proposed trust based algorithm	52

Figure No. 3.17	Total number of request completed in faulty environment	53
Figure No. 3.18	Total number of request Failed in faulty	53
Figure No. 4.1	Proposed FGA algorithm initialization	60
Figure No. 4.2	Proposed fault aware genetic algorithm	60
Figure No. 4.3	Proposed FGA valuation phase	60
Figure No. 4.4	Proposed FGA allocation phase	51
Figure No. 4.5	Proposed FGA flow diagram	51
Figure No. 4.6	Comparison of improvement in request completed	53
Figure No. 4.7	Comparison of improvement in request failed	53
Figure No. 4.8	Comparison of failure probability with variable resources	64
Figure No. 4.9	Comparison of failure probability with variable request	64
Figure No. 4.10	Comparison of reliability with variable resources	65
Figure No. 4.11	Comparison of reliability with variable requests	66
Figure No. 4.12	Comparison of execution time with variable resources	66
Figure No. 4.13	Proposed BBC algorithm initialization	70
Figure No. 4.14	Proposed genetic algorithm	70
Figure No. 4.15	Proposed evaluation phase	71
Figure No. 4.16	Big crunch phase	71
Figure No. 4.17	Allocation phase	71
Figure No. 4.18	Proposed big bang big crunch algorithm flow diagram	72
Figure No. 4.19	Comparison of improvement in execution time	73
Figure No. 4.20	Comparison of improvement in execution time with changes in population size.	74
Figure No. 4.21	Comparison of execution time of individual requests. For 1000 request count	74
Figure No. 4.22	Comparison of execution time of individual requests. For 1500 request count	75
Figure No. 4.23	Comparison of execution time of individual requests. For 2000 request count	75
Figure No. 4.24	Comparison of execution time of individual requests. For 2500 request count	76
Figure No. 4.25	Comparison of execution time of individual requests. For 3000 request count	76

Figure No. 4.26	Comparison of execution time of individual requests. For 3500 request count	77
Figure No. 4.27	Comparison of Average Start time of system with increase in request count.	77
Figure No. 4.28	Comparison of Average Finish time of system with increase in request count	78
Figure No. 4.29	Proposed FBBC algorithm initialization	82
Figure No. 4.30	Proposed FBBC algorithm	83
Figure No. 4.31	Proposed FBBC evaluation phase	83
Figure No. 4.32	Get Fittest with least difference from Center of mass	83
Figure No. 4.33	Get Fittest with least fitness value	84
Figure No. 4.34	Big crunch phase	84
Figure No. 4.35	Proposed FBBC allocation phase	84
Figure No. 4.36	Proposed fault aware big bang big crunch algorithm	85
Figure No. 4.37	Comparison of improvement in scheduling time	86
Figure No. 4.38	Comparison of improvement in request failed	87
Figure No. 4.39	Comparison of improvement in request completed	87
Figure No. 4.40	Comparison of failure probability with variable request count	88
Figure No. 4.41	Comparison of reliability with variable request count	88
Figure No. 4.42	Comparison of execution time with variable request count	89
Figure No. 4.43	Proposed fault aware honey bee algorithm	92
Figure No. 4.44	Comparison of request failure count.	93
Figure No. 4.45	Comparison of request completed count.	94
Figure No. 4.46	Comparison of scheduling delay	95
Figure No. 4.47	Comparison of failed request count	95
Figure No. 4.48	Comparison of completed request count	96
Figure No. 5.1	Proposed PFARA algorithm initialization	100
Figure No. 5.2	Proposed PFARA algorithm resource allocation	101
Figure No. 5.3	Power consumption	102
Figure No. 5.4	Comparison of request failure count	103
Figure No. 5.5	Comparison of request completed count	103
Figure No. 5.6	Proposed PFARA algorithm initialization	107

Figure No. 5.7	Proposed TDARPA algorithm (2)	108
Figure No. 5.8	Comparison of request completed	109
Figure No. 5.9	Comparison of request failed	110
Figure No. 5.10	Comparison of power consumed in Kwh	110

CHAPTER 1

INTRODUCTION

Cloud computing is a most widespread and popular form of computing, promising high reliability for customers and providers both at the same point of time from many fields of sciences or industry. Clients from the different field are served by datacenters in cloud environment geographically spread over the world. Cloud serves a large number of requests coming from various sources over datacenter with high power consumption. However, to provide such a large computing power required a huge power, leading to high power consumption and cost. Request types in cloud system also affect the services which are public and private requests whose proportion is random in nature. A survey in 2006 over the performance of cloud environment in the USA shows datacenter consumed 4.5 billion kWh units of power, which is 1.5% of total power consumed in the USA and this power requirement is increasing 18% every year [1]. In general, cloud computing deals with various issues like poor resource utilization and load balancing and many more. Some of the issues are discussed as follows: 1) as cloud computing tools are used by industry and they have issues with the rapidly growing request and a number of servers deployed, increasing the power consumption. 2) Task allocation of request among datacenter without having knowledge of QoS provided by servers. 3) Current task allocation algorithms only focus on balancing the request and improve utilization of the system but not the failure probability of system. 4) High loaded data centers have high failure probability and due to high load, this may lead to slow down of datacenter and poor QoS (Quality of service) to the client and client provider. 5) While few of the servers are overloaded and some of them are idle or under loaded. 6) Some request needs to be computed with QoS but due to high load and fault rate they may not get the QoS promised which is not appropriate to the user and will be a critical

issue. 7) As per recent study [2-5], utilization of data centers is a major problem because 60% data centers are idle and most of 20% data centers are utilized and waste of the resources.

This shows the poor utilization of resources but this shows the importance of a new approach that has sufficient strategy to minimize wastes of resources and increasing reliability by allocating task over resources which in the case of Cloud is VM with low failure probability to provide high QoS to users. The existing algorithms only take into consideration cloud as non-faulty in nature and fail to provide specific QoS when a fault occurs. So to overcome these issues and improve the performance of the system, we have proposed approaches for resource load balancing and allocation. Figure 1.1 shows cloud computing features, type and various other properties [6]

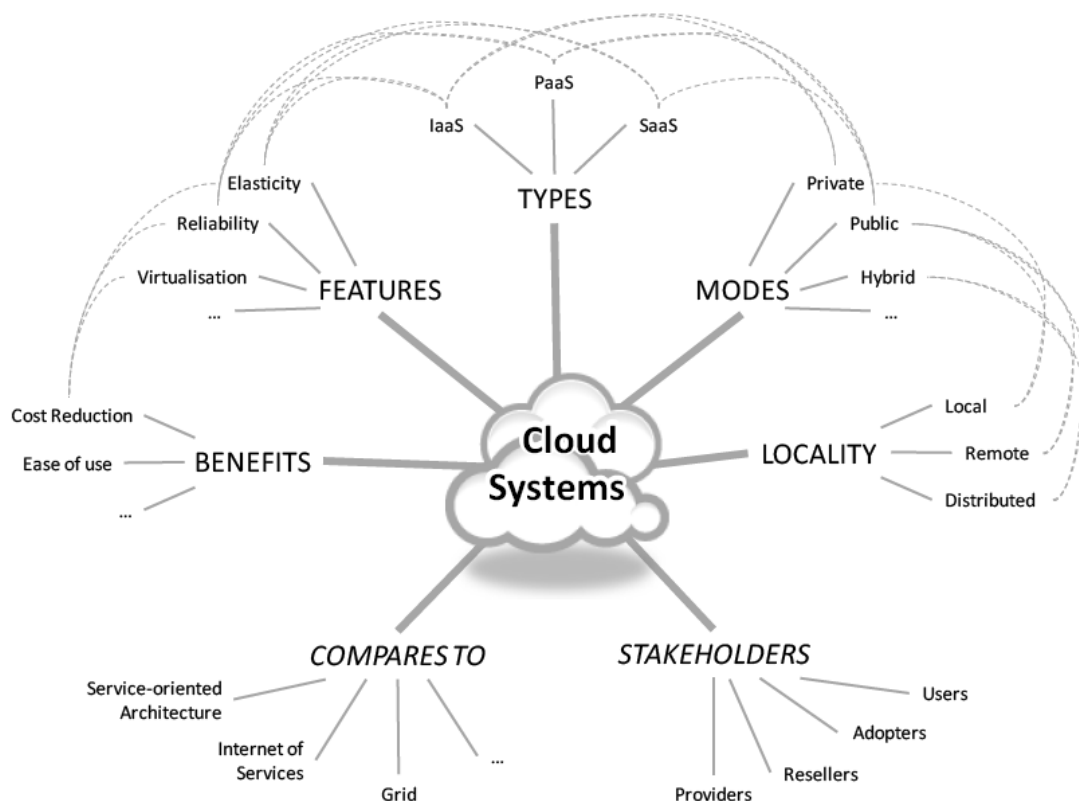


Figure 1.1 Cloud system characteristics and properties

1.1. Characteristics of Cloud

Cloud is a distributed environment, where the servers are placed at various geographical locations but seems to a user as a single entity. Cloud computing provides better performance than any other distributed system like Grid computing or cluster computing and

many more. There are various characteristics of cloud computing which makes it superior than any other system which are as follows [6, 17];

a) High Availability

One of the most important features of the cloud is all time availability of resources in form of storage, computational capability and high network resources. This property also states that the resources are available in overloading conditions also.

b) Pay per use model

This feature made cloud computing popular in the industry due to affordable nature of cloud by an industry with high infrastructure or a business holder with the small requirement can easily manage and have its own infrastructure and high computing system at a low cost. Cloud computing allows a user to pay for only those resources, which are used by him for that specific period of time rather than purchasing a complete server or private infrastructure.

c) Elasticity

Cloud is said to be flexible and scalable at the same time. This feature allows the cloud to scale its resources up or down based on the user or business needs for a period of time. This allows the cloud to have high availability under overloaded condition also and provided uninterrupted services to the user without failure and high quality of service.

d) Reliability

Cloud computing ensures to provide high reliable computing services and resources to the user which means that the user will be provided with uninterrupted services with the quality of services as assured to the client.

1.2. Business Models

Cloud computing provides various service driven business models to provide a different level of computation to the users. Cloud computing provides 3 type of service models listed as:

Software as a Service (SaaS), platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Our work focuses on improving the performance of cloud infrastructure as a service in a faulty cloud environment [17]. The fault is a behavior of every distributed system because fault may occur any time that may be due to system failure, network failure or disk failure.

1.3. Issues in Cloud Computing

Cloud computing deals with various issues to maintain above discoursed characteristics and quality of serves assured to the user by cloud providers in term of high resource availability, computational capability [7-8]. Some or the issues dealing with resource management, resource scheduling, and managing system performance are discoursed below.

- Resource allocation
- Load balancing
- Migration
- Power efficient resource allocation and load balancing algorithms
- Cost efficient resource allocation and load balancing algorithms
- Fault tolerant algorithms
- Behavior-based algorithms
- Trust management

1.3.1. Resource Allocation

Resource Allocation strategy (RAS) in the cloud is all about the scheduling of tasks or requests by cloud provider in such a manner to balance the load over all the servers and provide high Quality of Service to clients. It also includes the time required to allocate the resources and the resources available. The main aim is to improve the utilization of resources and complete all the request within the deadline and with least execution time [9].

An optimal RAS should avoid the following criteria as follows:

- a) *Resource contention* situation arises when two applications try to access the same resource at the same time.
- b) *The scarcity of resources* arises when there are limited resources.
- c) *Resource fragmentation* situation arises when the resources are isolated
- d) *Over-provisioning* of resources arises when the application gets surplus resources than the demanded one.
- e) *Under-provisioning* of resources occurs when the application is assigned with fewer numbers of resources than the demand.

Resource allocation algorithm can be categorized into three subcategories as from the literature review conducted over existing proposed algorithms.

Categorization is as follows:

- 1) Static
- 2) Dynamic
- 3) Learning-based.

Static scheduling algorithms are referred to algorithms which are not affected by system and behavior of cloud some of the algorithms line SJF, FCFS, Round robin etc [11]. On the other hand, dynamic algorithms are those whose objective function depends on the system parameters line deadline, available resources, resource utilization of host and many more example of these algorithms is a deadline-based algorithm, cost-based algorithm, utilization based algorithm [5-10]. The problem with these algorithms is that they do not take into consideration the previous performance of host and system as a whole. Moreover, the past faulty nature of the system is not taken into consideration and leads to large request failure. Dynamic algorithms deal with the issue of local minima these algorithms are not able to find a global best solution and stuck in local best solution.

1.3.2. Load Balancing

Load balancing aims to distribute load across multiple resources, such as server, a server cluster, central processing. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource

Goal of Load Balancing [12] are as follows:

- 1) To improve the performance substantially.
- 2) To improve system stability.
- 3) To have scalability in the system.
- 4) To improve the system condition under high load or request rate.

Types of Load balancing algorithms [13]

Sender-Initiated: When load balancing algorithm is triggered by the sender.

Receiver Initiated: When load balancing algorithm is triggered by the receiver.

Symmetric: It is the combination of both sender initiated and receiver initiated.

Load balancing is also used to manage the average utilization of the system as a whole to avoid creation of hot spots i.e. the request should not be clustered on a single datacenter rather should be spread over the servers. So it aims to find an underloaded server and move the requests to that selected server. This makes a requirement of a load balancing algorithm to fulfill these requirements taking into consideration system utilization and quality of service without failure.

1.3.3. Migration

Migration in cloud infrastructure plays an important role in cloud Infrastructure under system overloading condition. In cloud infrastructure when the server gets overloaded i.e., the utilization is beyond a threshold is considered to be overloaded, in such condition we need to migrate a virtual machine from overloaded server to an under loaded or neutral server [14]. This helps to balance the load and prevent the server from any failure. So there is a requirement of an intelligent and efficient migration algorithm to balance the condition and improve the performance of the system.

1.3.4. Power efficient resource allocation and load balancing algorithms

The power efficiency of a cloud environment is an important issue for a green cloud environment. As 53% of the total expense of a datacenter is spent on cooling i.e. power consumption [15].

In a survey in 2006 on datacenters established U.S consumed more than 1.4% of total power generated during the year [16]. Therefore we require improving the power efficiency of infrastructure. The problem can be solved in various ways and various proposals have been made to solve and improve the performance. So to do this we need to design power-aware resource allocation and load balancing algorithm to improve the total power consumption of the system and any such algorithm will result in a reduction of overall power consumption.

1.3.5. Cost efficient resource allocation and load balancing algorithms

Cloud computing uses pay-per-use model to ensure least cost and payment only for the resources used. To maintain this feature cloud controller algorithms like resource allocation migration and load balancing are responsible for maintaining this characteristic by offering the resources which can complete the client request on time and within the budget of client and have least cost that can be offered. So we require cost aware algorithm which are cost efficient and can provide the best system performance by improving utilization and

power consumption all at the same time [10, 15]. These type of algorithm are referred to as multi-objective algorithms, there are many proposals made for improving the performance of the system but they only take into consideration either power or cost, so cannot guarantee the best performance.

1.3.6. Fault tolerant algorithms

Cloud computing environment is a type of distributed environment like grid computing and cluster computing. Existing algorithms consider cloud as nonfaulty but faults are a part of distributed environment which may be due to hardware or software failure at any point of time [93, 99, 100]. There are many fault aware and fault prediction algorithms been proposed for grid environment to improve the reliability of the system. So similarly we require fault aware algorithms to make system fault aware reduce the failure probability of the system and increase the reliability of the system.

1.3.7. Behavior-based algorithms

Most of the resource allocation and load balancing algorithm proposed for cloud infrastructure are dynamic algorithms like min-min, max min and many more. These algorithms take into consideration only the current behavior \ status or the server and system for selection of server. The problem with these algorithms is that they do not take into consideration the previous performance of the system for prediction of the better solution rather than stuck in local minima. Behavior-based algorithm lists genetic algorithm, ant colony, particle swan optimization, monkey search and many more. So there is a need of algorithms taking into consideration the previous and present performance of the system for decision making.

1.3.8. Trust management

Trust models are been used in all form of distributed environments ranging from MANETS (Mobile ad hoc network), Sensor network and Grid computing to validate the reliability of nodes over distributed network. In grid computing, various trust models are been proposed to ensure trust in term of security and reliability of the server or the node. Trust models are to resolve the problem of reliability in any heterogeneous environment, which contributed of nodes having different configuration spread over a network. There are many models being proposed in a cloud computing environment.

What is Trust?

Trust can be defined as an entity based on reliability and firm belief based on an attribute of the entity. Trust is the firm belief in the competence of an entity to act as expected, such that this firm belief is not a fixed value associated with the entity, but rather it is subjected to the entity's behavior and applies only within a specific context at given time [18]. The definition simply means that trust is a variable changing believe, based on both static and dynamic parameters.

Trust can also be defined as “the subjective probability by which an individual expects that another individual performs a given action on which its welfare depends” [19-20].

Trust can be categorized into three major classifications which are as follows [21]:

- a) *Blind trust*: This is the default trust before any event in the system, and which would include an agent to initiate a relationship with unknown entities.
- b) *Conditional trust*: This is a classic state of trust during the life of the agent. This condition trust is likely to evolve, and can be subject to some sets of constraints or condition.
- c) *Unconditional trust*: Such a trust is the probability be configured directly by an administrator, and would not be sensitive to successful/unsuccessful interaction and external recommendation of any other sources of evolution of the conditional trust.

1.4. Problem statement

The aim of this work is to make system fault tolerant and more reliable computing system with improved performance in cloud infrastructure environment. A number of algorithms have been worked out for long period of time but they assumed cloud as non-faulty. So in our work, we have proposed various fault tolerant algorithms to resolve various issues as follows:

- 1) To design a fault and deadline aware load balancing algorithms for private and hybrid cloud, which aim to improve QoS of load balancing algorithm and minimize the faults, resource utilization, minimize response time and avoiding overloading of any single resource in cloud.
- 2) To design learning based fault aware resource allocation algorithms, to provide a global best schedule with least scheduling time complexity.
- 3) Designing fault aware and power-efficient scheduling algorithms for improving power efficiency and request failure count in the cloud.

1.5. Parameters Used

Fault rate: defined as the total count of request failed over a period of time T

Failure Probability: as the probability of request to fail on a specific host or system.

Reliability of a system: This feature of a system can derived from the failure probability of system which can be defined as:

$$Reliability \% = 100 - Failure_Probability \% \quad (1.1)$$

Power Efficiency: The ratio of the output power over the input power i.e. the percentage power consumed over a period of time.

Utilization: this is the capability of the host to be used out of total available resources.

Average Resource Utilization: This is an average of utilization of resources over the whole system i.e. all available hosts.

Average start time: Average waiting time of request before been scheduled or allocated.

Average Finish time: Average of finishing time of all the request executed by system.

Scheduling Delay: Total time to find a suitable resource for a set of tasks.

MakeSpan: Total execution time of system including scheduling delay for set of requests /Task

1.6. Performance Parameters

To study the performance of proposed algorithm over existing algorithm we require to compare these parameters listed below:

Average utilization: Average utilization is the average percentage of time during which the server is busy processing jobs during a simulation

Power Utilization: Power utilization can be defined as the power consumed in kWh during the simulation.

Average queue length: This is the average size of the queue of a server during a simulation.

Request failure count: Total count of requests failed during a simulation.

Request completion count: Total count of requests Completed during a simulation.

Average start time: This the average of the start time of all the requests generated during the simulation.

Average finish time: This the average of finish time of all the requests generated during the simulation.

Scheduling delay: The times taken to find a suitable server for a set of given requests.

Makespan: The time taken to complete all requests over a given cloud environment.

Failure probability: the probability of failure of each request is a given system.

Reliability: Reliability of a system can be defined as the probability of the system being reliable, which can be defined as $(1 - \text{failure probability})$.

1.7. Outline of the Thesis

The thesis has been organized into 6 chapters and out of that CHAPTER 1 presents Introduction comprises problem statement and various issues in cloud computing. CHAPTER 2 presents the existing proposed work to solve the problem of resource allocation and load balancing with different combinations of authors taking different performance matrices. This chapter also discussed the various methods power efficient, cost efficient algorithms and algorithms inspired by nature. The preliminary notations are introduced to keep the clarity of usage throughout the thesis. CHAPTER 3 presents three techniques for load balancing for cloud storage in faulty environment to improve the fault tolerant behavior and reliability of cloud. The approaches are proposed for private and hybrid cloud environment. CHAPTER 4 presents a set of learning based techniques for the faulty cloud to find global best solution and shows improvement in scheduling delay, failure count, failure probability and the reliability of the system, CHAPTER 5 presents a set of power efficient and fault aware approaches inspired from nature like honey bee and ant colony algorithms to find best suitable resource. Finally, followed by the conclusion and future scope of the research work for further research are provided in CHAPTER 6.

CHAPTER 2

LITERATURE REVIEW

2.1. Introduction

In this section, we have discoursed various existing approaches from the field of resource allocation, load balancing, green computing and trust management. This section aims to identify the research gaps and focus on the current state of artwork in the field of resource allocation, load balancing, cost efficiency and green computing.

2.2. Resource Allocation

Many researchers have done research and introduce us some beneficial and optimal scheduling algorithm. [22] Proposed a modified Min-Min algorithm, this chooses the task with least completion time and schedule to serve accordingly. Author has proposed load balancing Min-Min algorithm which having basic properties of Min-Min algorithm and consider minimizing completion of all request. In this proposal three level of service models are used.

1. Request manager- To take request and forward to Service managers.
2. Service manager- various manger works or task and dispatch them to respective service node.
3. Service Node- Service node provide service to request which came to request mode

They have merged two approaches (OLB Opportunistic load balancing and load balance min-min) scheduling algorithms in this model. The main focus of combined approaches is to

distribute the request or dispatched task basis of their completion time to suitable service node via an agent. This approach not saying about main system, suppose if request are somehow moving or scheduled in the same server and due to lots of load sever need more power to complete these request and more physical heat will generate and to stop heating system need an external cooling system which also lead to extra power source and one more important thing is due to overheating system performance slow down The same way [23] proposed and another algorithm for task scheduling, this paper proposed VM resource allocation basis on genetic algorithm to avoid dynamic VM migration to completion of request. They have proposed a strategy to share or allow resource equally to VM so it can work fast and minimize response time to subscribe. They also proposed hotspot memory (virtual memory) assignment and dispose that after completion of request via remapping of VM migration. Here VMware distribution tool is used to schedule computation work in a virtual environment. As genetic algorithm characteristics is to find best, fittest VM in terms of Cloud computation.

This paper checks fitness of each VM and schedule task accordingly. When creating a VM a process executes to create that and increase process work that also lead to more process and increase energy consumption. Hu, Jinhua et al.[24] Proposed another scheduling algorithm, this paper proposed an approach for collective collaborative computing on trust model. The trust value taking as a factor for task scheduling, trust value mutually took from consumers as well service provider, which make it fail free execution environment. Here they have proposed a mathematical equation to calculate the Reputation point which enhances the reputation of VM in terms of fast execution and type of task. If the reputation of VM is high them more task allocation will be happening to that VM. To calculate Reputation many factors have to consider which also reflect QoS of cloud computing. This paper also proposed a way to serve a request reliability, as well trust management with a reputation of VM factor which are lead to trustworthy. Trust has calculated by a mathematical equation and schedule accordingly.

Hu, Jinhua et al. [25] proposed a live VM migration algorithm, this paper proposed a method for VM live migration with various resource reservation system. VM migration is taking place on the basis of source machine load, if the load is high then it can wear, during execution of the request it migrates the VM to another server or data centers to complete the task without interruption for better performance. Resource reservation done both sides, i.e.,

Source machine and target machine as well will in such manner CAP (maximum availability of CPU) allocate them and adjust memory resource dynamically. At the end of target machine, they properties time bound program which will keep monitoring for cup resource utilization. Memory Reservation done by allocating crating certain number of VM and when the migration process comes into existence these VM got shut down to evacuate the space to migrate VM. Sometime it may be possible that target machine not having enough space to migrate in such condition that physical machine should remove from candidate machine for migration and which physical machine having the capability or enough space will lead to migrate VM. This paper implemented and simulated using Xen Virtualization.

Barroso et al.[26] This paper proposed an algorithm, dynamic and integrated resource scheduling algorithm for cloud data center which balance load between servers in overall run time of request, here they are migrating an application from one data center to another without interruption. Here they are introducing some measurement to ensure load balancing. They have given a mathematical reputation to calculate imbalance load to calculate average utilization to its threshold value to balance load. To implement DAIRS they have used physical server with physical cluster and Virtual servers with virtual cluster. Application migration saves time instead of migrating whole VM data. Zhanjie Wang [27] proposed an dynamic algorithm for resource allocation in cloud using fuzzy logic and pattern recognition based on power and storage parameters. The propose algorithm is derived from FastBid algorithm. The algorithm tries to improve the network traffic and communication load over the system. The algorithm shows better result than Min- Min algorithm in term of makespan and network load

Parvathy S. Pillai [28] et al. proposed a novel resource allocation algorithm derived from game theory for resource allocation in cloud. In this work author has used uncertainty principle of game theory for allocation of virtual machines in cloud. This work improves the communication cost and resource wastage over the system. Abdullah Yousafzai [29] et al. surveyed and reviewed resource allocation algorithm in cloud. This work contributed an review and comparative study or current state of art cloud resource scheduling and allocation algorithms for cloud. Moreover this article proposes an taxonomy for resource allocation in cloud environment, which shows various ways to solve the issue of resource allocation and different aspects of resource allocation. Figure given below shows the taxonomy.

Many other resource allocation algorithm are been proposed [88 -97] using various dynamic techniques to improve the performance of the system are been studied.

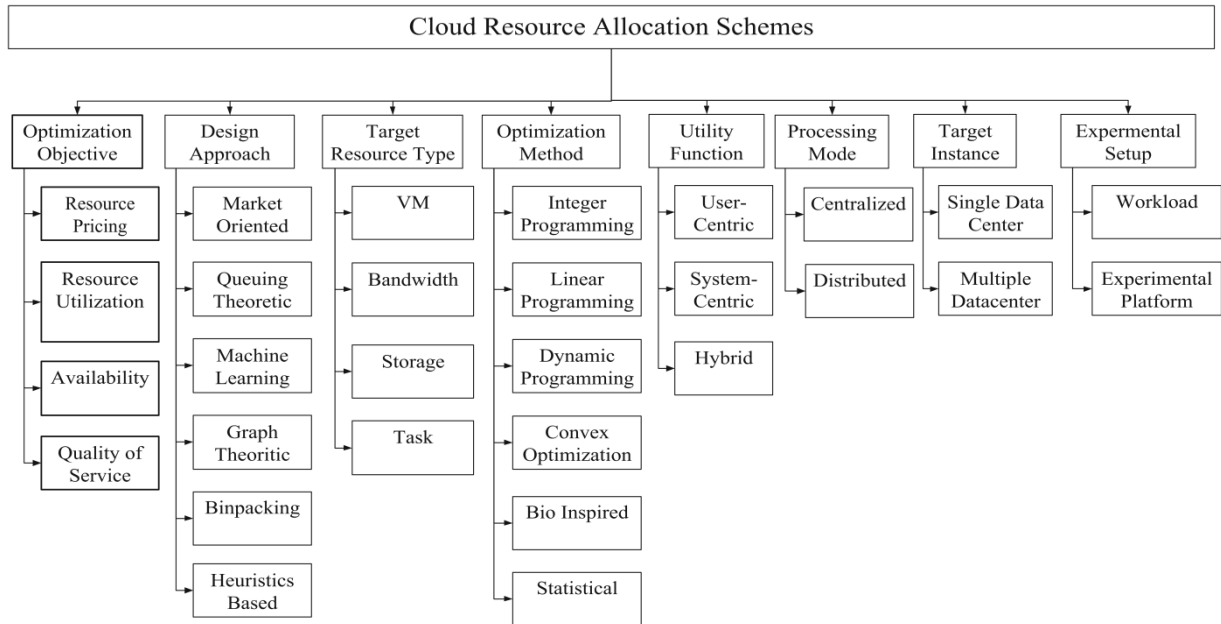


Figure 2.1 Taxonomy for resource allocation in cloud [29]

2.3. Load Balancing Algorithms

Various load balancing tactics have been proposed till now which can also be classified into static and dynamic in nature [30]. Yamamoto et al. [31] proposed a disbursed strategy to balance the load using replication of data. Authors have proposed two replication approaches 1) in the route random replication procedure, replicas saved within the peers along the trail of soliciting for to look. 2) In the course adaptive replication procedure replicas saved most effective within the peers in step with their likelihood of replication. This paper does not use the knowledge about the ability of servers for selection of server.

In [32] Rao et al. have offered a framework for load balancing in distributed environment, named as HiGLOB. Right here, authors have used two principal add-ons 1) histogram supervisor - generates a histogram to preserve a global information regarding the distribution of the load within the system, and 2) load-balancing manager - reallocates the load at any time when the node becomes overloaded or under loaded. Nevertheless, there's overhead associated at the same time setting up and preserving the histograms.

Zeng et al. [33] have proposed a load re-balancing algorithm to work out the crisis of load balancing in distributed environment. They have additionally ensured the reliability of the process where one chunk of a file and two duplicate copies are allocated in three exceptional

different servers at a time. In this algorithm author uses the master server periodically for checking of chunk servers and to differentiate which chunk server is over-loaded and which is not. Nonetheless, this master server turns into a single-point failure for the load balancer. Fan et al. [34] have proposed an adaptive load optimization algorithm (AFLBA) for the Hadoop distributed file procedure which uses two modes: 1) disk utilization expense system and 2) carrier blocking off rate system. The proposed algorithm uses the storage average utilization of each data node and probability of blocking consumer request of each knowledge node. Since this algorithm isn't disbursed so it creates a performance bottleneck node within the HDFS.

Hasio et al. [35] and Chung et al [36] have proposed an improved load balancing algorithm for distributed file system to overcome the issue of bottleneck and improve the performance of system. They have proposed to use CHORD protocol for creation the node server.

Many other load balancing approaches have been proposed to avoid the condition of overloading [101, 102] using max-min, min-min and dynamic strategy.

2.4. Power Efficient Algorithms

Several researchers have introduced various models and methods to conserve energy. Some of them are discoursed below.

Louis Rilling et al. [37] proposed a virtual infrastructure optimization solution using the ant colony optimization algorithm for finding better paths through graphs. The most common approach while performing workload consolidation is that the workload is allotted to a physical machine (e.g. CPU) and those resources which require excessive provisioning are converted into a lower power state.

Osvaldo Adilson de Carvalho Junior et al. proposes the use of a function that can ensure the most appropriate behaviour to the principles of Green IT but not the quality of service. For this he proposes the use of GreenMACC (Meta-scheduling Green Architecture) and its module LRAM (Local Resource Allocation Manager) to automate the execution of all scheduling policies implemented in the Scheduling Policies Module so as to provide Quality of Service in Cloud Computing and determine its flexibility. [38] Task consolidation is an efficient method which is used to reduce power consumption by increasing the resource utilization but due to task consolidation resources may still draw power while being in the idle state. Young Choon Lee et al. has introduced two algorithms to maximize the utilization

of resources of the cloud. The two algorithms are ECTC and MaxUtil. ECTC works on the premise of calculating the energy which is being used by a particular task when there are simultaneous tasks running parallel with it, and then it is compared with the optimal energy which is required. MaxUtil focuses more on the mean usage of a particular task when it is being processed. [39]

Dzmitry Kliazovich et al. presented a simulation environment for data centers to improve their utilization of resources. Apart from working on the distribution of the tasks, it also focuses on the energy used by the data center components. The simulation outcomes are obtained for various architectures of data centers. In [40] Robert Basmadjian et al. proposed the use of proper optimization policies reducing the power usage and increasing the resource utilization without sacrificing the SLAs. He developed a model which worked on incrementing the capability of the processor to process tasks. [41] Zhou Zhou et al proposes a Three Threshold Energy Saving Algorithm [TESA] which has three thresholds to divide hosts between heavy load, light load & middling load. Then based on TESA 5 VM migration policies are suggested which significantly improves energy efficiency. [42].

Dung H Plan et al. proposed GreenMonster protocol which improves renewable energy consumption while maintaining performance by dynamically moving services across IDCs. GreenMonster uses Evolutionary Multi-objective Optimization Protocol [EMOA] to make service placement and migration decisions. [43]. Liang Liu et.al. proposed a new VM architecture which has capabilities of Live Virtual Machine Migration, VM placement optimization and online VM Monitoring. This architecture gives us a considerable energy saving. [44]. Aman Kansal et al. proposes a power metering solution for virtual machines. The proposed solution has a very small runtime overhead and provides accurate and practical information for power capping to improve the energy efficiency of the datacenters. [45].

Abbas Horri et al. [46] proposed a novel approach to improve the power efficiency of system for cloud infrastructure based on the resource utilization history of virtual machines in cloud.

The first work in large-scale virtualized datacenters has been proposed by Nathuji and Schwan [47]. In their proposed method, the resource management is split into local and global managers. Local manager coordinates power management methods of VMs in each host because the authors assumed that VM guests have a power aware OS. Global manager monitors the performance of multiple hosts and selects the appropriate host for requested VM migration. However, in situation that the guest OS is non-power-aware, this power

management method may be inefficient [47]. Salimi and Sharifi in [48] proposed an approach to schedule a set of VMs on a shared PM. The goal of the scheduling algorithm was to minimize the execution times (Makspan) of batch applications running on VMs based on considering the interferences of concurrent VMs. To identify the interference, they first presented an interference model in terms of number of concurrent VMs, processing utilizations of VMs and also the network latency. Nasrin Akhter & Mohamed Othman [49] surveyed and reviewed energy aware resource allocation algorithm in cloud. This paper reviews lasted proposal made for improving the energy efficiency of system. Major contribution this work is the broad study and classification of various ways to improve power consumption in cloud environment. The figure below shows the taxonomy proposed

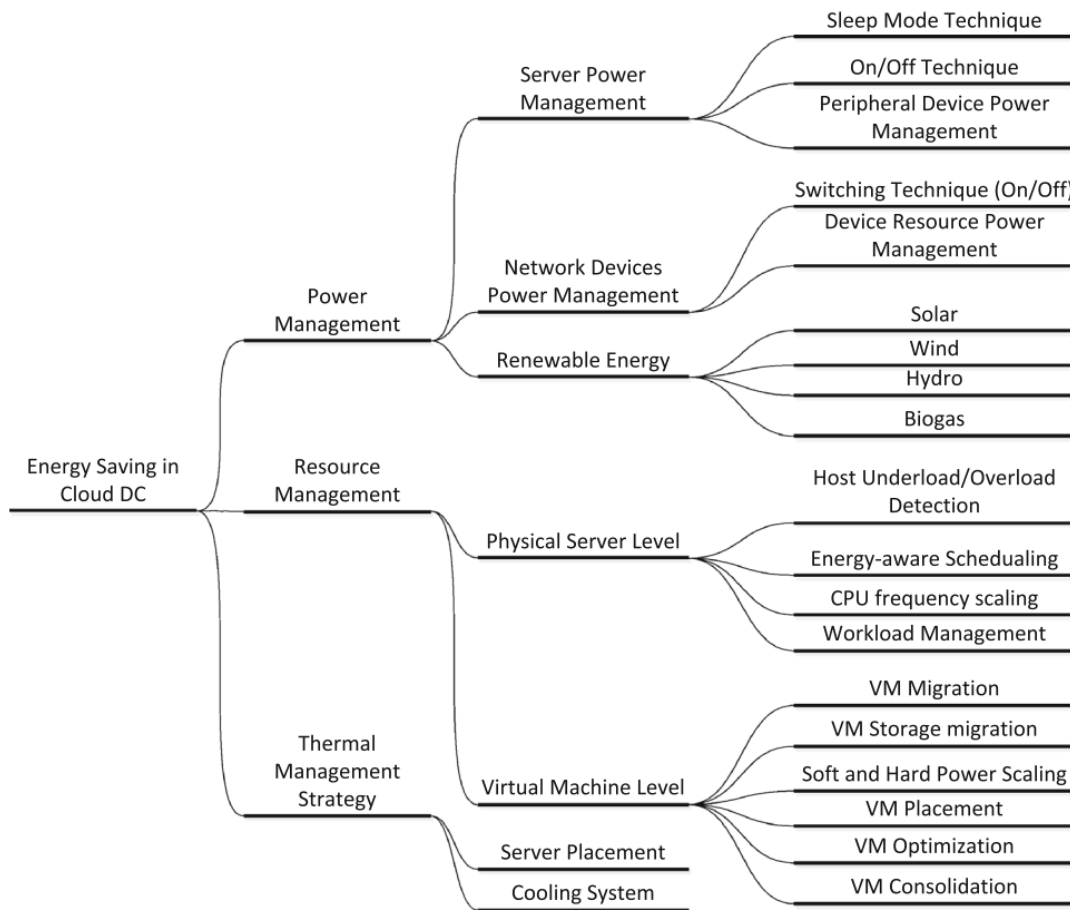


Figure 2.2 Energy aware allocation taxonomy

2.5. Cost Efficient Algorithms

Li Chunlin et al. [50] proposed and const and energy aware resource provisioning algorithm for cloud. This paper presents the cost and energy aware service provisioning scheme for mobile client in mobile cloud. Proposed work proves to be cost optimal and energy efficient

as compares to simply cost aware allocation algorithms. Ehsan Ahvar et al. [51] has proposed an network aware cost optimal algorithm. This algorithm takes into consideration network performance and cost for resource allocation and selection of best server, using artificial algorithm to perform better than typical greedy heuristics. Khaled Metwally et al. [52] proposed a Mathematical modeling based on Integer Linear Programming (ILP) technique to solve optimally the resource allocation problem. However, ILP technique is known for solving well known problem of scheduling in operating system. Author has proposed a model to use linear programming for selection of appropriate resource. Balaji Palanisamy et al. [53] proposed a cost aware allocation algorithm for MapReduce in cloud. This article presents a new MapReduce service model for cloud named Cura. Cura is cost efficient MapReduce model and cloud service to select the resource at run time for distributed problem with least cost and most efficient resource. Cura is also responsible for creation and selecting of cluster for dealing with workload. It also includes, VM-aware scheduling and online virtual machine reconfiguration, for better management and reconfiguration resources.

2.6. Behavior based algorithm

Behavior are the algorithm which are inspired from behavior of nature and behavior of animals and other living organism around us and their hierarchical evolution over the decades. These behaviors inspire us to make decisions based on previous behaviors or the environment for making better decision that may be prediction or forecasting. Some of these algorithms proposed in the field of cloud computing are discussed below.

Bei Wang & Jun Li [54] proposed a genetic algorithm based load balancing algorithm. In order to boost the search efficiency, the min-min and max-min algorithm are used for the population initialization. But these may get stuck in local minima and to find best solution genetic algorithm is proposed. Proposed algorithm proves to provide better solution but the scheduling delay to find best solution is much higher than min-min and max-min algorithms. Keke Gai [55] proposed a cost efficient data / storage allocation algorithm using genetic algorithm for video and metadata storage over cloud. This algorithm aims to provide heterogeneous memory storage space over cloud with least cost using genetic programming to select cheapest service provider. Output proves that the proposed algorithm proves to provide improved communication costs, data move operating costs and energy performance.

Lizhen Cui [56] proposed a genetic algorithm based replica management algorithm for cloud. Author has proposed a tripartite graph based model to formulate the data replica placement problem and propose a genetic algorithm based data replica placement strategy for scientific applications to reduce data transmissions. The proposal provides better performance than random selection policy in Hadoop Distributed File System. Jasraj Meena [57] proposed a cost efficient genetic algorithm to optimize the cost for work flow schedule rather than for single tasks. The proposed algorithm proves to execute the workflow with least cost. The algorithm is been tested over popular workflow like Montage, LIGO, CyberShake, and Epigenomics.

Anjuli Garg [58] proposed a honey bee life cycle based task scheduling strategy for cloud. Author has taken into care utilization and task size to schedule the task and select the server which can execute with least execution time. Anqi Xu [59] proposed an Particle Swarm Optimization for task scheduling for cloud infrastructure to improve the Quality of Service of system. The author has taken into consideration multi objective to improve Makespan and cost. The algorithm proves to perform better than ACO and min-min algorithm. Bohrer et al. [60] proposed a most known base scheduling algorithm ACO (ant colony optimization) they proposed ant colony optimization algorithm to load balance by distributing request in a cloud computing environment. This paper proposed LBACO with dynamic load balancing strategy to distribute load among the node. The problem with traditional ACO in cloud is that it's a schedule task to most frequent (high pheromone intensity) node, if what if node is bearing heavy load in such situation may create a problem of overhead. This paper proposed and LBACO algorithm to reduce such problem. In this algorithm decrease the time of computation and monitor load on each VM with tracking previous scheduling. Xiaobo et al. [61] proposed and Real-time VM provisioning model, which is based on energy models which follow a Min-Price RT-VM Provisioning to allocate VM. Suraj, S. Rin et al. [62] proposed a genetic algorithm for task allocation in cloud environment with least execution time and maximum resource utilization.

Many other proposal made [82-86] using ACO, genetic algorithm and other learning based algorithm are been studied. Jaradat [87] proposed a Big Bang-Big Crunch optimization algorithm to solve the problem of scheduling classed for a timetable. This algorithm has proved to perform better than existing GA based algorithm.

2.7. Trust Models

Numerous trust models have been proposed in cloud. MohdaIzua Mohd Saad proposed a novel data provenance trusted model to provide secured access to data provenance via a secured communication channel [63]. This model also proposes consolidation storage with logging for virtual storage at physical layer in cloud environment. As shown in figure 1.

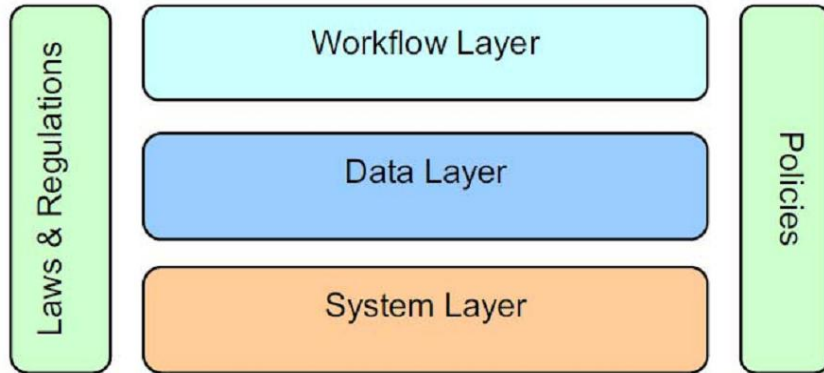


Figure 2.3. The trust cloud framework.

WenAn Tan proposed a trust service-oriented workflow scheduling algorithm [64]. The scheduling algorithm uses a trust metric that is combination of direct trust and recommendation trust. Proposed model also provided balancing policy to balance user requests, based on time, cost, and trust. Rizwana A.R. Shaikh proposes a trust based solution in terms of a trust model that can be used to calculate the security strength of a particular cloud service [365]. Proposed algorithm uses trust value for selecting a trusted cloud service.

[66] Xiaodong Sun introduces a trust management model based on fuzzy set theory and named TMFC including direct trust measurement and computing, connecting, and trust chain incorporating where the issue of recommended trust has been addressed to find the miss behavior of intermediate middle nodes. And this proposed model is designed for the cloud users to make decision on whether to use the services of some cloud computing providers by using trust value sets about providers and then finding trust relationships among them.

QiangGuo introduced a definition of trust in cloud systems and the properties of trust are analyzed [67]. Based on the properties of trust of a server, a trust evaluation model called ETEC is proposed. Proposed trust model includes a time based comprehensive evaluation

method for calculation of direct trust and a space evaluation method for calculating recommendation trust of server. For computing the trust in cloud, an algorithm based on the trust model is given. Experimental analysis shows that the proposed model can calculate the trust value of server effectively and reasonably in cloud computing environments. Xiaoqiong Yang also proposed A Statistical User-Behavior Trust Evaluation Algorithm Based on Cloud Model for statistic behaviors. Proposed algorithm used threshold for each type of behaviors and each user's performance and its membership status in cloud [68]. Then the membership degree and the behavior weight will be used to calculate the user's trust using a simple normalization function. Proposed algorithm uses the evaluated domain trust and recommendation trust, behavior trust for users' further dynamic authorization of access control and request load balancing. Junfeng Tian proposed a Trusted Control Model of Cloud Storage with access control (TCMCS) to handle all the interactions between a client and cloud storage to ensure the secure user access and data manipulation. The proposed trust model is responsible for managing different cloud storage and manages security and integrity of user data over the cloud. Since users only need to care about their own business logic and the development of application program is greatly simplified [69]. Proposed model can be specified as shown in figure 2.

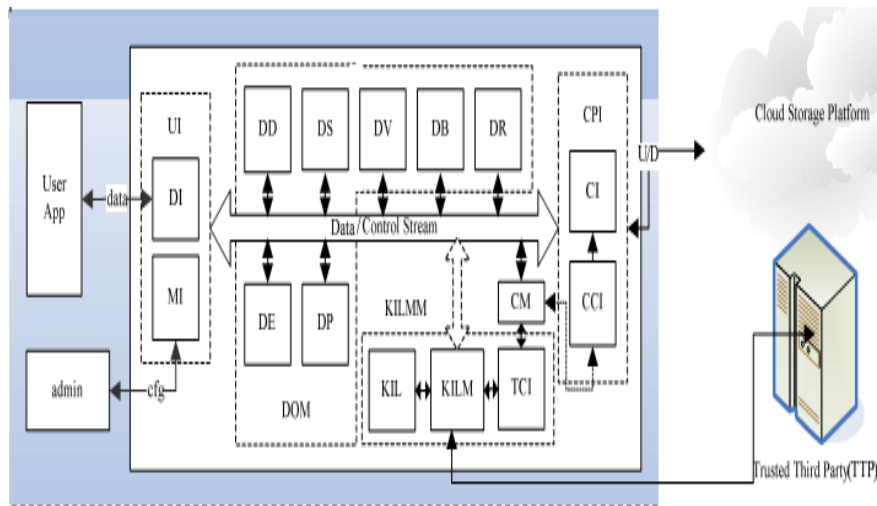


Figure 2.4 The logical structure diagram of TCMCS.

In [70] gupta has proposed a QoS Based Trust Management Model for Cloud IaaS that is suitable for trust value management for the cloud IaaS parameters. Proposed a scheduling algorithm based on trust value is done for better resources allocation and enhance the QoS provided to the users. In this paper, an approach for managing trust in Cloud IaaS is proposed.

Various other trust models are also been proposed [103 -106] to improve the system performance and reliability of reliable computing.

2.8. Conclusion

From above discussion, this can be seen that various resource allocation and load balancing algorithms fall short with problem of request scheduling using dynamic techniques to improve the performance of system. These algorithms do not consider load over the system or faults that may occur periodically over system, and also lack into consideration of previous performance history of the machines/servers.

In section 2.3, various load balancing algorithms are discoursed but these proposals consider only current load over the servers and do not search for global solution that can be based on performance of the server. These algorithms have considered only the current performance and not the physical capability of servers. Moreover, existing load balancing algorithms have assumed system as non faulty that leads to a large number of faults over the system.

In section 2.3 and 2.4 discourses about energy and cost efficient allocation and load balancing algorithms for cloud but they are not suitable for real-time systems because they do not take into consideration physical aspects of servers, deadline of requests, and considers cloud as non faulty. So proves to provide poor performance for request with tight deadline and if the system is faulty in nature i.e. the server may have high capability in terms of number of cores and RAM but may be faulty in nature, in that case existing algorithm goes under worst case performance. These algorithms also do not provide support for the reliability of the distributed systems.

Above discoursed behavior based algorithms in section 2.5 are inspired from nature which are used for resource allocation and load balancing but they only take into one parameters at a time like utilization, cost or power efficiency so cannot improve the performance of the system in all. Moreover, these algorithms do not consider the faulty nature of cloud, deadline, physical aspects of servers and previous performance of servers i.e. request failure count that had occurred or load over the serves, so cannot guarantee a high reliability and fault tolerant of the system. This may lead to large request failure counts due to overloading, deadline failure, and faults over the system which results in poor performance, and low reliability of the system.

This chapter identifies various techniques and issues and provide future directions to propose new methodologies for efficiency improvement, and fault tolerance in the cloud computing.

CHAPTER 3

FAULT AND DEADLINE AWARE LOAD BALANCING

With the rapid growth in technology, there is a huge proliferation of data requests in cyberspace. Distributed system/servers play a crucial role in the management of request in cloud which are distributed among the various geographical zones. Many of time the system gets over loaded due to few of servers with high number of request and some of servers being idle. This leads to degradation of performance of over loaded servers and failure of requests. On these over loaded servers average response time of server increases. So there is a requirement to design a load balancing algorithm to optimize resource utilization, response time and avoid overload on any single resource.

The management of data in cloud storage requires a special type of file system known as distributed file system (DFS), which had functionality of conventional file systems as well as provide degrees of transparency to the user, and the system such as access transparency, location transparency, failure transparency, heterogeneity, and replication transparency [71]. DFS provides the virtual abstraction to all clients that all the data located closest to him. Generally, DFS consists of master-slave architecture in which master server maintains the global directory and all metadata information of all the slave servers. Whereas, slave represents a storage server that stores the data connected to master server and other storage servers as well. This storage server handles the thousands of client requests concurrently, in DFS. The load distribution of requests on these storage servers is uneven and lead to performance degradation overall. Resources are not exploited adequately, because some

server gets too many requests and some remain idle. In a distributed storage system, load can be either in terms of requests handled by a server or storage capacity of that server or both.

In this section, we have proposed a set of approaches that balances the load of servers and effectively utilizes the server capabilities and resources. The main contribution of this work is to improve the average resource utilization of system and removing hot spots and cold spots in the system i.e. the unbalancing of requests over the system should be removed.

3.1. Approach 1: Fault and Load Aware Load Balancing in Cloud Storage

In this approach, we have proposed a Fault and Load based Load balancing algorithm (FLBLBA) that can balance a load of servers dynamically by considering its parallel processing capability, processing time and its request queuing capacity. Proposed algorithm aims to improve the performance cloud storage system by reducing request failure count, Average queue length, average utilization, total execution time.

The work is divided into various sections, where section 1 focusses on basic discription of problem. Section 2 & 3 describes the proposed approach and algorithm. In section 4 we have presented experimental results and comparative study of proposed algorithm.

3.1.1. Problem Statement

Distributed file systems provide a common virtual file system interface to all users as in DFS storage servers are distributed geographically and because of this load dis- tribution of client's requests to these servers become uneven. This problem can be illustrated clearly through Figure 3.1. Here, we have taken five storage servers S1, S2, S3, S4 and S5 with their respective service rate (S_r) present in the system. Service rate of a server signifies the number of requests processed by a server in a given time. Initially at time $t=0$, we assume that each server receives an approximately equal amount of requests as shown in Figure 3.1(a). We have taken total 8 requests to illustrate the scenario of our problem statement. In the second case as shown in Figure 1(b) after time $t=2$, each server process the client's requests as per its service rate and server S1 requests gets over much earlier than other servers and S1 becomes idle. Server S3 and S5 are fully loaded and takes their time to process all requests. From this scenario, we can say that distributed file system does not utilize each server

efficiently. In real- world situation, these requests are too large as compare to server service rate.

So in order to increase the system performance some requests which are in queue must be migrated to the idle servers or least loaded server and completes the request without failure. Our aim is to avoid queue like situations, utilizing the capability of each server efficiently and fulfill maximum request without failure.

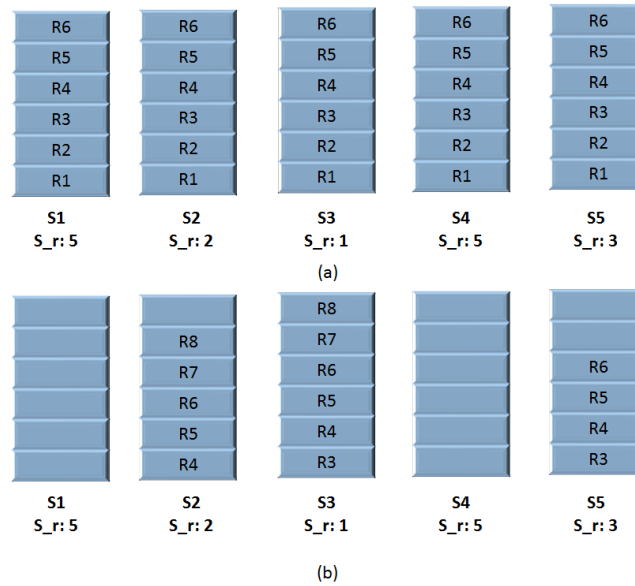


Figure 3.1 Problem statement for load balancing (a) at time $t=0$, servers receive equal amount of client requests. (b) at time $t=2$, scenario of servers after processing the receive requests.

3.1.2. Problem Approach

Here, we have proposed a Fault and Load based Load balancing algorithm (FLBLBA) that can balance the load of servers dynamically by considering its parallel processing capability, processing time and its request queuing capacity. Proposed approach takes four main parameters of a server 1) Server request queue size - buffer space to store the client requests to be handled by the server. 2) Server service rate (λ) - the number of CPUs available for processing the client request in a server. 3) Processing time (S_T) – time takes to process a request which differs from server to server. 4) Fault rate. Modern servers are equipped with many features like multiple CPUs, large storage, high I/O capability etc. We have chosen the multiple CPUs feature as a main parameter for load balancing of our proposed approach.

Following are the few assumptions that we have considered for our proposed approach:

- It is assumed that all the servers belong to same organization which can be geographically apart from each other. So each server maintains the replica of every server data.
- It is also assumed that all servers are strongly connected with each other through high bandwidth medium.
- Each server maintains global view which contains the information of its neighbors through master server.

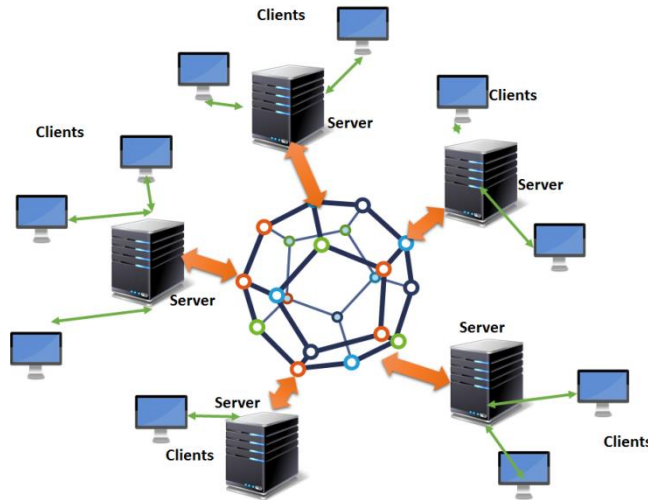


Figure 3.2 Organization of distribute storage servers.

Figure 3.2 shows the general scenario of distributed storage servers. In figure 2, there could be N connected servers where $N \in \{1, 2, 3, \dots, n-1\}$, in the system. Each server has following properties such as request queue, number of CPUs, storage capacity. Clients send their requests to the respective server. Many times the incoming request rate (ρ) increases exponentially to a particular server. This is because of the series of client's requests to that data that is stored within the server. In case, when a server gets too many requests than server buffers them in their request queue and the size of request queue gets increases dynamically only upto its predefined threshold limit. Once, the request queue breaches the threshold limit than server is considered as overloaded server and triggers the load balancer. Load balancer classifies the least loaded server on the basis of their request queue and processing capacity. As soon as the least loaded server gets classified than overloaded server migrate its load to that server and balances the load. Various notations are used in the proposed approach and represented as follows:

ρ - Current queue size of server.

- λ_i - Service rate that is number of request processed simultaneously on a server.
 S_T -Service time is the time taken by server to process the request
 $Q_L_{Current}$ -Current queue length of server
 $Q_L_{Threshold}$ - Threshold limit of server request queue.
 ΔL_i - additional load on server i.
 W_i - Waiting time for a request at server i.
 FT_i - Count of request failed.
 FR_i -Fault rate that is the numbe of request failed due to system failure over time t.
 F_j - Fitness value of neighbors of server i. ($j \in \{1,2,3 \dots n-1\}$)

We have considered the real world scenario where the server request queue size and service rate changes with respect to time t dynamically and represented as $\delta\rho$ and $\delta\lambda$ respectively

$$\delta\rho = \frac{\rho}{\delta t} \text{ and } \delta\lambda = \frac{\lambda}{\delta t} \quad (3.1)$$

Fault rate of a server can be given as:

$$FR_i = FT_i / \text{time} \quad (3.2)$$

Storage server is said to be overloaded if:

$$\delta\rho > Q_L_{threshold} \quad (3.3)$$

When server i where $i \in \{1,2,3 \dots n-1\}$ is overloaded then it calculates the amount of extra load ΔL_i on that server which can be calculated as follow:

$$\Delta L_i = Q_L_{current} - Q_L_{threshold} \quad (3.4)$$

The condition when a load balancer module gets triggered on the overloaded server i is given below:

$$T(i) = \begin{cases} 1, & \Delta L > 0 \\ 0, & otherwise \end{cases} \quad (3.5)$$

$T(i)$ = Triggering function.

Once, the load balancer module is triggered, server i find the least loaded or idle server that can accommodate its load and adequately process the service requests without failure. For this load balancer calculates the fitness value F_j that can be calculated using the following fitness function:

$$\Delta M_j = Q_{L_{threshold}} - Q_{L_{current}} \quad (3.6)$$

Here, ΔM_j is free request queue of server j . If ΔM_j is negative, then server j request queue is overloaded otherwise it is least loaded.

$$F_j = \alpha_1 \cdot \Delta M_j + \alpha_2 \cdot \lambda_j + \alpha_3 \cdot \left(\frac{1}{FR_j}\right) + \alpha_4 \cdot \left(\frac{1}{W_j}\right) \quad (3.7)$$

Here, α_1 and α_2 are constants and may vary according to scenario such that

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \quad (3.8)$$

For our proposed scenario, we have considered the value of α_1 and α_2 is 0.5 it is because both the parameters play the equal role in load balancing. In this way, load balancer calculates the fitness value for each neighbors of server i . and select that server which has maximum fitness F_j value, i.e. fault rate of server less than migrating server and migrate the ΔM_j amount of load to server j . Selecting the server with maximum fitness value in turn decreases the failure probability of request and completes the request as soon as possible with least waiting time.

3.1.3. Proposed Model

Proposed algorithms have been designed to balance the client requests over the servers and distribute the load over the system uniformly. Here, load balancer as shown in figure 3.3.(a) regularly exams for the request queue dimension of server and tries to restrict the problem of overloading of any server with the aid of migrating the extra request to other idle or least loaded and least faulty neighboring server in cloud. Proposed load balancing algorithm is divided into two stages. In first stage list of idle servers is created, and in second stage the server with highest fitness value and which can fulfill the request with least failure probability.

Stage I

Algorithms checks and calculate the fitness value for the neighbor server to store them in a list shown in figure 3.3(b). Load balancer utilizes this list to select the server that has highest fitness value. Load balancer calculates the waiting time over each server from above list which can be given as:

$$W_k = \frac{Q_{L_{current_k}}}{\lambda_k} \times S_{T_k} \quad (3.9)$$

Equation 1 shows the W_k waiting time of i^{th} request at server 'k'.

Stage II

In second stage load balancer then finds the server with least waiting time, least fault rate and highest service rate i.e. highest fitness value from the list.

A proposed algorithm also tries to improve the server response time by selecting the server having least CPU utilization. In this way, proposed algorithm utilizes the idle or underutilized server to increase the overall performance of the system and reduce requests failure over the system by reducing the probability of request failure.

Algorithm : Load balancing

1. **FLBLBA** (Server s , $Q_{L_{current}}$, λ_k , S_{T_k} , FR_i)
Input: Server s , Queue length $Q_{L_{current}}$, service rate λ_k , service time S_{T_k} , fault rate FR_i
 2. $s \leftarrow$ server
 3. $Q_{L_{current}} \leftarrow$ current queue size
 4. $\lambda_k \leftarrow$ Service rate of server k
 5. $S_{T_k} \leftarrow$ Service time of server k
 6. $FR_i \leftarrow$ fault rate of server k
 7. Compute W_k
 8. **if** ($\delta\rho > Q_{L_{Threshold}}$) **then**
 9. check server queue status.
 10. Add request to queue;
 11. Process_request();
 12. **else**
 13. server is overloaded
 14. $S \leftarrow$ Find_server(server_neighbour_list L)
 15. find under loaded server.
 16. $S \leftarrow$ migrate request
 17. Goto step 7
- Output:** Load balances the request.
-

Figure 3.3 (a) Proposed load balancing algorithm.

Algorithm : To find suitable server

```

1. Find_server(server_neighbour_list L )
   Input: server_neighbour_list L
2. for k=1 to L.size()
3.   S1←L.get()
4.    $F_k = \alpha_1 \Delta M_j + \alpha_2 \lambda + \alpha_3 (1/FR_j) + \alpha_4 (1/W_j)$ 
5.   temp_list t←  $F_k$ 
6. end for
7. L2 = Sort(t)
8. S2 ←min (L2)
9. return S2
Output: The server with minimum fitness
value.

```

Figure 3.3 (b) Find a neighbor server algorithm.

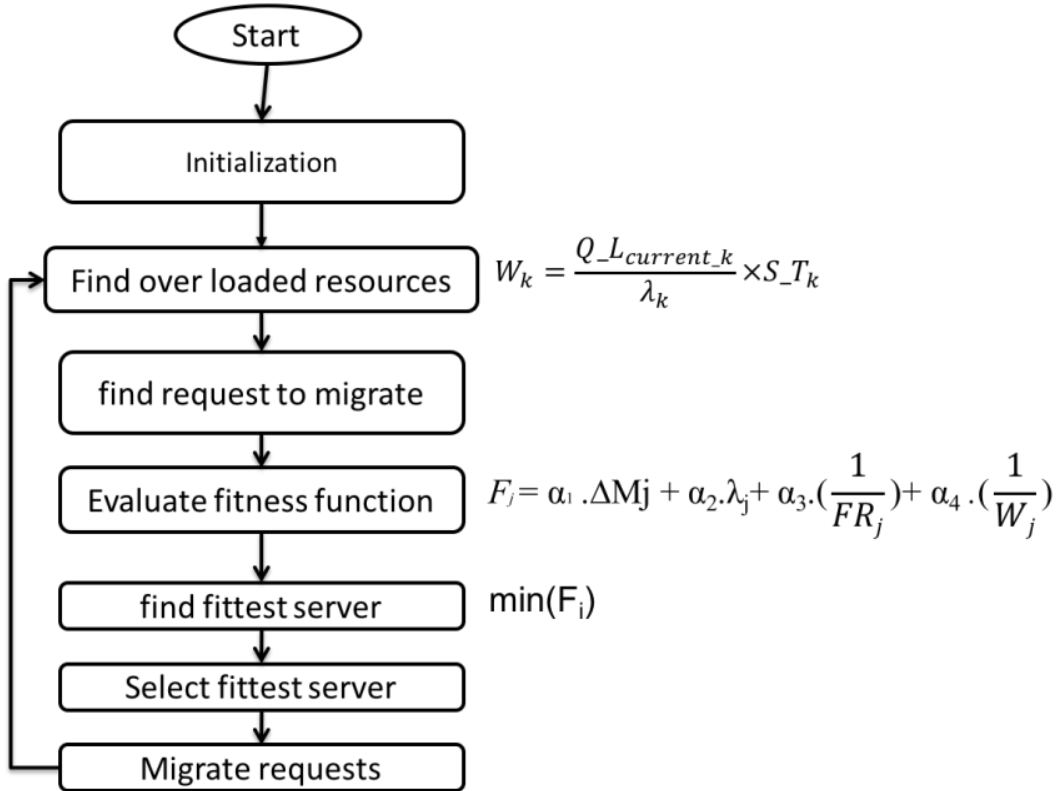


Figure 3.4 Proposed algorithms flow diagram

Figure 3.4 shows the flow of the algorithm with various phases of algorithm and interaction among them to find the fittest server for each request.

3.1.4. Experiment and Result

Performance analysis of proposed FLBLBA algorithm is finished using CloudSim [11] simulator where we now have thousands of requests to be completed by 12 storage servers. All of the servers work concurrently with constant quantity of CPU cores to process the client request rapidly. Each server has a request queue to buffer the incoming client requests, storage ability to store the data and fulfill the client requests.

For the given problem statement in section 3 where the load is unbalanced, it is assumed that half of storage servers get client requests and others remain idle. Our motive is to equally distribute the received client requests among the servers to avoid the scenario of overloading. In the simulation scenario numbers of storage servers are kept fixed with varying number of requests handling. We have also compared the obtained results with the least load balancing algorithm. Following table depicts the configuration parameter for our simulation environment.

Table 3.1: Experimental parameters used for simulation environment

No. of client requests	No. of Servers	No. of CPU cores available per server	Storage capacity of servers (GB)	Server queue length
800	12	7	500	20
1000	12	7	500	20
1200	12	7	500	25
1800	12	7	500	25
2400	12	7	500	25

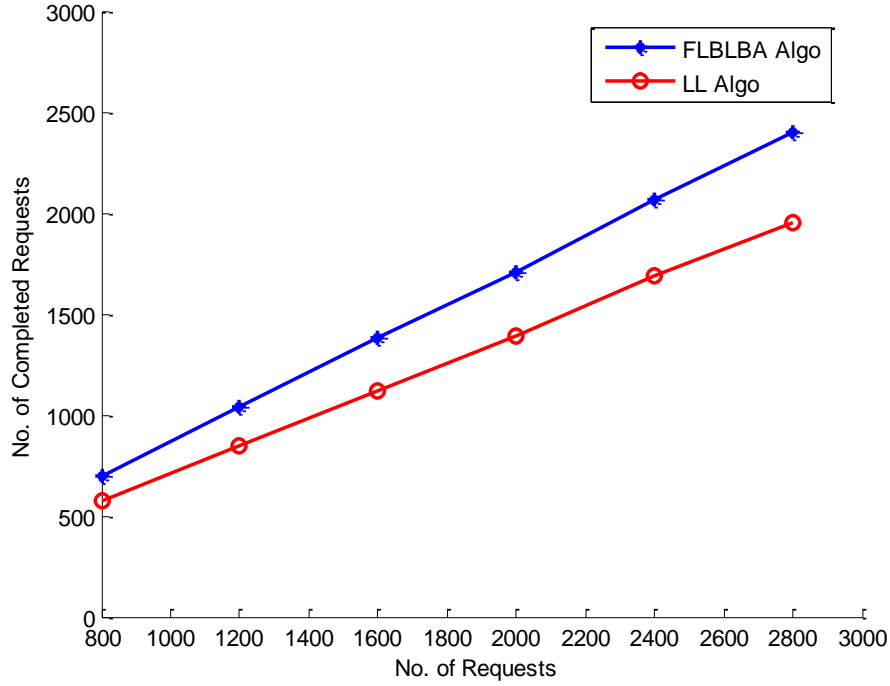


Figure 3.5 Number of request completed

Figure 3.5 shows the number of processed client requests by server in a given time. Here, Figure 3.5 represents the graph between numbers of sent requests vs. numbers of completed request whereas Figure 3.6 represents the graph between no. of sent requests vs. no. of failed requests for the proposed and least load algorithms. In least loaded algorithm when any server get overloaded then load balancer selects the server of which request queue is least loaded without considering the CPU parameter. For the proposed algorithm we have considered the CPU parameter and from obtained results as shown in Figure 3.5, Figure 3.6 and Figure 3.7 that the proposed algorithm perform much better over the least load algorithm. Figure 3.8 shows that the proposed FLBLBA algorithm improves the average utilization of the system drastically over increasing requests due to improvement in total request completed. In all set of client requests, proposed algorithm process more number of client's request with better overall response time as shown in Figure 3.7.

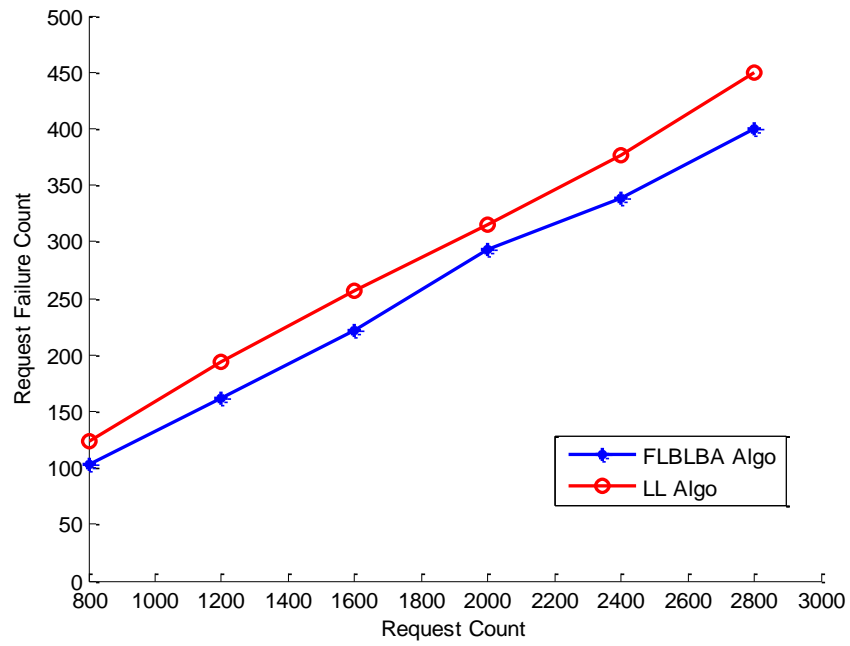


Figure 3.6 Number of sent requests vs. no. of failed requests.

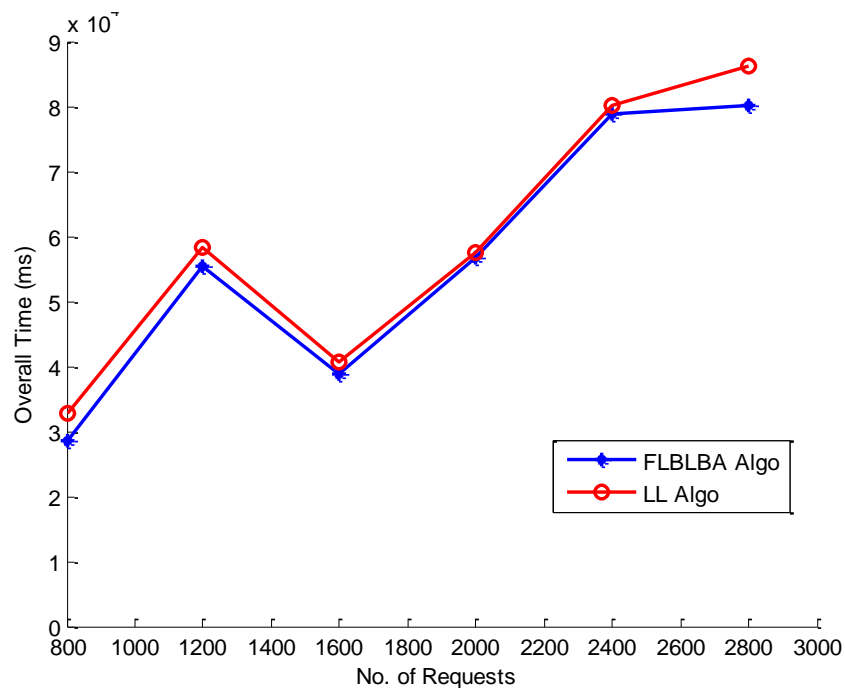


Figure 3.7 Overall response time.

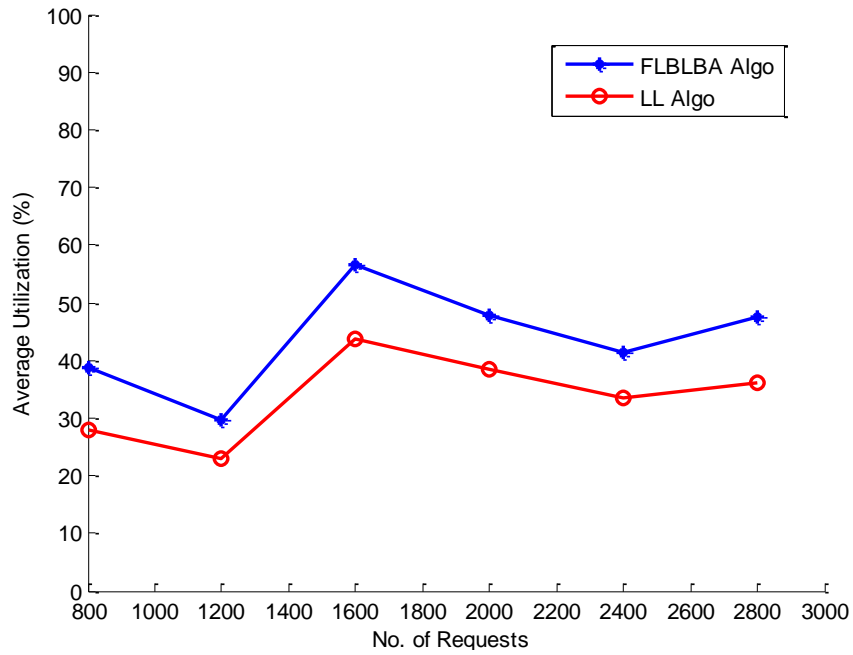


Figure 3.8 Average utilization of system

3.2. Approach 2: Deadline Aware Load Balancing of Distributed Servers in Distributed

In this approach, we have proposed a Deadline based Load balancing algorithm (DDLBA) that can balance the load of servers dynamically by considering its parallel processing capability, processing time and its request queuing capacity. Proposed algorithm aim to overcome the failures due to long waiting time rather than network or system failure. Algorithm takes care of request which have small deadline and need to be assigned to a server with high service rate that can fulfill the request within the deadline. Proposed algorithm improves the performance of cloud storage system by reducing request failure count, Average queue length, average utilization, total execution time.

The work is divided into various sections, where section 1 & 2 describes the proposed approach and algorithm. In section 3 we have presented experimental results and comparative study of proposed algorithm.

3.2.1. Proposed Approach

Here, we have proposed a Deadline based Load balancing algorithm (DDLBA) that can balance the load of servers dynamically by considering its parallel processing capability, processing time and its request queuing capacity. Proposed approach takes three main

parameters of a server 1) Server request queue size - buffer space to store the client requests to be handled by the server. 2) Server service rate (λ) - the number of CPUs available for processing the client request in a server. 3) Processing time (S_T) – time takes to process a request which differs from server to server. Modern servers are equipped with many features like multiple CPUs, large storage, high I/O capability etc. We have chosen the multiple CPUs feature as a main parameter for load balancing of our proposed approach.

Following are the few assumptions that we have considered for our proposed approach:

- It is assumed that all the servers belong to same organization which can be geographically apart from each other. So each server maintains the replica of every server data.
- It is also assumed that all servers are strongly connected with each other through high bandwidth medium.
- Each server maintains global view which contains the information of its neighbors through master server.

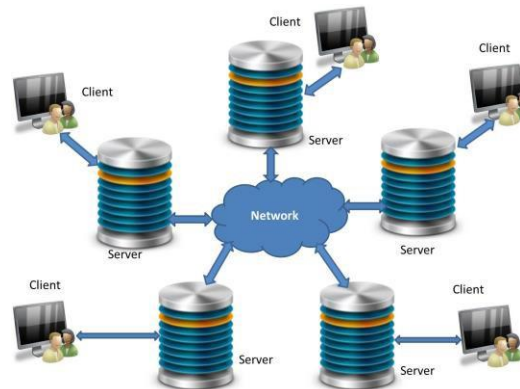


Figure 3.9 Organization of distribute data servers.

Figure 3.9 shows the general scenario of distributed storage servers. In figure 3.9, there could be N connected servers where $N \in \{1, 2, 3, \dots, n-1\}$, in the system. Each server has following properties such as request queue, number of CPUs, storage capacity. Clients send their requests to the respective server. Many times the incoming request rate (ρ) increases exponentially to a particular server. This is because of the series of client's requests to that data that is stored within the server. In case, when a server gets too many requests than server buffers them in their request queue and the size of request queue gets increases dynamically only upto its predefined threshold limit. Once, the request queue breaches the threshold limit

than server is considered as overloaded server and triggers the load balancer. Load balancer classifies the least loaded server on the basis of their request queue and processing capacity. As soon as the least loaded server gets classified than overloaded server migrate its load to that server and balances the load. Various notations are used in the proposed approach and represented as follows:

ρ	- Current queue size of server.
λ	- Service rate that is number of request processed simultaneously server.
S_T	-Service time is the time taken by server to process the request
Q_L threshold	- Threshold limit of server request queue.
Q_L Current	- current capacity of server request queue at time t.
ΔL_i	- additional load on server i.
DL_i	- Deadline time of request i
F_j	- Fitness value of neighbors of server i.($j \in \{1,2,3 \dots n-1\}$)

We have considered the real world scenario where the server request queue size and service rate changes with respect to time t dynamically and represented as respectively

$$\delta\rho = \frac{\rho}{\delta t} \text{ and } \delta\lambda = \frac{\lambda}{\delta t} \quad (3.10)$$

$$\delta\rho > Q_L_{threshold} \quad (3.11)$$

When server i where $i \in \{1,2,3 \dots n\}$, is overloaded then it calculates the amount of extra load ΔL_i on that server which can be calculated as follow:

$$\Delta L_i = Q_L_{current} - Q_L_{threshold} \quad (3.12)$$

The condition when a load balancer module gets triggered on the overloaded server i is given below:

$$T(i) = \begin{cases} 1, & \Delta L > 0 \\ 0, & otherwise \end{cases} \quad (3.13)$$

Once, the load balancer module is triggered, server i find the least loaded or idle server that can accommodate its load and adequately process the service requests without deadline failure. For this load balancer calculates the fitness value F_j that can be calculated using the following fitness function:

$$\Delta M_j = Q_{L_{threshold}} - Q_{L_{current}} \quad (3.14)$$

Here, ΔM_j is free request queue of server j . If ΔM_j is negative, then server j request queue is overloaded otherwise it is least loaded.

$$F_j = \alpha_1 \Delta M_j + \alpha_2 \lambda \quad (3.15)$$

Here, α_1 and α_2 are constants and may vary according to scenario such that

$$\alpha_1 + \alpha_2 = 1 \quad (3.16)$$

For our proposed scenario, we have considered the value of α_1 and α_2 is 0.5 it is because both the parameters play the equal role in load balancing. In this way, load balancer calculates the fitness value for each neighbors of server i and select that server which has maximum fitness F_j value and waiting time of request less than deadline of request and migrate the ΔM_j amount of load to server j .

3.2.2. Proposed Model

Proposed algorithms have been designed to balance the client requests over the servers and distribute the load over the system uniformly. Here, load balancer as shown in Figure 3.10(a) is responsible for consistently assessment of the request queue dimension of servers and tries to prevent the main issue of overloading of any server with the aid of migrating the request to other idle or least loaded neighboring servers which can also complete the request in deadline without failure in the system. Proposed load balancing algorithm is divided into two stages. In first stage list of idle servers is created, and in second stage the server with highest fitness value and which can fulfil the deadline is selected.

Stage I

Algorithms checks and calculate the fitness value for the neighbour server to store them in a list shown in figure 3.10 (b). Load balancer utilizes this list to select the server that has highest fitness value. Fitness value is evaluated based on deadline and the empty

Stage II

In second stage load balancer calculates the waiting time over each server from above list which can be given as:

$$W_k = \frac{Q_{L_{current,k}}}{\lambda_k} \times S_{T_k} \quad (3.17)$$

$$W_k > DL_i \quad (3.18)$$

Equation 3.18 shows the waiting time of k^{th} request. DL_i deadline of a request. Load balancer then finds the server with waiting time less than request deadline and highest fitness value from the list. Proposed algorithms also try to reduce the server response time by selecting the server with least CPU utilization. In this way, we utilize the server to increase the overall performance of the system and reduce request deadline failures over the system.

Algorithm : Load balancing

```

1. DDLBA (Server s,  $Q_{L_{current}}$ ,  $\lambda_k$ ,  $S_{T_k}$ ,  $DL_i$ )
Input: Server s, Queue length  $Q_{L_{current}}$ , service rate  $\lambda_k$ ,
service time  $S_{T_k}$ , deadline  $DL_i$ 
2.  $s \leftarrow$  server
3.  $Q_{L_{current}} \leftarrow$  current queue size
4.  $\lambda_k \leftarrow$  Service rate of server k
5.  $S_{T_k} \leftarrow$  Service time of server k
6.  $DL_i \leftarrow$  Deadline time of request i
7. Compute  $W_k$ 
8. if ( $\delta\rho > Q_{L_{Threshold}}$ ) then
9.   check server queue status.
10.  Add request to queue;
11.  Process_request();
12. else
13.  server is overloaded
14.   $S \leftarrow$  Find_server( server_neighbour_list L )
15.  find under loaded server.
16.   $S \leftarrow$  migrate request
17. Goto step 7
Output: Load balances the request.

```

Figure 3.10 (a) Load balancing algorithm.

Algorithm : To find suitable server

```

1. Find_server(server_neighbour_list L )
   Input: server_neighbour_list L
2.  for k=1 to L.size()
3.    S1←L.get()
4.     $F_k = \alpha_1 \Delta M_j + \alpha_2 \lambda$ 
5.    temp_list t←  $F_k$ 
6.    L2 = Sort(t)
7.  end for
8.  for j=1 to L2.size()
9.    if(L2.  $W_k < DL_i$ ) then
10.     S2 ←L2
11.  end for
12. select server which fits the deadline and
    have maximum fitness value.
13. return S2
Output: The server with minimum fitness value.

```

Figure 3.10 (b) Find neighboring server algorithm.

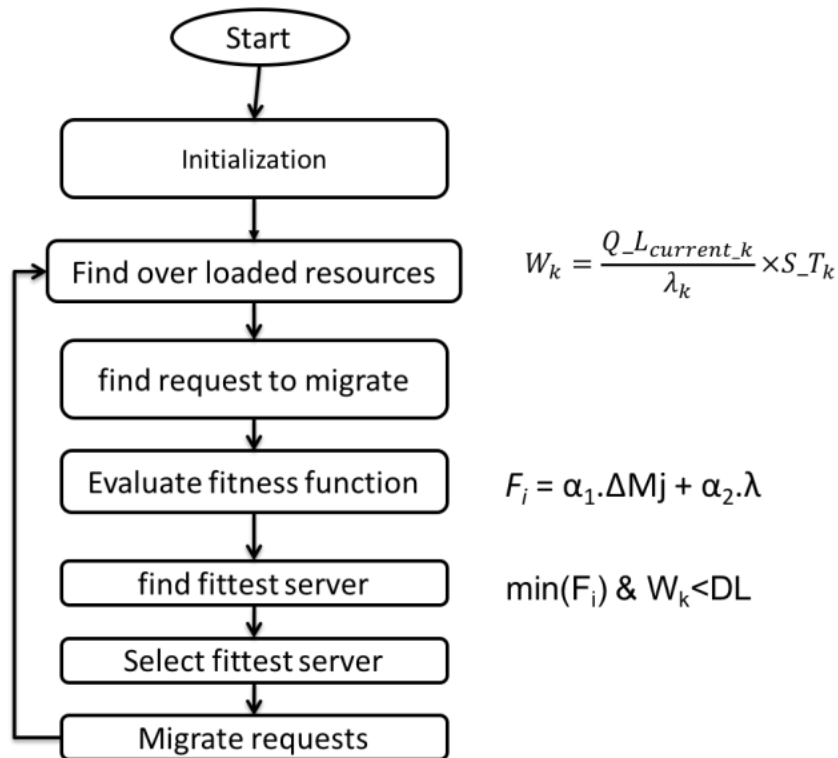


Figure 3.11.Proposed algorithms flow diagram

Figure 3.11 shows the flow of the algorithm with various phases of algorithm and interaction among them to find the fittest server for each request.

3.2.3. Experiment and Result

Efficiency evaluation of proposed DDLBA algorithm is completed utilizing simulations the place we've created hundreds of thousands of virtual machine requests to be served via 12 storage servers. All servers are deployed in distributed environment with unique count of CPU cores complete the requests. Each server has a fixed request queue size to buffer the incoming client requests and storage capacity. For the given main issue assertion in part 3 where the load is unbalanced, it is assumed that half of storage servers get client requests and others stay idle. Our aim is to equally distribute the received requests among the many servers to restrict the scenario of overloading. Within the simulation situation numbers of storage servers are kept fixed with various quantities of requests handling. We've got additionally compared the bought results with the least load balancing algorithm. Following table depicts the configuration parameter for our simulation atmosphere.

Table 3.2: Experimental parameters used for simulation environment

No. of client requests	No. of Servers	No. of CPU cores available per server	Storage capacity of servers (GB)	Server queue length
800	12	7	500	15
1000	12	8	500	15
1200	12	9	500	20
1800	12	10	500	20
2400	12	11	500	20

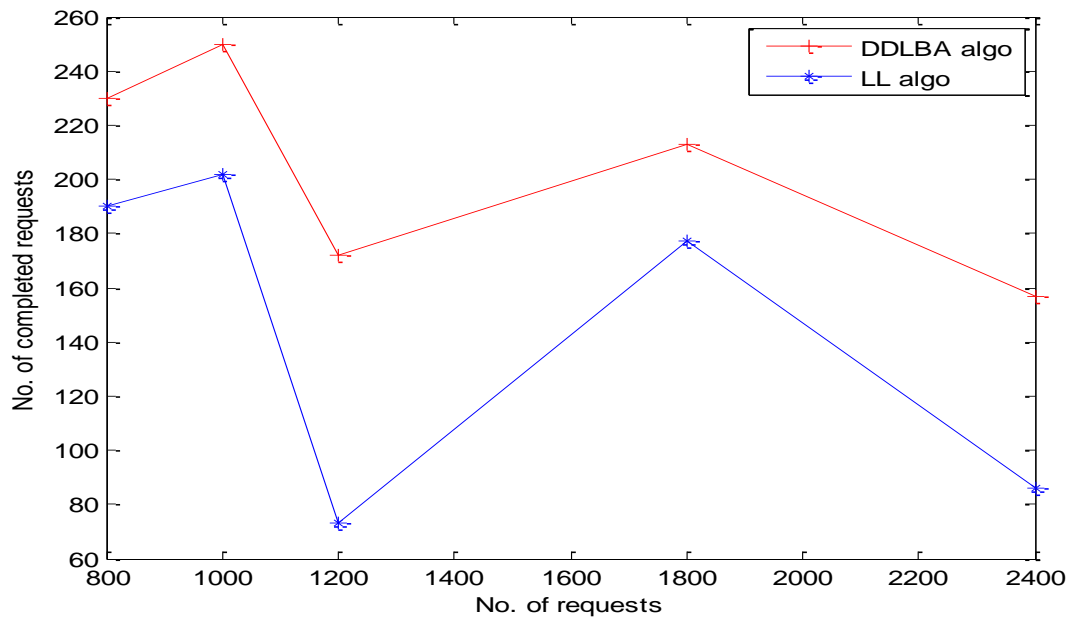


Figure 3.12 Comparison between no. of sent requests vs. no. of completed requests.

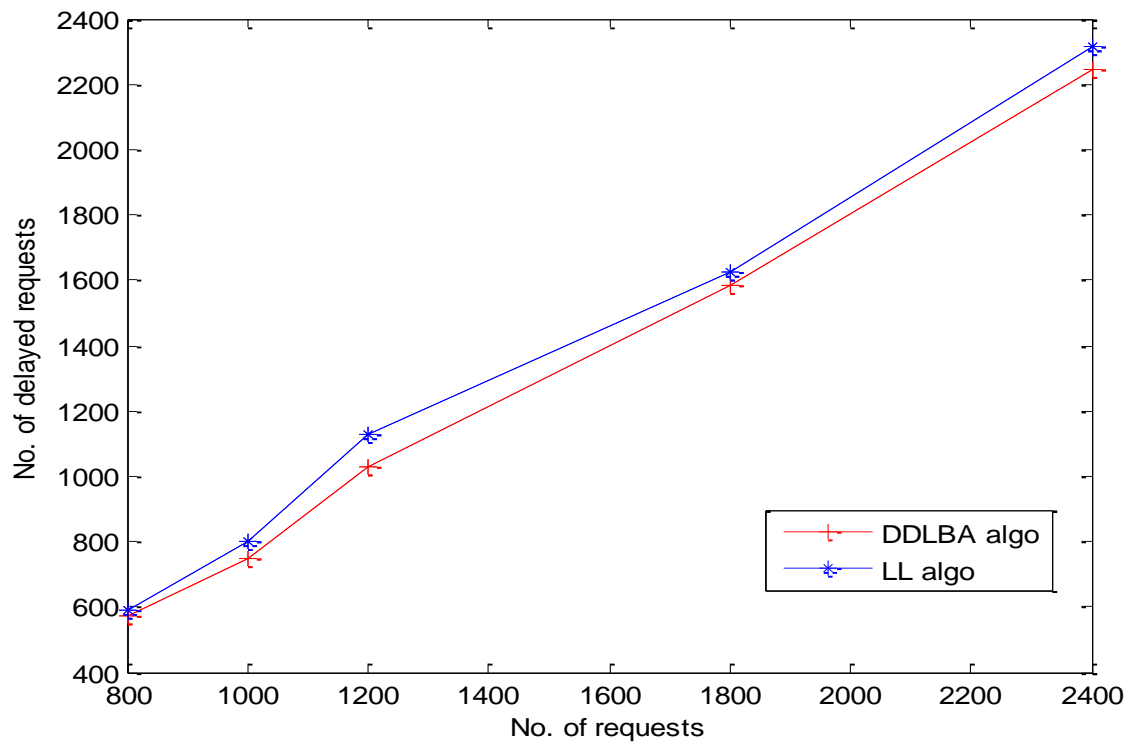


Figure 3.13 Comparison between no. of sent requests vs. no. of postponed requests.

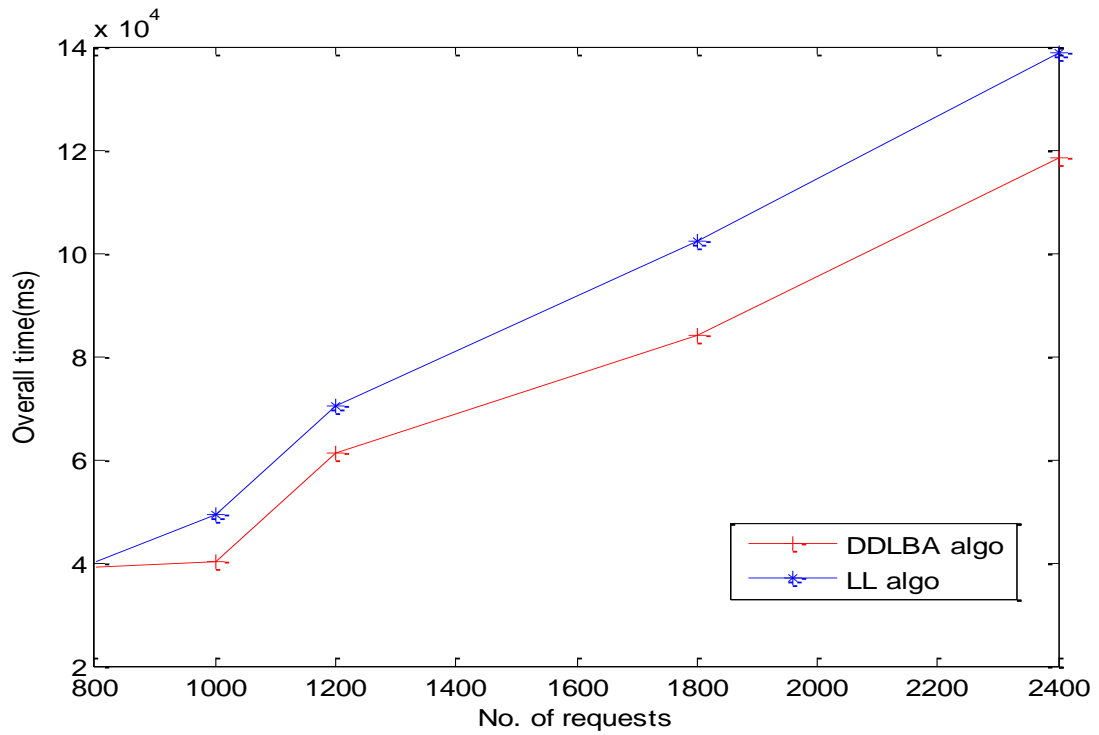


Figure 3.14 Overall response time.

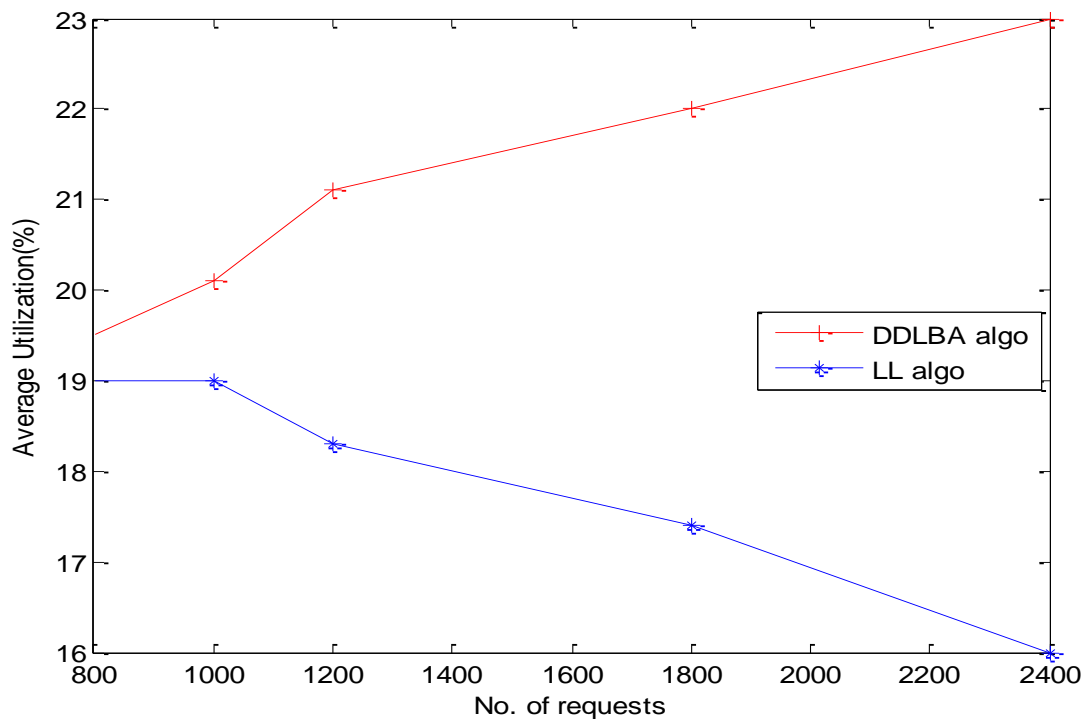


Figure 3.15 Average utilization of system

Figure 3.12 shows the number of processed client requests by server in a given time. Here, figure 3.12 represents the graph between numbers of sent requests vs. numbers of completed

requests. Whereas figure 3.13 represents the graph between numbers of sent requests vs. numbers of postponed requests for the proposed and least load algorithms. In least loaded algorithm when any server get overloaded then load balancer selects the server of which request queue is least loaded without considering the CPU parameter. For the proposed algorithm we have considered the CPU parameter and from obtained results as shown in figure 3.12 & 3.14 that the proposed algorithm perform much better over the least load algorithm. Figure 3.15 shows that the proposed DDLBA algorithm improves the average utilization of the system drastically over increasing requests due to improvement in total request completed. In all set of client requests, proposed algorithm process more number of client's request with better overall response time as shown in figure 3.14.

3.3. Approach 3: Load Balancing Algorithm for Hybrid Cloud IaaS

Data confidentiality and trust formation are the major security concerns in the cloud. Therefore, there is a firm need to establish an Inherent trust on cloud service provider in order to assure the cloud behavior, data protection and make cloud technology globally acceptable and reliable for users. Data protection deals with protecting the individual or organization's private data which is shared over the cloud. It is possible only when the security and trustworthiness of both the service provider and user is ensured. Therefore there is a need to establish trust between both and hence we need to develop a trust management model.

Trust models are been used in all form of distributed environments ranging from MANETS (Mobile ad hoc network), Sensor network and Grid computing to validate the reliability of nodes over distributed network. In grid computing various trust models are been proposed to insure trust in term of security and reliability of the server or the node. Trust models are to resolve the problem of reliability in any heterogeneous environment, which contributed of nodes having different configuration spread over a network. There are many models being proposed in a cloud computing environment. Mostly trust models are proposed for cloud, SaaS (Software as a service) to overcome security issues. Some of the trust models proposed are being discussed in section II. The models proposed do not fit the cloud IaaS (Infrastructure as a Service) environment because they do not take into consideration the performance of datacenter for trust and also most of the trust models proposed uses direct trust based on the static parameters but the trust vale may change as the performance of datacenter changes.

So in this approach, we have proposed a hybrid trust management model to overcome this problem, by taking into consideration datacenter characteristics which vary from datacenter to datacenter. In proposed trust model we have used direct trust and recommended trust value to evaluate the variable trust value over the period of time. These trust value are been used by the scheduling algorithm proposed to improve the scheduling of the resources for hybrid cloud.

3.3.1. Proposed Approach

The Trust Model we are proposing here will work for Private, Public and Hybrid Cloud. It will take into consideration both the Direct and Indirect Trust or recommended trust. For the Calculation of Trust Value we will take Memory (RAM), MIPS (Million Instruction per Cycle), Frequency (Frequency of data center) and Fault Rate. We will initially calculate only Direct Trust and as the time will evolve we will also consider the indirect trust. We will initiate the parameter MIPS and Fault with zero and Memory and MIPS will be according to the datacenter. With these we will calculate the Trust Value and initial load balancing will be done according to that. As the Time will evolve, we will calculate the Indirect or Recommended Trust also using Fault Rate which we initially considered zero and Response Time. Now, since the request from both public and private cloud will start arriving, we assume that some of them will not be accomplished due to technical faults and hence the parameter Fault Rate will have some value other than zero.

After Calculating Values for both we will rate the datacenters according to the trust value calculated, This Trust value will be combined of Direct and Indirect Trust. And hence allocate the private request to a server with high trust value since the request has lower chances of going down and allocate the public request to a server with low trust value since the request has higher chances of going down. We will keep this dynamic process going in order to ensure a Trust Based Load Balancing.

Steps for load balancing algorithm.

Pseudo Code

- Start by Calculating individual trust values of each data center.
- Consider only direct trust initially and as the time evolve indirect trust will also come in consideration.
- Take these factors into consideration for calculation of trust value for Direct Trust:

- RAM (Memory)
 - MIPS
 - Frequency of data center
 - Fault Rate (initially 0)
- Now, according the size of request start dividing them into public or private.
 - Send the private cloud requests to a data center with high trust value since they are less likely to fail.
 - Send the public cloud requests to data center with low trust value as they are more likely to fail.
 - Meanwhile calculate the Indirect Trust, which is based on:
 - Fault Rate (number of task failed/total tasks)
 - Response Time (Time taken to accomplish a standard request.)
 - Update the trust value with the values from Indirect Trust and now allocate the load accordingly.

We will initially calculate only Direct Trust and as the time will evolve we will also consider the indirect trust. We will initiate the parameter MIPS and Fault with zero and Memory and MIPS will be according to the datacenter. With these we will calculate the Trust Value and initial load balancing will be done according to that. As the Time will evolve we will calculate the Indirect or Recommended Trust also using Fault Rate which we initially considered zero and Response Time. Now since the request from both public and private cloud will start arriving we assume that some of them will not be accomplished due to technical faults and hence the parameter Fault Rate will have some value other than zero.

RAM: flash memory of the server.

MIPS: Millions of instructions per second of server.

Initial_Trust : Trust value of the server based on the executorial power i.e. RAM and MIPS.

Fault_Rate : Number of faults in an server over a period of time 't'.

Updated_Trust : Updated Trust value based on initial trust and fault occurred over a period of time.

α_1, α_2 & α_3 : Constants

$$\text{Initial_Trust} = \alpha_1 * \text{RAM} + \alpha_2 * \text{MIPS}; \quad (3.19)$$

Where

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \quad (3.20)$$

$$Updated\ Trust = InitialTrust + \alpha_3 * \left(\frac{1}{FaultRate}\right) \quad (3.21)$$

Algorithm : Load Balancing Algorithm

1. Load Balancing Algorithm(Req_list r)
 Input: Requests list r
 2. Initialize servers
 3. Calculate individual trust values of each data center
 4. CheckDirect_TrustValue ()
 5. CheckIndirect_Value ()
 6. dividing them into public or private according the size of request.
 7. **if** (Data Center Value > Threshold) **then**
 8. Send the private cloud requests.
 9. **else if** (Data Center Value < Threshold)
 10. Send the public cloud requests.
 11. **else**
 12. Keep Searching
- Output:** All request been scheduled.
-

Figure 3.16 Proposed trust based algorithm

Figure 3.16 shows the pseudo code for proposed algorithm with various steps to find the fittest server for each request.

3.3.2. Experiment and Results

Proposed trust model is simulated using Cloudsim API [24]. Cloudsim basically support cost estimation, and Random Load balancing of resource and has no support to study behaviour of datacenter and trust model. So to study the datacenter behaviour trust parameter is introduces as datacenter property, which depends on parameter defined in Cloudsim. Based on this attribute we have computed the result to show the real problem. For this we have considered two classes of budget range, with 3 datacenters and 300 user requests.

Table 3.3: Servers Parameters

Server Name	Fault rate	Hard disk (MB)	RAM (MB)	MIPS	Cores
Server1	0.143	1000000	20048	10000	4
Server2	0.125	1000000	20048	10000	6
Server3	0.5	1000000	20048	10000	4

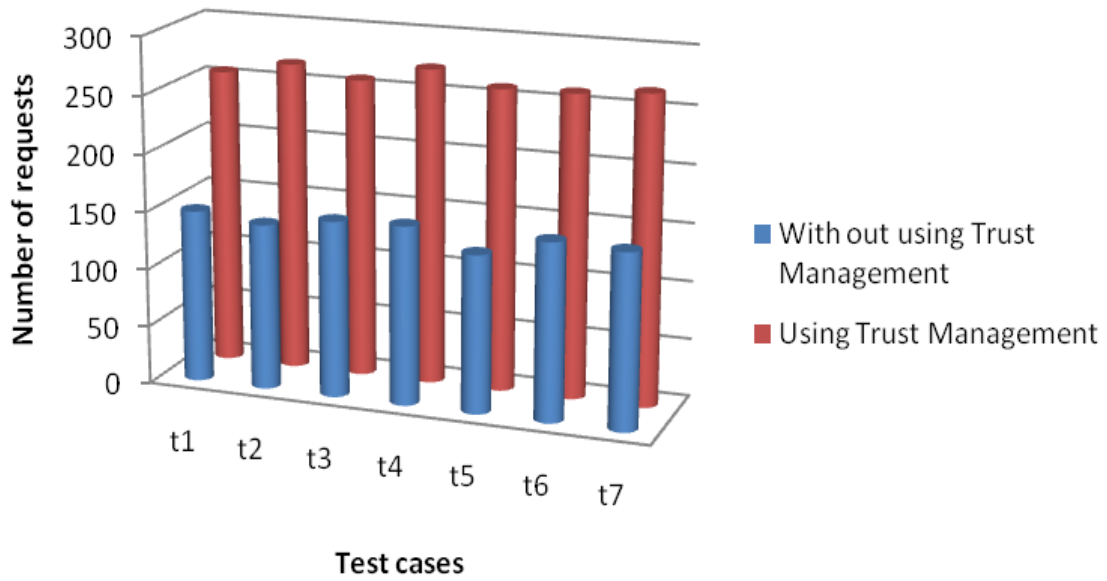


Figure 3.17 Total number of request completed in faulty environment

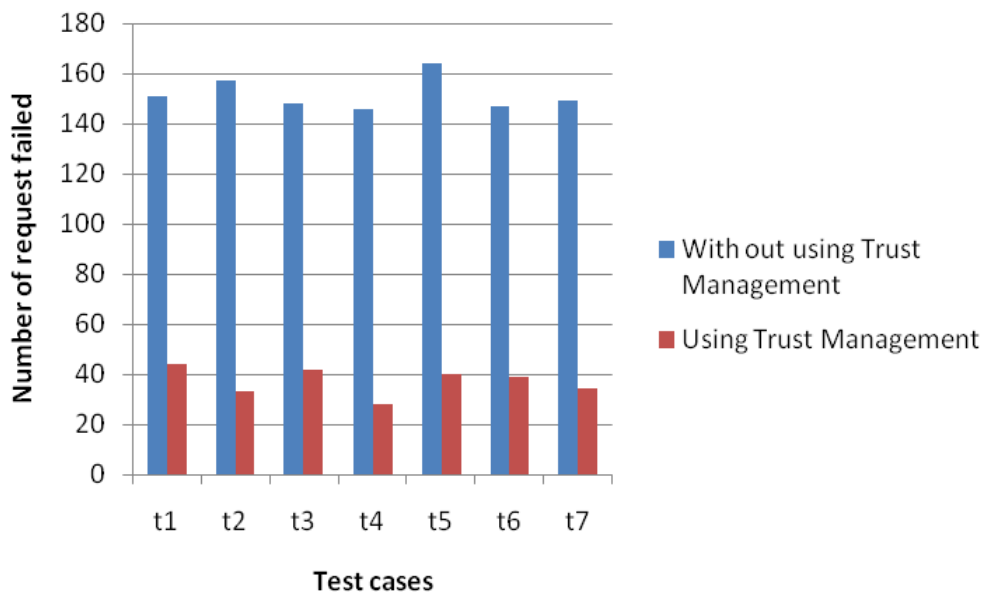


Figure 3.18 Total number of request failed.

Figure 3.17 and 3.18 represents the improvement in request failure and completed count using proposed algorithm as compared to existing algorithm without trust management.

3.4. Conclusion

The main achievement of this work is to find the rich literature and solve the issue of load balancing in fault aware cloud environment. In distributed file system, data is dispersed among different storage servers located geographically far away from each other. To provide the desired quality of service to the clients, performance of the distributed file system matters a lot. Response time is the major parameter that may affect the performance of the any distributed file system. Proposed approach 1 & 2 claims to reduce the delayed requests and also reduces the overall system response time. Approach 1 also considers the physical aspects of a server like available number of CPU cores in a server, request queue size or buffer to store the incoming client requests. Moreover approach 2 also considers the deadline of client requests to reduce request failure due to deadline. Obtained result shows the improvements over previously worked least loaded algorithm and more number of client requests are processed by the system without delay and in case of overloading and failure the load balance distribute the requests accordingly to neighbor servers. In this approach 3 different type of trust models and load balancing algorithm are been discussed with their drawbacks. To overcome the drawbacks a trust based fault aware load balancing algorithm is proposed which perform better then load balancing algorithm implemented in CloudSim. For future work this trust model may be compared with other models and see the improvement in the QoS.

The results obtained from our approaches are very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed load balancing algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, reduced average utilization, average delay and high request completion count.

CHAPTER 4

LEARNING BASED FAULT AWARE RESOURCE ALLOCATION

Distributed and cloud computing environments are presently slanting and more prevalent for computation by various organizations like Google, Amazon, Microsoft etc. as the cloud size increase there is huge expansion in power consumption over the server farms. Also with increase in request load over a datacenter increases the request failure probability. To overcome these issues request should be scheduled in more efficient manner, improving resource utilization, request failure count and reliability of system. Recent studies show that the failure probability of a server increases linearly with the increase of independent resources (processors), and result into request failure at datacenters. So to resolve this issue various approaches have been proposed to improve the performance of cloud environment.

In this section, we have proposed a set of learning based algorithms for task allocation to minimize the request failure and to improve QoS (Quality of Service) over a data center. Proposed approaches aim to provide a global best schedule with least scheduling time complexity. Proposed algorithms has proven to have better performance in term of load and request failure rate as compared to previously proposed task allocation algorithm for cloud IaaS.

4.1. Approach 1: Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure

To overcome these issues a fault aware learning based resource allocation algorithm is been proposed using Genetic algorithm. Genetic algorithm helps us to find a solution, which cannot be, achieved by any static or dynamic algorithm. Moreover fault tolerant genetic algorithm help to find a fittest solution in term of least makespan (Time taken to complete a request) and least request failure probability. Proposed algorithm uses Poisson probability distribution for random request failure at virtual machine i.e. at host and datacenter level. On the other hand, request failure over a datacenter may occur randomly due to storage, network failure or VM crashes. Based on fault over a datacenter and computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce request failure probability. According to algorithm collect the information of data center resources and capability, and the count of failure occurred over a period of time on a datacenter.

4.1.1. Proposed algorithm

Proposed FGA (Fault aware Genetic Algorithm) is divided into four phases which are as follows:

- a) Initialization
- b) Evaluation and selection
- c) Crossover
- d) Mutation

a) Initialization

In this phase we have a set of tasks ($T_1, T_2, T_3, T_4, T_5, T_6, \dots, T_n$) and a set of resources in term of virtual machine ($VM_1, VM_2, VM_3, VM_4, VM_5, \dots, VM_m$) are pre allocated on hosts in distributed datacenters. Here we initialize a set of sequences or schedules allocated randomly, each sequence acts as a chromosome for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for algorithm. Next population is initialized which is a

set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

b) Evaluation and selection

In this phase we evaluate the fitness value for each schedule in a population or chromosome, which depends up on the computing capability, total time taken to complete the schedule, average utilization and the failure probability of complete schedule. Fitness value is evaluated using a fitness function defined below.

Where

F_i : Faults occurred on a system over the time T

FR_i : Fault rate that is the number of request failed due to system failure over time t .

F_{Pi} : Failure probability over a Host i .

RE_i : Reliability of a Host i .

λ : Fault rate over a time T

Since faults over a datacenter are random in nature and follows Poisson distribution, which over a period of time t and $t + \Delta T$ can be defined as:

$$F_{Pi}(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(-\lambda(t + \Delta T))}{\exp(-\lambda t)} \quad (4.1)$$

$$F_{Pi}(t) = 1 - \exp(-\lambda \Delta t) \quad (4.2)$$

$$RP_i = e^{-\lambda t} = e^{t/m} \quad (4.3)$$

If

VM_MIPS_i : MIPS of i^{th} virtual machine

T_Leng_i : Length of i^{th} Task

Then the predicted time to complete a task T_i is defined:

$$T_Exei = \left(\frac{T_Leng_i}{VM_MIPS_i} \right) \quad (4.4)$$

$$Total_time = \sum_{i=1-n} \frac{T_Leng_i}{VM_MIPS_i} \quad (4.5)$$

The fitness value for a chromosome is defined by the fitness function gives as:

$$Fitness_chromosome_i = \alpha(Total_time_i) + \beta(FP_i) \quad (4.6)$$

Where

$$\alpha + \beta = 1 \quad (4.7)$$

Based on the fitness value of chromosome the fittest one is selected having least fitness value. The population is sorted based on the fitness value and best two are selected for next phase.

c) Crossover

In this step two fittest solutions based on least make span and failure probability is selected. We have used multi point crossover to generate new fittest schedule/ chromosome. This module is responsible for generation of new schedule/ chromosome by combining to selected having least fitness value and interchange two or more scheduled tasks between selected fittest schedules. The new generated schedule ins added to the existing population.

Steps to generate crossover are as follows.

1. The two fittest chromosomes are selected
2. A new fittest chromosome is generated using multi point cross over by interchanging the set of schedules between two chromosomes.
3. The new chromosome replaces the chromosome with highest fitness value.

d) Mutation

In this phase new merging the new offspring, and modifying the existing chromosomes with new solution. This forms a new set of schedules and population which form a better solution after each iteration. After specific count of iteration predefined as an input to genetic algorithm, best chromosome is selected i.e. the chromosome with least fitness value is selected for schedule.

Proposed algorithm

Fault Based Genetic Algorithm Task Allocation

Algorithm:-FGATA(VM List VM_i , Task list T_i , population size Po , Iteration Itr)

//Input : Po , VM_i , Itr and T_i

1. $VM_i \leftarrow VM_List()$;
2. $FI \leftarrow getFault()$;
3. $i \leftarrow \text{No. of VM}$
4. $T_i \leftarrow Task_List()$;
5. $C \leftarrow Genetic_algo (Vmi, Ti, Po, Itr)$;
6. $Allocate_Resource(C)$; // processing the client request.

Figure 4.1 Proposed FGS algorithm Initialization

Genetic Algorithm

Genetic_algo (VM_i, Ti, Po, Itr)

//Input : Po , VM_i , Itr and T_i

1. $Po \leftarrow Initiate_Population(T_i)$;
2. $Evaluation()$;
3. $C1 \leftarrow getFittest1()$;
4. $C2 \leftarrow getFittest2()$;
5. $Crossover(C1, C2)$
6. $Mutation(Po, C1, C2)$;
7. $Return(getFittest())$;
6. End

Figure 4.2 Proposed fault aware genetic algorithm.

1. $Evaluation()\{$
2. For each C_i $i=0 - po$
3. For each T_i
4. $temp = \alpha (T_i/VM_i) + \beta (FP_i)$
5. $Fitness_i = Fitness_i + temp$
6. End
7. END
8. }

Figure 4.3 Proposed FGA evaluation phase

```

1. Allocate_Resource(C){
2.  $C_i \leftarrow \text{Getchromosomes}()$ ;
3. For each  $C_i$ 
4.   Allocate( $C_i$ );
5. END
6. }

```

Figure 4.4 Proposed FGA allocation phase

Proposed algorithm provides a benefit over existing static scheduling algorithm, that it can search for best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in find a solution with similar high utilization and least failure probability.

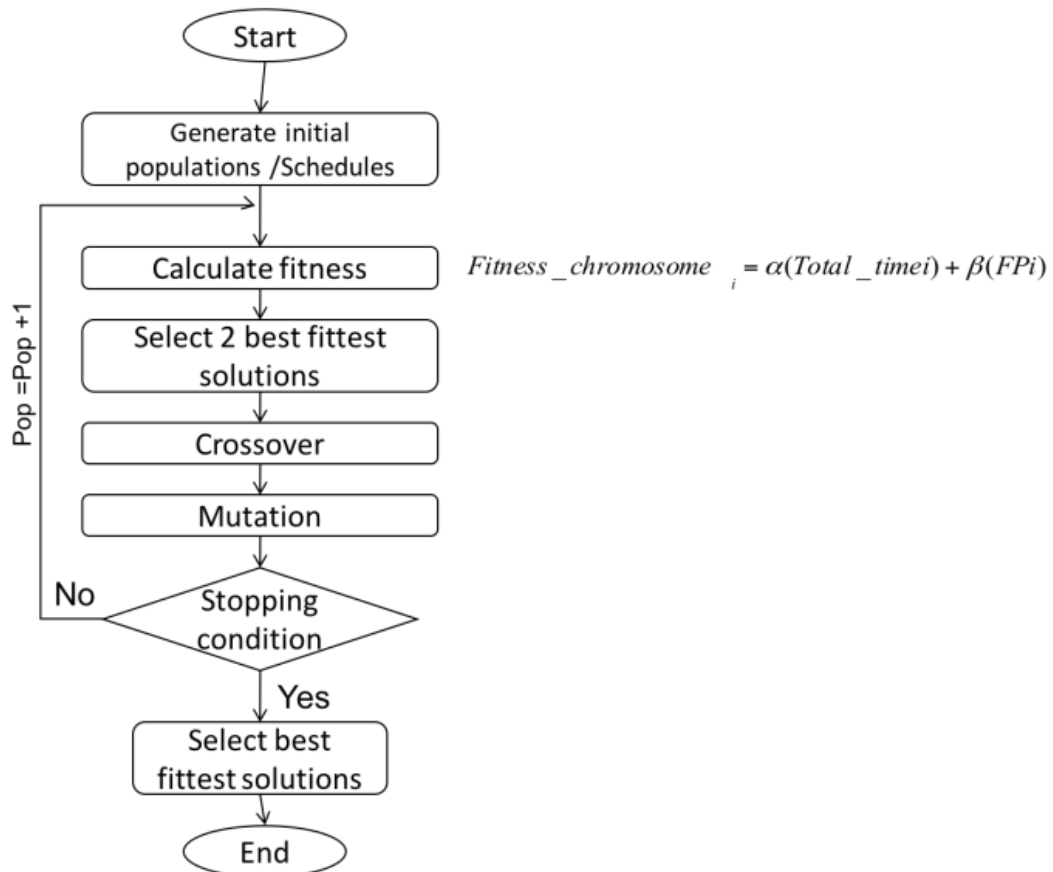


Figure 4.5 Proposed FGA flow diagram

4.1.2. Experiment and Results

For simulation CloudSim 3.0 API is used. CloudSim 3.0 [72] provides linear power model simulation to find the power consumption in cloud. Proposed fault aware genetic task allocation algorithm is implemented in CloudSim replacing existing task scheduling to find the global best schedule. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request and fault model in distributed environment.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [62]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown in figures below. Table 4.1 shows the environment specification and parameters used for simulation.

Table 4.1
Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	HOST
D0	2000	10000	100000	4	6	2
D1	2000	10000	100000	4	6	2
D2	2000	10000	100000	4	6	2
D3	2000	10000	100000	4	6	2
D4	2000	10000	100000	4	6	2
D5	2000	10000	100000	4	6	2
D6	2000	10000	100000	4	6	2
D7	2000	10000	100000	4	6	2
D8	2000	10000	100000	4	6	2
D9	2000	10000	100000	4	6	2

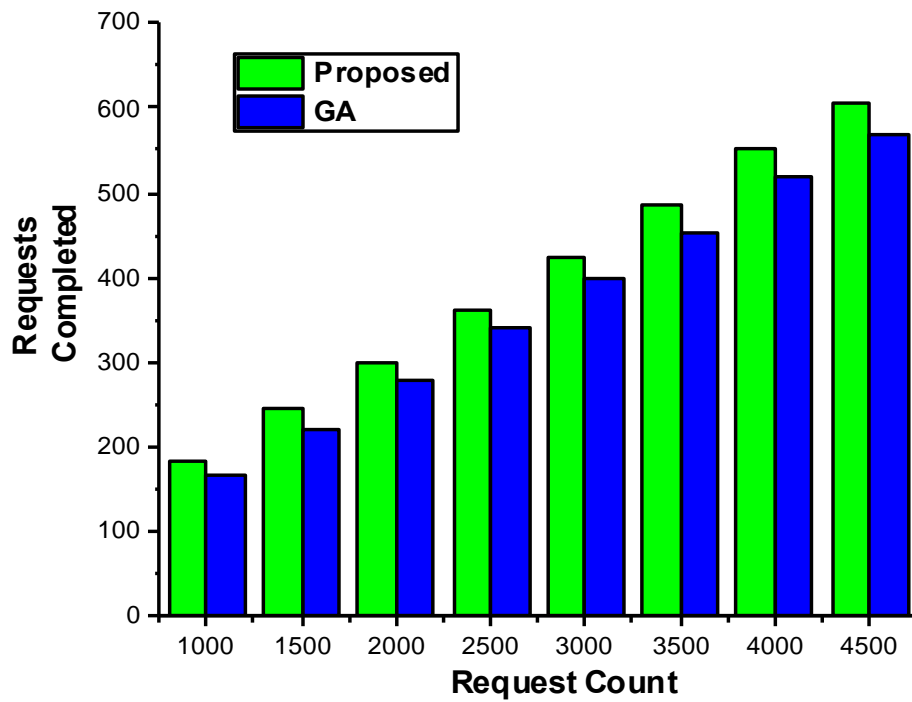


Figure 4.6 Comparison of improvement in request completed

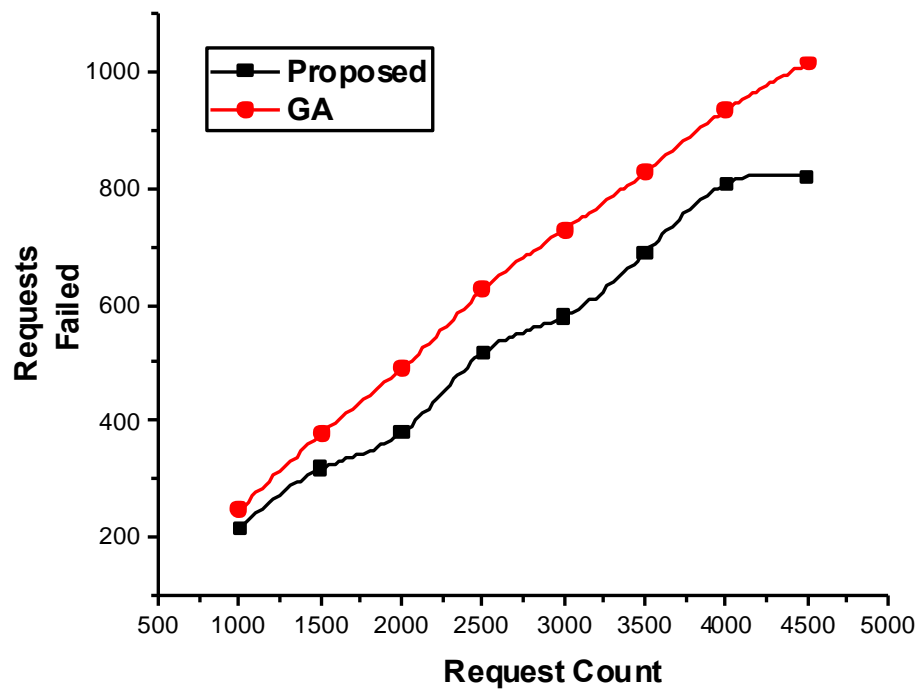


Figure 4.7 Comparison of improvement in request failed

Figure 4.6 & 4.7 compares the improvement in number of request failed and request competed with increase in number of requests over the system. The failure count reduces over the proposed system in increase in completed requests over the system.

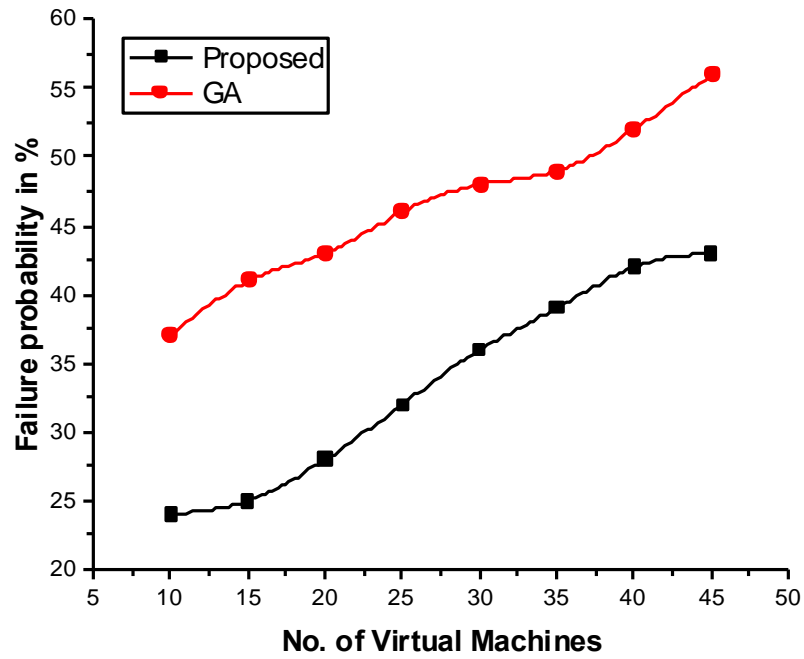


Figure 4.8 Comparison of failure probability with variable resources

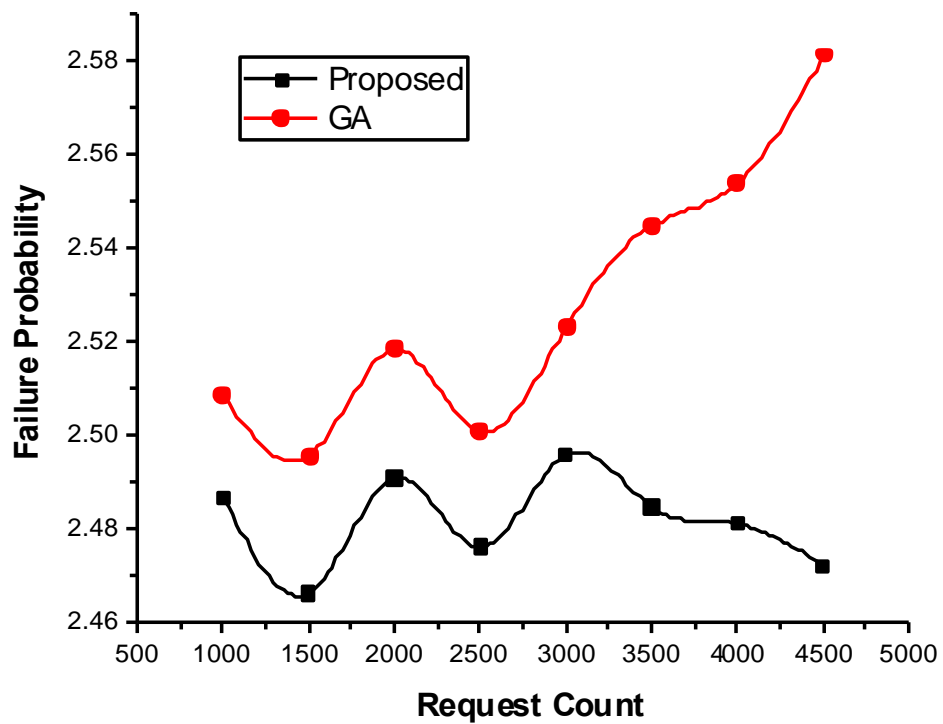


Figure 4.9 Comparison of failure probability with variable request

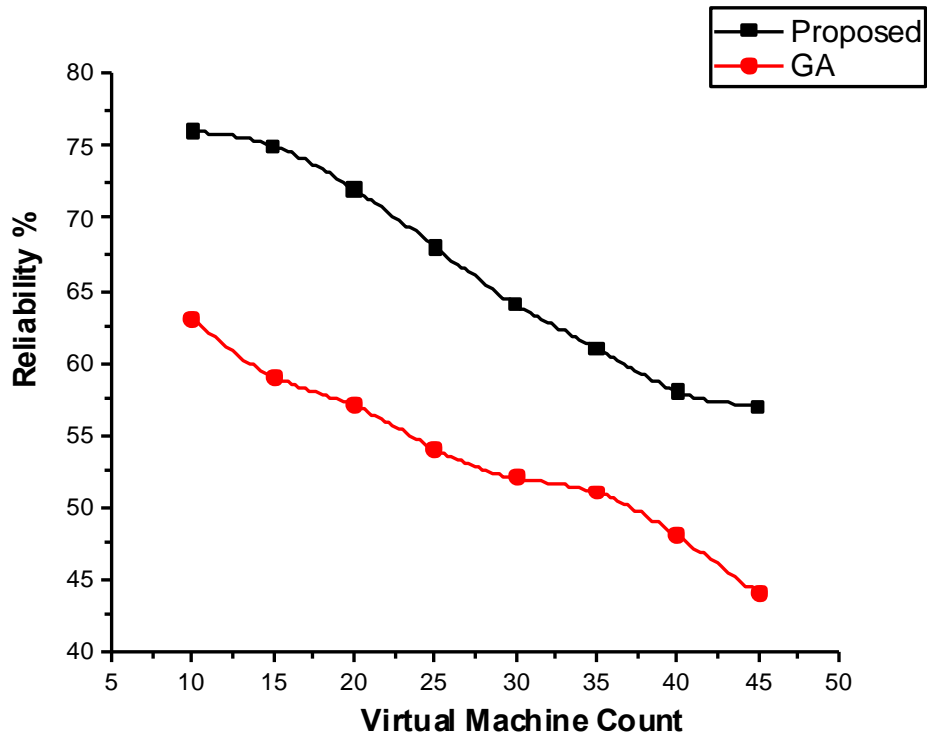


Figure 4.10 Comparison of reliability with variable resources

Figure 4.8 discourses the improvement in failure probability with increase in number of resources since with increase in number of VM's the probability of failure increases over the system. Figure 4.9 shows the improvement in failure probability with increase number of request count. Figure 4.10 & 4.11 shown the increase in reliability with increase number of VM's and request counts. Figure 4.11 shows improvement in reliability as the number of requests increases. Figure 4.12 shows the drawback of proposed algorithm with small increase in complete execution time as the number of request completed increases.

From experimental result section, it is clear that proposed fault aware GA (Genetic Algorithm) provides better QoS (Quality of service) as compared to previous proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete maximum number of requests with least failure probability, proposed algorithm shown that it can maximize reliability and minimize the number of request failed. This strategy has proven that it provides better QoS in term of high reliability with increase in number of requests and resources with failure probability.

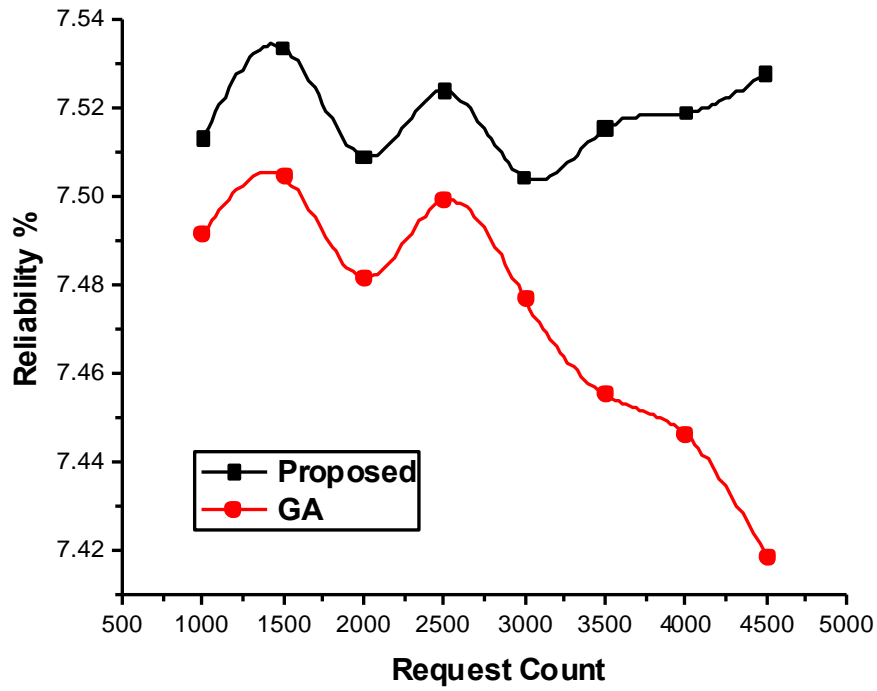


Figure 4.11 Comparison of reliability with variable requests

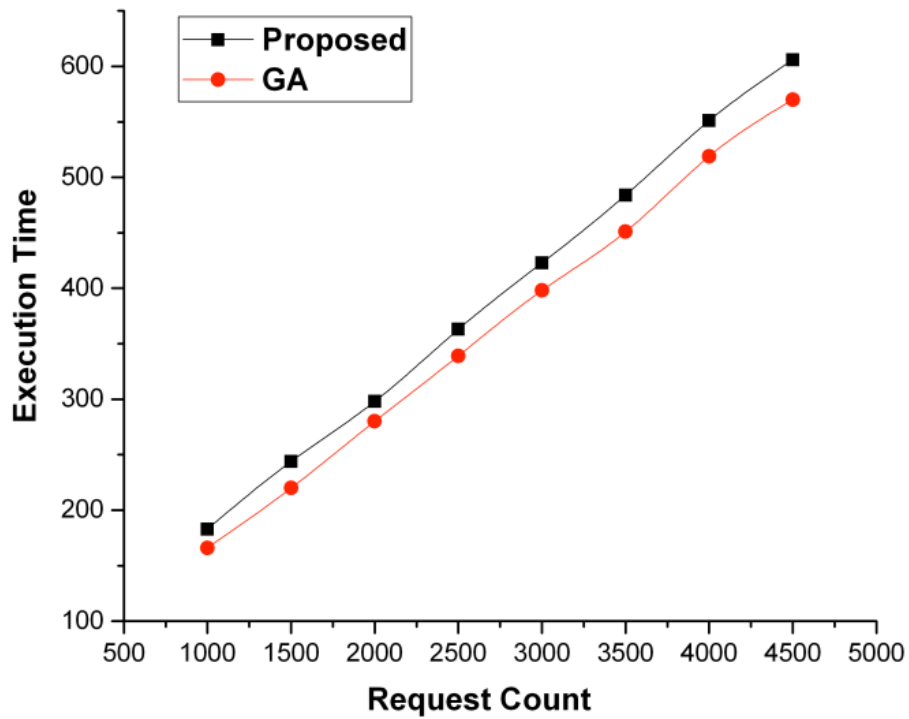


Figure 4.12 Comparison of execution time with variable resources

4.2. Approach 2: Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure

In this approach, we have proposed a task allocation algorithm based on Big Bang-Big Crunch (BBC) algorithm. This algorithm is motivated from the physics behind creation of universe theory in astrology. BBC algorithms refers to the evolution of universe and end of universe which says universe is a finite space which once expanded with force binding it and will end into a single point referred as a black hole. Algorithm also suggests that any element of universe cannot be suggested as center of universe. Similar to this we have proposed a task allocation algorithm to find a single best solution from a large set of solutions. Where generation of universe is referred as Big Bang phase and dissipation of universe in black hole near the center is said to big crunch phase.

Proposed algorithm uses Poisson probability distribution for random request at virtual machine i.e. at host and datacenter level. Based on computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce time complexity of solution. According to algorithm collect the information of data center resources and its capability. Proposed algorithm is similar to Genetic algorithm (GA) but the problem size reduces after each phase and will give you a single point solution i.e. the global solution. But in existing GA the population size remain same and there is no guarantee that the global best is achieved.

4.2.1. Proposed Algorithm

Existing proposed algorithms are either discuss about task scheduling or resource utilization and some of them talk about task or VM migrating to fulfill requests but the existing algorithms are static or dynamic in nature and they may suffer from local minima solution considering that as the best solution. But a better solution for task allocation may be possible. So to overcome these learning based algorithms were proposed like Genetic algorithm and PSO (particle swarm optimization). The issue with these algorithms is that they have very high time complexity more over they depend upon the iteration and the initial; population size, which affects their solution. If the population size of the iterations/generation are less then there less probability to get best solution.

Proposed algorithm is divided into four phases which are as follows:

- a) Big Bang / Initialization phase
- b) Evaluation phase
- c) Crossover / Center of mass
- d) Big Crunch phase

a) Initialization

In this phase we have a set of tasks ($T_1, T_2, T_3, T_4, T_5, T_6, \dots, T_n$) and a set of resources in term of virtual machine ($VM_1, VM_2, VM_3, VM_4, VM_5, \dots, VM_m$) are pre allocated on hosts in distributed datacenters. Here we initialize asset of sequences or schedules allocated randomly, each sequences act a chromosomes for genetic algorithm. The complete set of chromosomes is said to be a population, acting as a input for algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

b) Evaluation and selection

In this phase we evaluate the fitness value for each set of sequence or chromosome, which depends up on the computing capability, total time taken to complete the schedule. Fitness value is evaluated using a fitness function defined below.

Where

VM_MIPS_i : MIPS of i^{th} virtual machine

T_Lengh_i : Length of i^{th} Task

$Fitness_chomosom_e_i$: Fitness value of chromosome/sequence i

Then the predicted time to complete a task T_i is defined:

$$T_Exei = \left(\frac{T_Lenght_i}{VM_MIPS_i} \right) \quad (4.8)$$

$$Total_time = \sum_{i=1-n} \frac{T_Lenght_i}{VM_MIPS_i} \quad (4.9)$$

The fitness value for a chromosome is defined by the fitness function gives as:

$$Fitness_chromosome_i = \alpha(Total_time_i) \quad (4.10)$$

Based on the fitness value of chromosome the fittest one is selected having least fitness value. The population is sorted based on the fitness value and best two are selected from next phase.

c) Crossover

In this step two fittest solutions based on least make span are selected based on the center of mass and the population sequence near to center of mass are selected for cross over.

The steps for selection are as follows:

1. Find Center of mass of fitness values of the sequences in population using mean.
2. Find the sequence having fitness value with least difference from the center of mass.

We have used multi point crossover to generate new fittest sequences/ chromosome. Steps to generate crossover are as follows.

4. The two fittest chromosomes are selected with least difference from center of mass and one having least fitness value.
5. A new fittest chromosome is generated using multi point cross over by interchanging the set of schedules between two chromosomes.
6. The new chromosome replaces the chromosome with highest fitness value.

$$C_Mass = \frac{\sum_{i=0}^n Fitness_chromosome_i}{PopulationSize()} \quad (4.11)$$

d) Big Crunch phase

In this phase new merging the new offspring, which can be better solution from all other chromosomes/sequences. A new population is generated with new offspring generated and removing two chromosomes with least fitness value i.e. the worst solution from the population, decreasing the population size by one. These steps are repeated for number of iterations. After specific count of iteration of proposed algorithm stop the iterations, when the

population size is one. This is said to be the stopping condition of BBC and the last solution is the best solution for a definite time interval and iteration. Each iteration can also be referred as “generation” to create new fittest solution.

Proposed algorithm

Big Bang-Big Crunch Algorithm Task Allocation

Algorithm:-BBC (VM List VM_i , Task list T_i , population size Po , Iteration Itr)
 //Input: Po , VM_i , Itr and T_i
 1. $VM_i \leftarrow VM_List()$;
 2. $i \leftarrow \text{No. of VM}$
 3. $T_i \leftarrow Task_List()$;
 4. $C \leftarrow BBC_algo(Vmi, Ti, Po, Itr)$;
 5. Allocate_Resource(C); // processing the client request.
 7. End

Figure 4.13 Proposed BBC algorithm initialization

BBC Algorithm

BBC_algo(VM_i, Ti, Po, Itr)
 //Input: Po , VM_i , Itr and T_i
 1. $Po \leftarrow \text{Initiate_Population}(Ti,)$;
 2. Evaluation();
 3. CenterMass(); // find mean of all fitness values
 4. $C1 \leftarrow \text{getFittest1}()$;
 5. $C2 \leftarrow \text{getFittest2}()$;
 6. $Po \leftarrow \text{Crossover}(C1, C2)$
 7. Big_Crunch($Po, C1, C2$);
 8. Return(getFittest());
 6. End

Figure 4.14 Proposed BBC Algorithm

Evaluation

```
1. Evaluation(){
2.   For each  $C_i$   $i=0$  -  $p_o$ 
3.     For each  $T_i$ 
4.        $temp = (T_i/Vm_i)$ 
5.        $Fitness_i = Fitness_i + temp$ 
6.     End
7. END
8. }
```

Figure 4.15 Proposed Evaluation Phase

Big_Crunch

```
1. Big_Crunch( $P_o$  ,  $C1$ ,  $C2$ )
2. {
3.   Delete_worst(); // delete element with
   least fitness value.
4.   Delete_worst();
5. }
```

Figure 4.16 Big Crunch Phase

```
1. Allocate_Resource( $C$ ){
2.  $C_i \leftarrow \text{Getchromosomes}()$ ;
3. For each  $C_i$ 
4.   Allocate( $C_i$ );
5. END
6. }
```

Figure 4.17 Allocation Phase

Proposed algorithm provides a benefit over existing static scheduling algorithm, that it can search for best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in find a solution with similar high utilization and less time complexity as compared to Genetic algorithm. Figure 4.18 shows the flow diagram for proposed algorithm and interaction between each module.

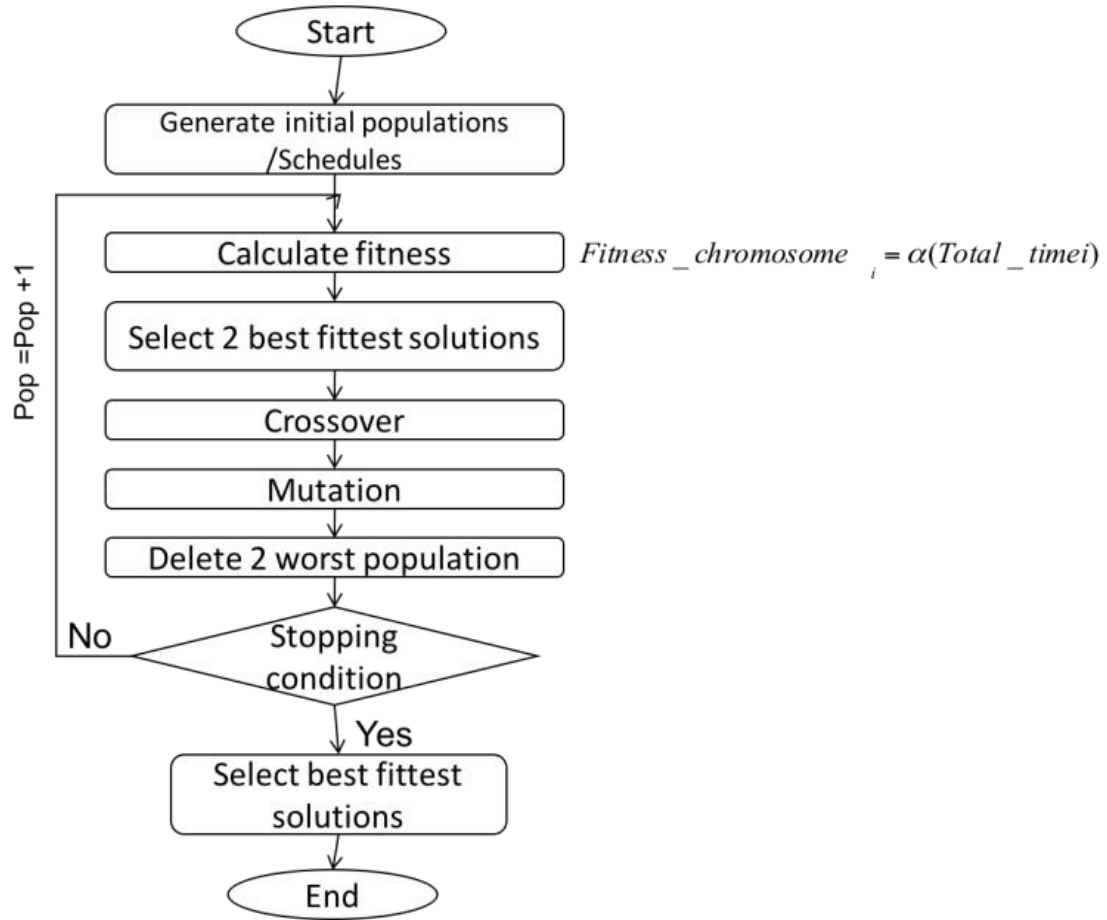


Figure 4.18 Proposed big bang big crunch algorithm flow diagram

4.2.2. Experiment and Results

Simulation has been performed on a simulation test bead using CloudSim 3.0 [72] tool kit for cloud simulation. CloudSim provides a cloud infrastructure environment with all environmental parameter to study the performance of cloud. Proposed Big Bang Big Crunch algorithm for task allocation is implemented in CloudSim replacing existing Round Robin algorithm. The algorithm aims to reduce the scheduling time and find an global best schedule with least make span. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request in distributed environment, with each server having two hosts each.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [62]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown

in figures below. Table 4.2 shows the environment specification and parameters used for simulation.

Table 4.2: Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	HOST
D0	2000	10000	100000	4	6	2
D1	2000	10000	100000	4	6	2
D2	2000	10000	100000	4	6	2
D3	2000	10000	100000	4	6	2
D4	2000	10000	100000	4	6	2
D5	2000	10000	100000	4	6	2
D6	2000	10000	100000	4	6	2
D7	2000	10000	100000	4	6	2
D8	2000	10000	100000	4	6	2
D9	2000	10000	100000	4	6	2

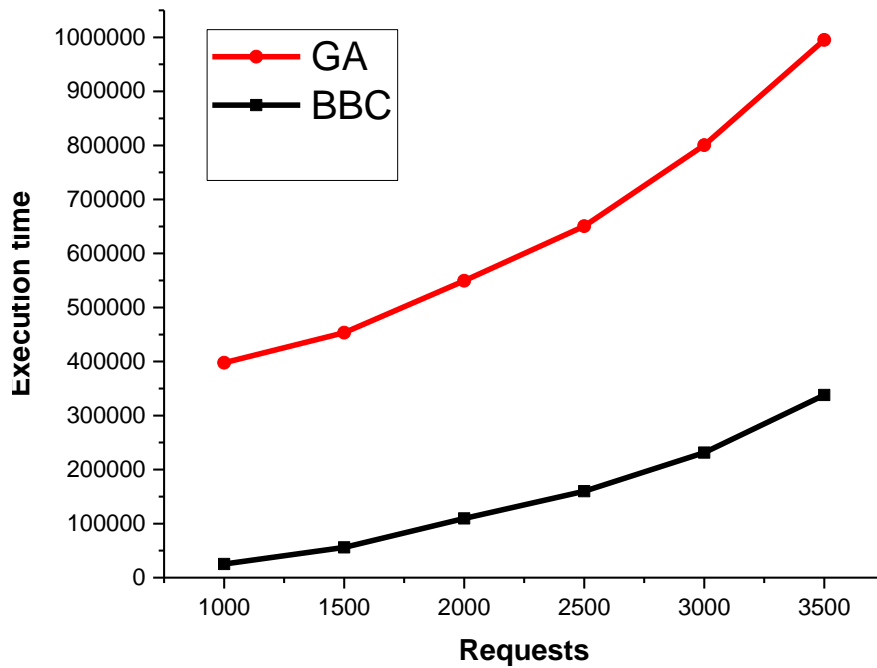


Figure 4.19 Comparison of improvement in execution time

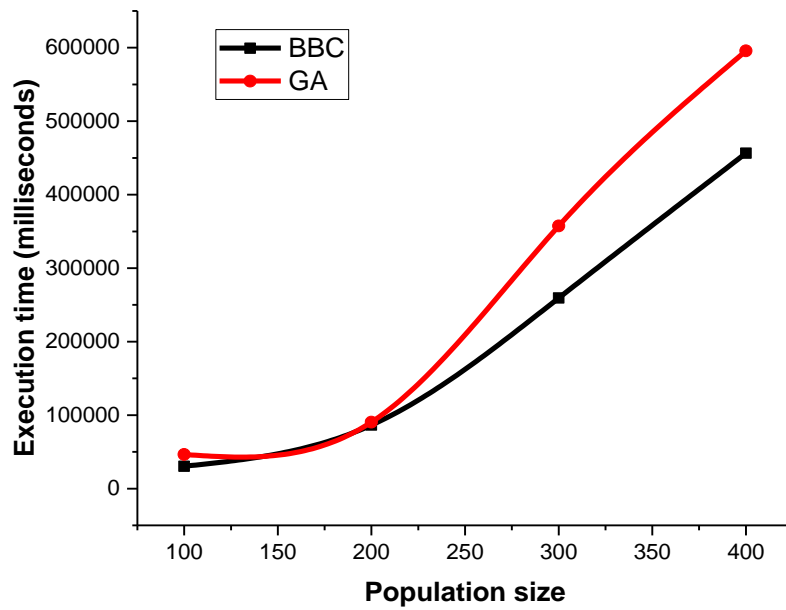


Figure 4.20 Comparison of improvement in execution time with changes in population size.

Figure 4.19 compares the improvement in execution time with increase in number of requests over the system. Figure 4.20 shows the improvement in execution time with increasing population size. Execution time has reduced over the proposed system with increase in completed request count over the system.

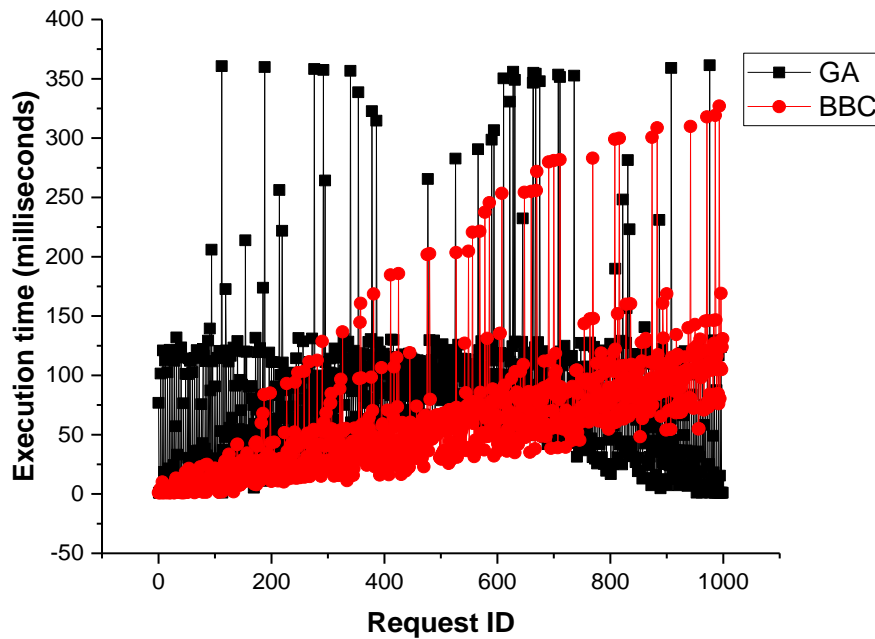


Figure 4.21 Comparison of execution time of individual requests.
For 1000 request count

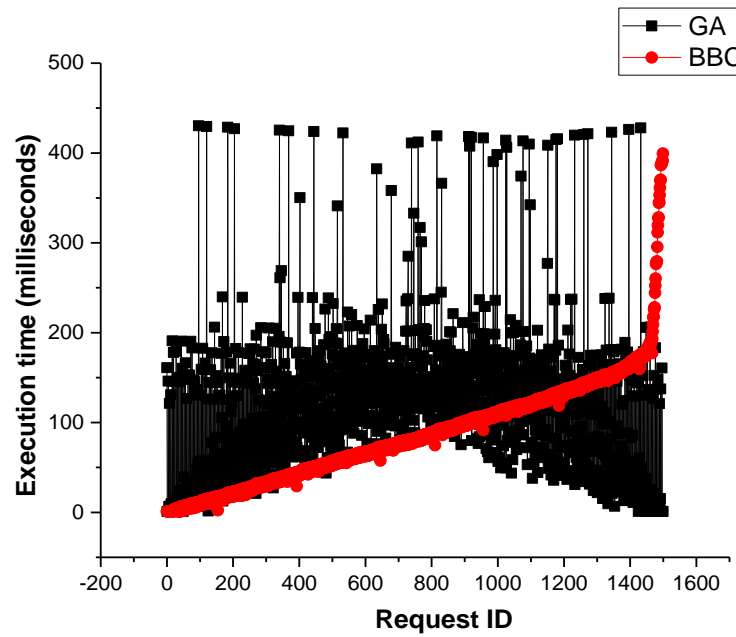


Figure 4.22 Comparison of execution time of individual requests.
For 1500 request count

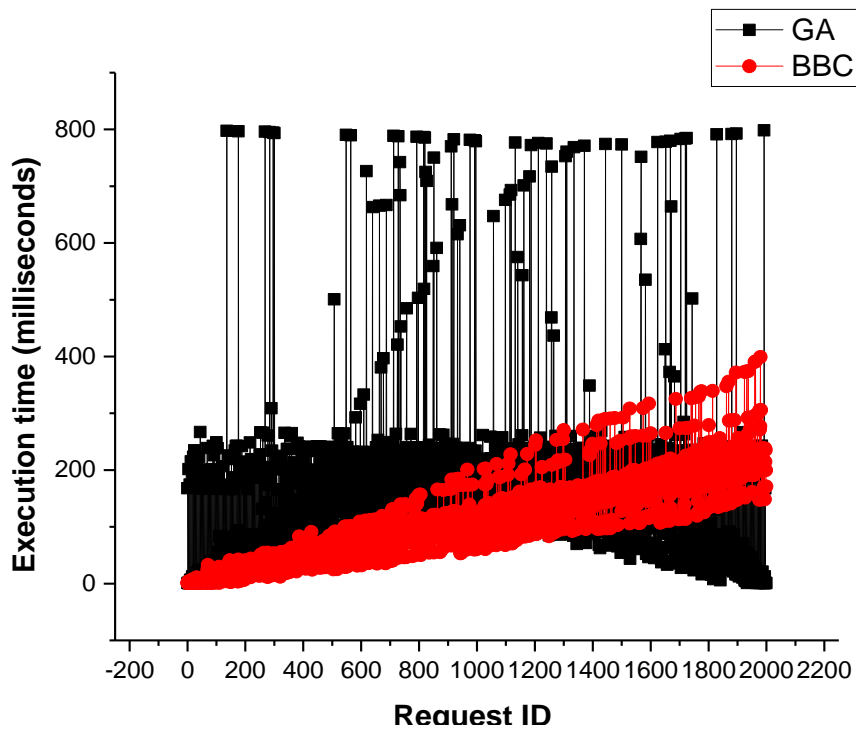


Figure 4.23 Comparison of execution time of individual requests.
For 2000 request count

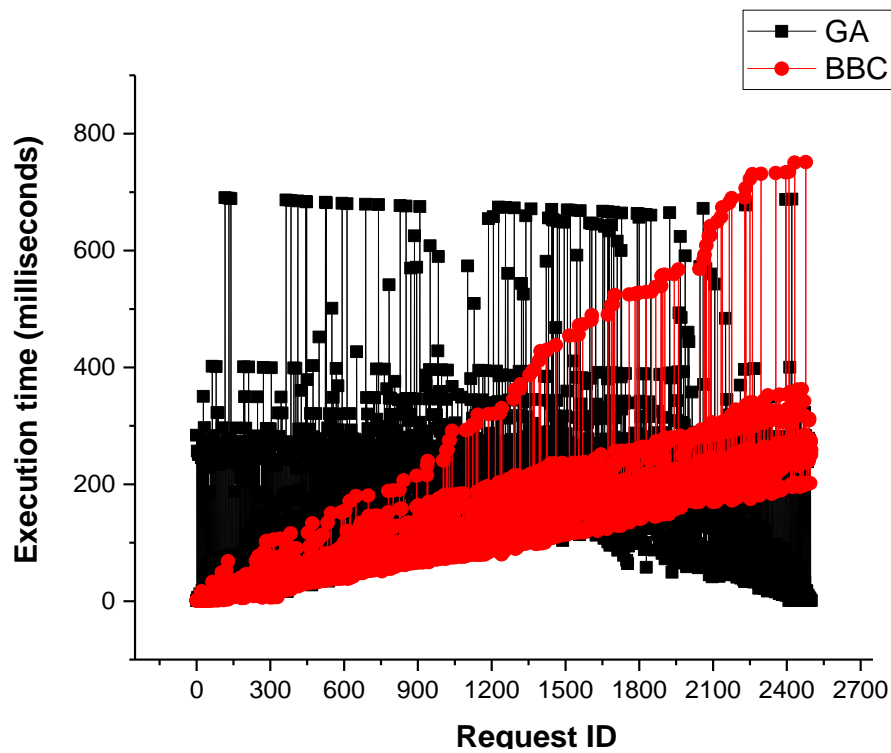


Figure 4.24 Comparison of execution time of individual requests.
For 2500 request count

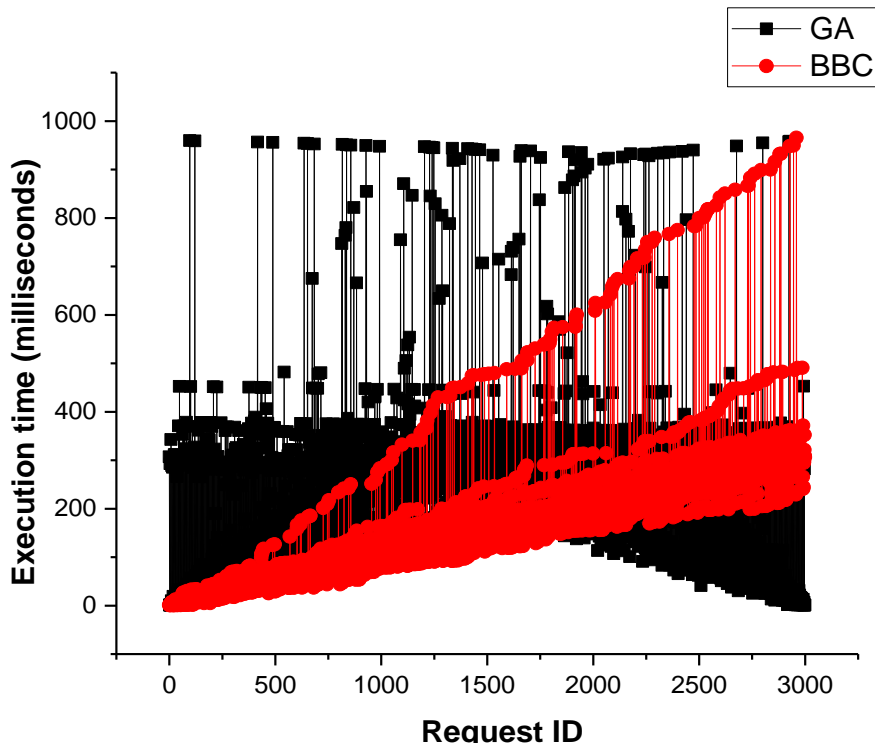


Figure 4.25 Comparison of execution time of individual requests.
For 3000 request count

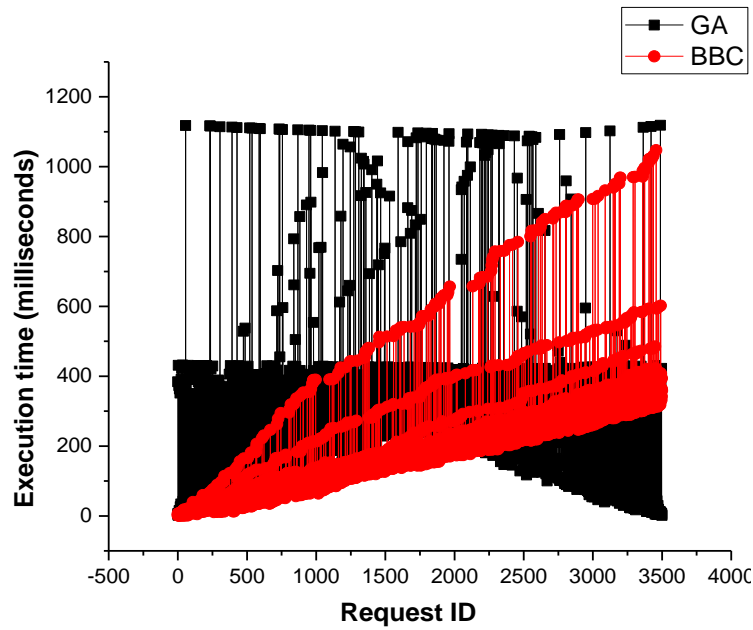


Figure 4.26 Comparison of execution time of individual requests.

For 3500 request count

Figure 4.21, 4.22, 4.23, 4.24, 4.25 & 4.26 shows the improvement in distribution of execution time for requests using proposed BBC algorithm for task allocation over cloud. The execution time of the requests has improved and majority of requests are completed in small execution time as compared to genetic algorithm.

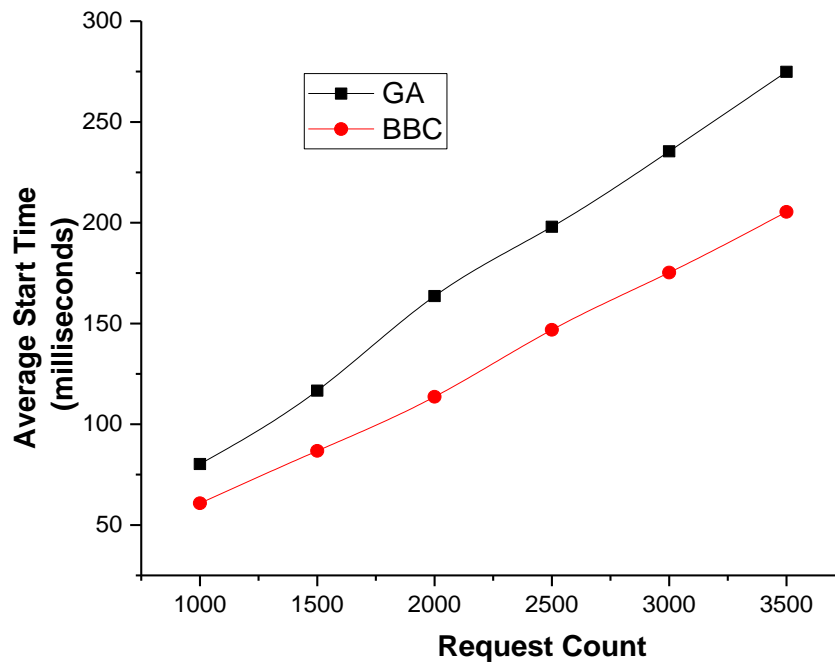


Figure 4.27 Comparison of Average Start time of system with increase in request count

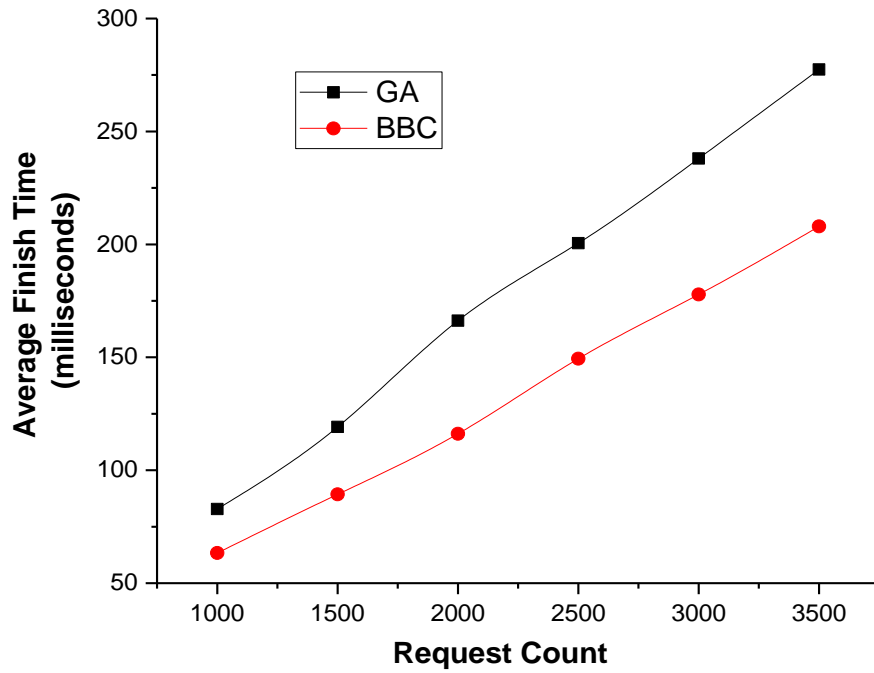


Figure 4.28 Comparison of Average Finish time of system with increase in request count

Figure 4.27 discourses the improvement in average start time with increase in number of request over the system, which shows the proposed algorithm proves to provide better start time than the conventional genetic algorithm. Figure 4.28 discourses the improvement in average finish time which reduces with increasing requests over the system the experiment has been performed over 1000, 1500, 2000, 2500, 3000 and 3500 request count. Proposed algorithm proves to provide reduced finish time as compared to existing genetic algorithm.

From experimental result section, it is clear that proposed BB-BC provides better QoS (Quality of service) as compared to previous proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete maximum number of requests with least execution time, proposed algorithm shown that it can provide better execution time over large requests with reduces average start time and average finish time over the system. Proposed algorithm reduces the number of iteration required to achieve a global best solution with least scheduling time. This strategy has proven that it provides better QoS in term of high reliability with increase in number of requests and resources with least scheduling time with decrease in execution time with increase in population size and number of requests. Proposed algorithm insures the schedule achieve is global best solution.

4.3. Approach 3: Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure

In this approach, we have proposed a fault aware Big Bang-Big Crunch (BBC) algorithm for task allocation in cloud infrastructure. The algorithm is motivated from Big Bang-Big Crunch (BBC) theory of creation of universe in astrology. BBC algorithm is similar to the algorithm as proposed in previous approach.. Similar to this we have proposed a task allocation algorithm to find a single best solution from a large set of solutions but in a faulty cloud environment. Where generation of universe is referred as Big Bang phase and dissipation of universe in black hole near the center is said to big crunch phase. Proposed algorithm aim to improve the performance of task allocation algorithm and reduce the request failure count. Proposed algorithm improves the reliability of system and finds the global schedule for tasks.

4.3.1. Proposed Algorithm

Existing algorithm and above proposed approached either take into consideration improvement in scheduling delay or fault tolerant behavior of cloud. Some other existing approaches take into consideration VM migration of better task management, cost improvement and power efficiency, which are static or dynamic in nature. These algorithm suffers from either cannot find global best solution or request failure in cloud. Proposed algorithm tries to improve both the parameters together. Algorithm is been tested with variable iterations and population sizes with different cloud infrastructures.

Proposed algorithm is divided into four phases which are as follows:

- a) Big Bang / Initialization phase
- b) Evaluation phase
- c) Crossover / Center of mass
- d) Big Crunch phase

a) Initialization

In this phase we have a set of tasks (T1, T2, T3, T4, T5, T6.... T n) and a set of resources in term of virtual machine (VM1, VM2, VM3, VM4, VM5.... VM m) are pre allocated on hosts in distributed datacenters. Here we initialize asset of sequences or schedules allocated randomly, each sequences act a chromosomes for genetic algorithm. The complete set of chromosomes is said to be a population, acting as a input for algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

b) Evaluation and selection

In this phase we evaluate the fitness value for each set of schedule or chromosome, which depends up on the computing capability, total time taken to complete the schedule and the failure probability of schedule.

Where

F_i : Faults occurred on a system over the time T

FR_i : Fault rate that is the number of request failed due to system failure over time t.

FP_i : Failure probability over a Host i.

RE_i : Reliability of a Host i.

λ : Fault rate over a time T

Since faults over a datacenter are random in nature and follows Poisson distribution, which over a period of time t and t+ ΔT can be defined as:

$$FP_i(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(-\lambda(t + \Delta T))}{\exp(-\lambda t)} \quad (4.12)$$

$$FP_i(t) = 1 - \exp(-\lambda \Delta t) \quad (4.13)$$

$$RP_i = e^{-\lambda t} = e^{t/m} \quad (4.14)$$

Equation 4.12 shows the fault over a time T and Δt using Poisson probability distribution.

Equation 4.14 represents the evaluation of reliability for a system.

If

VM_MIPS_i: MIPS of ith virtual machine

T_Lengi: Length of ith Task

Fitness_chromosome_i: Fitness value of chromosome/sequence i

Then the predicted time to complete a task T_i is defined:

$$T_{_}Exei = \left(\frac{T_Lenght\ i}{VM_MIPSi} \right) \quad (4.15)$$

$$Total_time = \sum_{i=1-n} \frac{T_Lenght\ i}{VM_MIPSi} \quad (4.16)$$

The fitness value for a chromosome is defined by the fitness function gives as:

$$Fitness_chromosome_i = \alpha(Total_timei) + \beta(FPi) \quad (4.17)$$

$$\alpha + \beta = 1 \quad (4.18)$$

Based on the fitness value of chromosome the fittest one is selected having least fitness value. The population is sorted based on the fitness value and best two are selected from next phase.

c) *Crossover*

In this step two fittest solutions based on least fitness value are selected based on the center of mass and the population sequence near to center of mass are selected for cross over. The steps for selection are as follows:

1. Find Center of mass from the sequences in population using mean.
2. Find the sequence having fitness value with least difference from the center of mass.
3. The selected sequence is used for generation of next fit element. The selected sequence be S1.
4. Select a second best sequence having least fitness value. The selected sequence be S2.

We have used multi point crossover to generate new fittest sequences/ chromosome. Steps to generate new fittest sequence using crossover are as follows.

1. A new fittest chromosome is generated using multi point cross over by interchanging the set of schedules between two chromosomes.
2. The new chromosome replaces the chromosome with highest fitness value chromosome.

$$C_Mass = \frac{\sum_{i=0}^n Fitness_chromosome_i}{PopulationSize()} \quad (4.19)$$

These steps help to find the global best solution as in each iteration the solution moves the mean toward the best solution by using crossover and generation new best solution.

d) Big Crunch phase

In this phase the new offspring generated by merging the two best solutions, can be better solution than all existing chromosomes/sequences. A new population is generated with new offspring generated in previous step and removing the chromosomes with highest fitness value i.e. the worst solution from the population, decreasing the population size by one. Repeat steps b, c & d and stop the iterations, when the population size is one or the integration count is zero. This is said to be the stopping condition of BBC and the last solution is the best solution for a definite time interval and iteration. Each iteration can also be referred as “generation” to create new fittest solution.

Proposed algorithm

Fault Big Bang-Big Crunch Algorithm Task Allocation

Algorithm:-BBC (VM List VM_i , Task list T_i , population size Po , Iteration Itr)
 //Input: Po , VM_i , Itr and T_i
 1. $VM_i \leftarrow VM_List()$;
 2. $i \leftarrow \text{No. of VM}$
 3. $T_i \leftarrow Task_List()$;
 4. $C \leftarrow BBC_algo(Vmi, Ti, Po, Itr)$;
 5. Allocate_Resource(C); // processing the client request.
 7. End

Figure 4.29 Proposed FBBC algorithm initialization

BBC Algorithm

```
BBC_algo(VMi,Ti, Po, Iteration)
//Input: Po, VMi, Itr and Ti
1. Po ← Initiate_Population(Ti,);
2. While (Iteration >0)
3. {   Evaluation_fitness();
4.     CenterMass();// find mean of all fitness values
5.     C1 ← getFittest1();
6.     C2 ← getFittest2();
7.     C3 ← Crossover(C1,C2)
8.     Big_Crunch(C3 );
9.     iteration--;
10. }
11. Return( getFittest());
6. End
```

Figure 4.30 Proposed FBBC algorithm

Evaluation

```
1. Evaluation_fitness(){
2. For each Ci i=0 to po // For each
   population
3.   Fitnessi = Make_span() +TotalFault();
4.   End
5. END
6. }
```

Figure 4.31 Proposed FBBC evaluation phase

Fitness

```
1. getFittest1()
2. {
3. fitness_mean=0;
4. mass_diff=0;
5. mass_diff_t=0;
6. // Loop through individuals to find fittest
7. for (int i = 0; i < populationSize(); i++)
8. {
9.   fitness_mean= fitness_mean+ tours[i].getFitness();
10. }
11. fitness_mean=fitness_mean/populationSize();
12. mass_diff=tours[0].getFitness()-fitness_mean;
13. for (int i = 0; i < populationSize(); i++)
14. {
15.   mass_diff_t=fitness_mean-tours[i].getFitness();
16.   if (mass_diff >= mass_diff_t)
17.   {
18.     fittest = getTour(i);
19.     mass_diff=mass_diff_t;
20.   }
21. }
22. return fittest;
23. }
```

Figure 4.32 Get Fittest with least difference from Center of mass

```

1. getFittest2 ( )
2. {
3. // Loop through individuals to find fittest
4.   for (int i = 1; i < populationSize(); i++)
5.   {
6.     if (fittest.getFitness() >= getpopu(i).getFitness()){
7.       fittest = getpopu(i);
8.     }
9.   }
10.
11.   return fittest;
12. }

```

Figure 4.33 Get Fittest with least fitness value

Big_Crunch

```

1. Big_Crunch(C3)
2. {
3. add C3 to existing population by
  replacing it with worst solution.
4. Delete_worst(); // delete element with
  least fitness value.
5. }

```

Figure 4.34 Big Crunch Phase

```

1. Allocate_Resource(C){
2.  $C_i \leftarrow \text{Getchromosomes}()$ ;
3. For each  $C_i$ 
4.   Allocate( $C_i$ );
5. END
6. }

```

Figure 4.35 Proposed FBBC allocation phase

Proposed algorithm provides a benefit over existing static scheduling algorithm, that it can search for best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in find a solution with similar high utilization, least failure probability, high reliability and less time complexity as compared to Genetic algorithm. Figure 4.36 shows the flow of algorithm and interaction among various phases of task allocation.

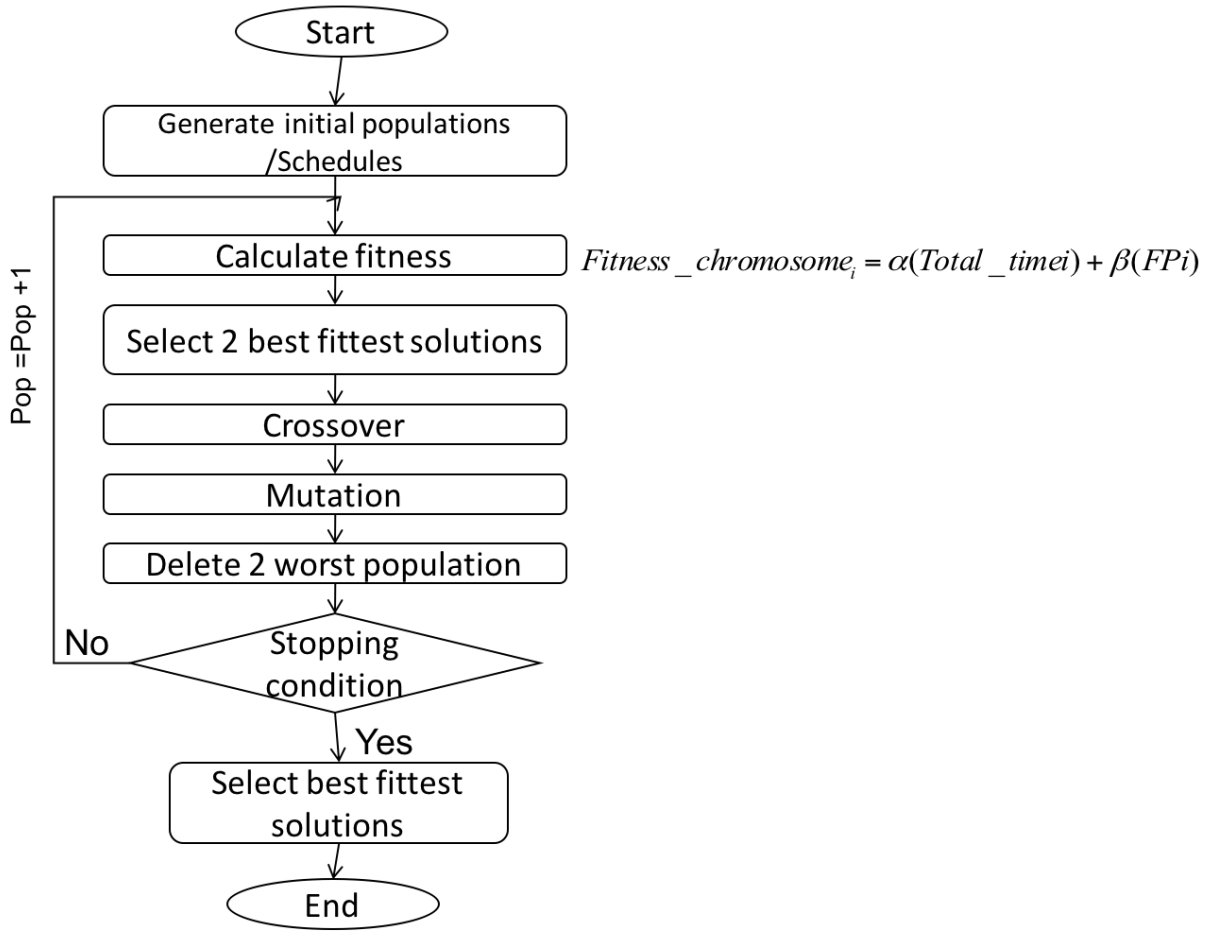


Figure 4.36 Proposed fault aware big bang big crunch algorithm

4.3.2. Experiment and Results

Simulation has been performed on a simulation test bead using CloudSim 3.0 [72] tool kit for cloud simulation. CloudSim provides a cloud infrastructure environment with all environmental parameter to study the performance of cloud. Proposed fault aware Big Bang Big Crunch algorithm for task allocation is implemented in CloudSim replacing existing Round Robin algorithm. The algorithm aims to reduce the scheduling time and find an global best schedule with least make span. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request in distributed environment, with each server having two hosts each.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [62]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown

in figures below. Table 4.3 shows the environment specification and parameters used for simulation.

Table 4.3
Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	HOST
D0	2000	10000	100000	4	6	2
D1	2000	10000	100000	4	6	2
D2	2000	10000	100000	4	6	2
D3	2000	10000	100000	4	6	2
D4	2000	10000	100000	4	6	2
D5	2000	10000	100000	4	6	2
D6	2000	10000	100000	4	6	2
D7	2000	10000	100000	4	6	2
D8	2000	10000	100000	4	6	2
D9	2000	10000	100000	4	6	2

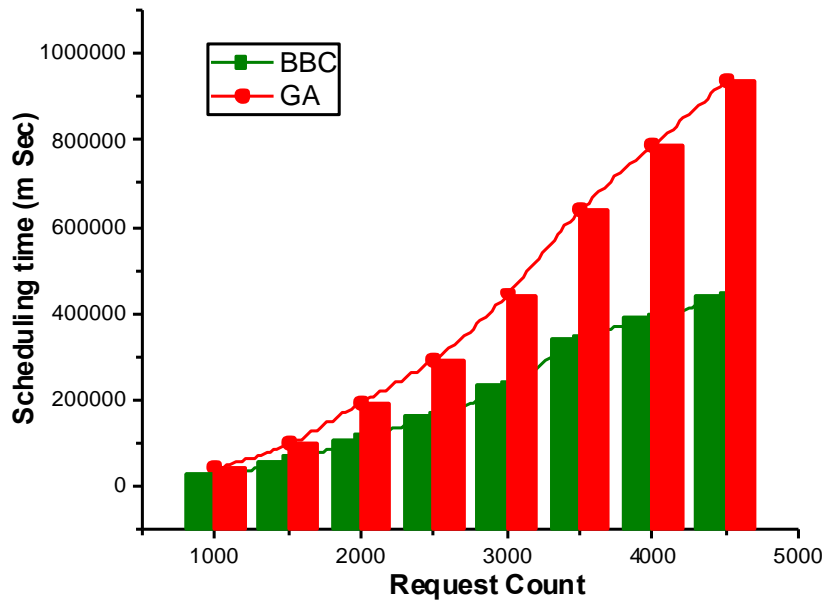


Figure 4.37 Comparison of improvement in scheduling time

Figure 4.37 shows the improvement in time takes to find a best schedule to allocate resources. In this figure both algorithms are learning based algorithm but proposed BBC algorithm proves to have less scheduling time. Figure 4.38 & 4.39 compares the improvement in

number of request failed and request competed with increase in number of requests over the system. The failure count reduces over the proposed system with increasing request count and proposed algorithm also shows improvement completed request count over the system.

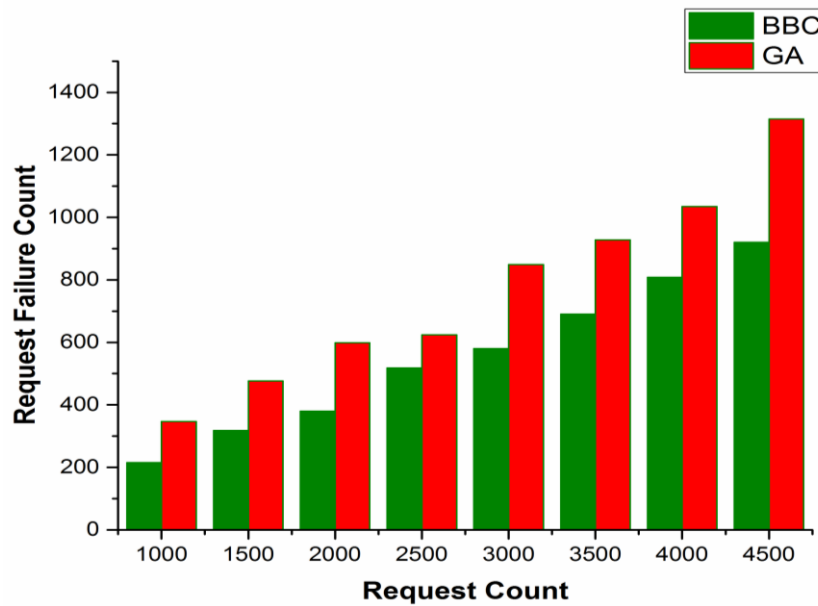


Figure 4.38 Comparison of improvement in request failed

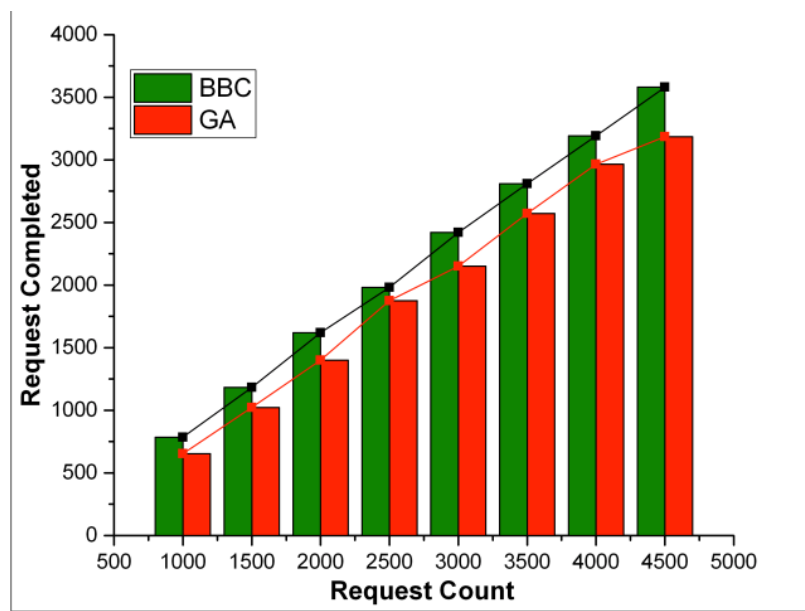


Figure 4.39 Comparison of improvement in request completed

Figure 4.40 discourses the improvement in failure probability with increase in number of resources, since with increase in number of request the probability of failure increases over

the system. Figure 4.41 shows the improvement in reliability with increase number of request count over a system.

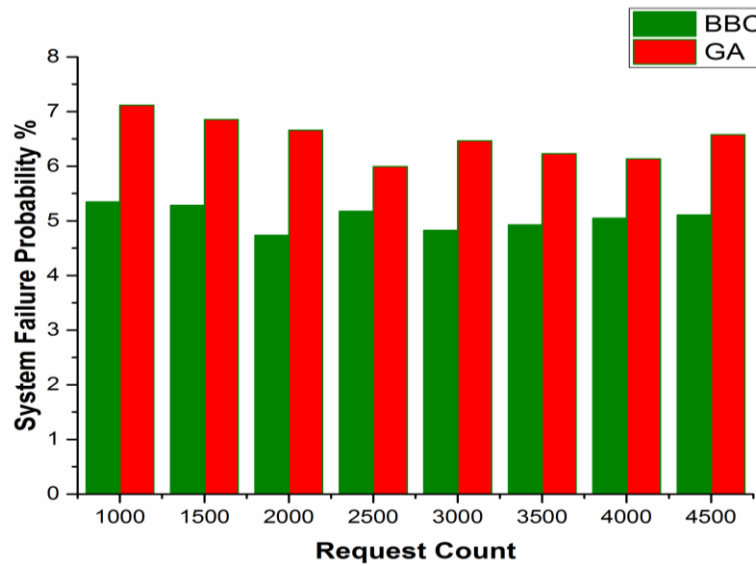


Figure 4.40 Comparison of failure probability with variable request count

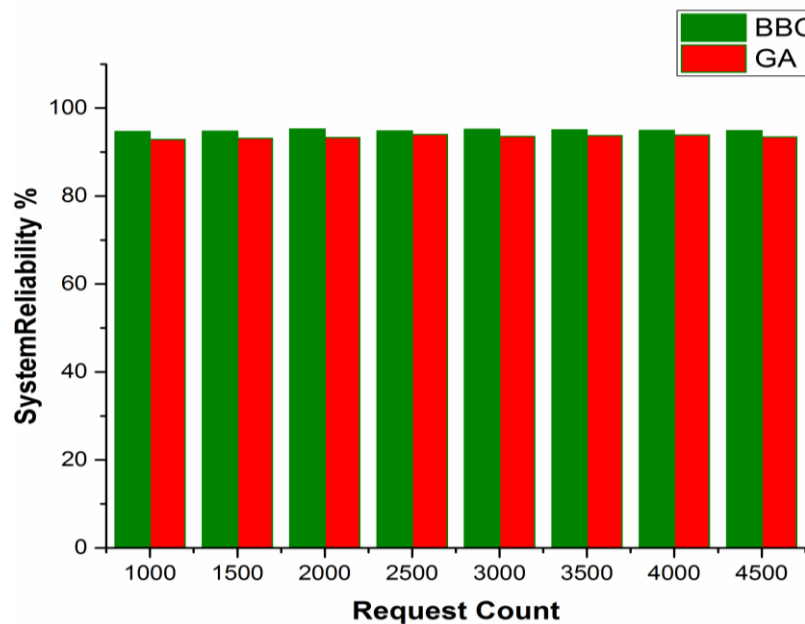


Figure 4.41 Comparison of reliability with variable request count

Figure 4.42 shows the drawback of proposed algorithm with small increase in total execution time. Overall result shows that the proposed algorithm improves the fault tolerant behavior of

system by reducing the request failure count of the system and improving the reliability of the system.

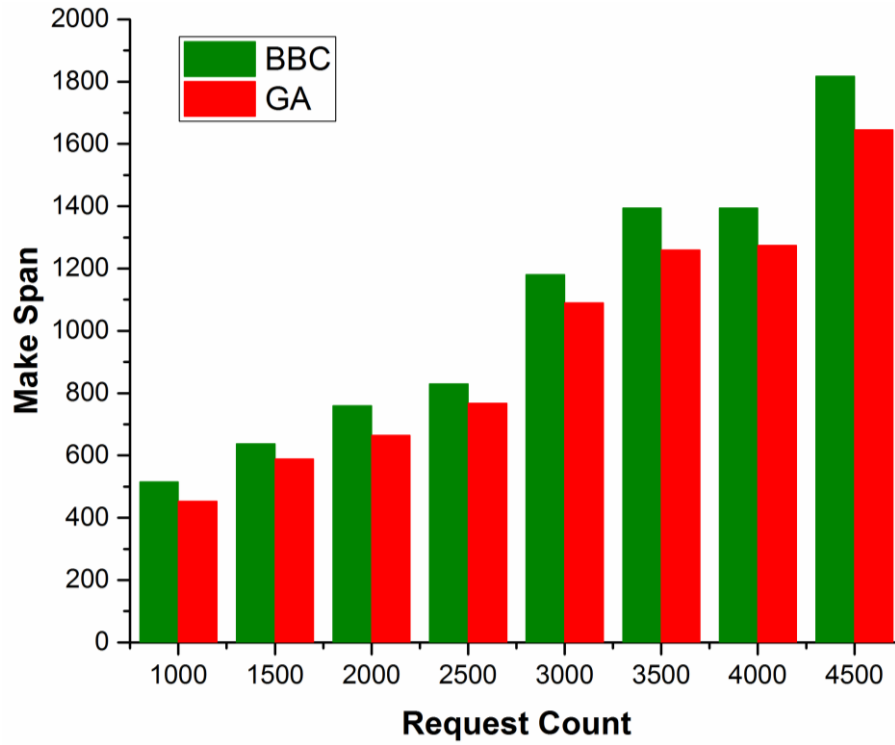


Figure 4.42 Comparison of execution time with variable request count.

From experimental result section, it is clear that proposed fault aware BBC provides better QoS (Quality of service) as compared to previous proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete maximum number of requests with least failure probability, proposed algorithm shown that it can maximize reliability and minimize the number of request failed. This strategy has proven that it provides better QoS in term of high reliability with increase in number of requests and resources with failure probability

4.4. Approach 4: Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure

There exist many load balancing algorithms proposed for grid and distributed computing environments [30]. But they do not take into consideration cloud as non faulty and QoS of datacenter. There are many cloud IaaS frameworks that provide cloud computing services and virtualization services to the user like OpenNode [73], CloudStack [74], Eucalyptus [75], CloudSigma [76], EMOTIVE (Elastic Management of Tasks in Virtualized Environments) [77] and Archipel.[78]

There are many solution been proposed over the time based on priority, cost, rank based which is used in OpenNebula [79] and round robin and power aware scheduling algorithm used in Eucalyptus and many more. But they do not take in to consideration the QoS parameters of the datacenters like fault rate, initialization time, MIPS and many more. So to overcome this issue and make system more reliable, fault and load aware honey bee scheduling algorithm is proposed.

4.4.1 Proposed Algorithm

Proposed algorithm is inspired for natural behavior of honey bee to find the best solution for designing optimal scheduling algorithm. The algorithm requires a number of parameters to be set, specifically: quantity of scout bees (n), quantity of nice websites out of m selected sites (e), number of websites selected out of n visited sites (m), number of bees recruited for high-quality e web sites, number of bees recruited for the opposite ($m-e$) selected sites, initial size of patches which includes site and its neighborhood and stopping criterion

Steps of proposed algorithm are as follows:

Step1. Initialize scout bees equal to number of datacenters.

Step2. Recruit scout bees for selected sites (more bees for best e sites) and evaluate fit nesses value for datacenter.

Step4. Assign bees to search randomly and evaluate their fit nesses for a request.

Step4. Stop when all bees have arrived, else wait.

Step5. Select the fittest bee from each datacenter.

Step6. Assign remaining bees to search randomly and evaluate their fit nesses for each request.

Step7. End While no request in queue.

In first step, the bee algorithm starts with the scout bees (n) being placed randomly in the search space. In step 2, the algorithm conducts searches in the neighborhood of the selected sites, assigning more bees to search near to the best ‘e’ sites i.e. search for new datacenters. In step 3 the fit nesses of the datacenters visited by the scout bees are evaluated. In step 4 waiting until all bees are arrived. In step 5, 6 bees that have the highest fit nesses are chosen as “selected bees” and sites visited by them are chosen for allocation of resources. In step 7 repeat all above steps until there is request in queue.

Most complicated part of this algorithm is fitness value calculation. Proposed algorithm take into consideration parameters of datacenter which are used for calculating fitness value for a datacenter are as follows.

- a) Initiation Time: How long it takes to deploy a VM.
- b) System load: Number of busy or allocated Machine Instruction per Second (MIPS) of a datacenter.
- c) Network load: Allocated network bandwidth out of total available bandwidth provided.
- d) Fault Rate: It is defined as the number of faults over a period of time.

In above mentioned parameters allocated MIPS (MP) and Bandwidth of a datacenter changes as the number of virtual machine allocated on a datacenter changes, but fault rate, initialization time that is the time taken to allocate resource at datacenter also increases as the load increases. Fitness (FT), allocated MIPS (MP), Fault rate (FR), Initialization time (IT), Network load (N_L).

$$FT = \alpha_1 \frac{1}{N_L} + \alpha_2 \frac{1}{FR} + \alpha_3 \frac{1}{MP} \quad (4.20)$$

$$\alpha_1 < 1, \alpha_2 < 1 \text{ \& } \alpha_3 < 1 \quad (4.21)$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \quad (4.22)$$

Fault rate (FR) is:

$$FR(t) = f(MP, N_L) \quad (4.23)$$

Where $FR(t)$ is number of faults over the time t , which is function of system and network load over the time t . α_1 , α_2 and α_3 are constant which represents the ratio of parameters contribution to fitness value. Figure 4.43 is the pseudo code for proposed algorithm with all its steps.

Algorithm : Honey bee allocation

1. Honey_bee_allocation(Req_list r)

Input: Requests list r

2. Initialize scout bees equal to number of datacenters.
3. Recruit scout bees for selected sites (more bees for best e sites)
4. Evaluate fitness value for datacenter.
5. Assign scot bees to search randomly and evaluate their fitness for a request.
6. Stop when all bees have arrived, else wait.
7. Select the fittest bee
8. Assign remaining bees to search randomly and evaluate their fitness for each request.
9. End While no request in queue.

Output: All request been scheduled.

Figure 4.43 Proposed fault aware honey bee algorithm

4.4.2. Experiment and Result

Proposed fault aware honey bee algorithm is simulated using CloudSim 2.0 simulator [24]. CloudSim originally support Round robin, cost based algorithm, and FIFO algorithm for scheduling the resource sequentially. Originally CloudSim 2.0 API does not support faults in cloud environment. So firstly occurrence of fault is added as a parameter for datacenter which responds to failure probability of the datacenter.

This CloudSim API is used to set up a cloud infrastructure environment for simulation. So that environment includes all cloud IaaS request functions and environmental parameters, host and datacenter parameters. Proposed algorithm is implemented in cloudsims changing existing algorithm to study and improve the performance. Comparative study is made between basic load aware honey bee (BLHB) and proposed fault based load aware honey bee algorithm (FLBH). Figure 4.44 shows the improvement in number of requests failed over a

system with increasing request counts. Where we have considered 3 servers with 2 host each. Table 4.4 shows the failure rate of respective server.

Table 4.4: Server fault rate

Server Name	Fault rate FR(t)
Server1	0.143
Server2	0.125
Server3	0.5

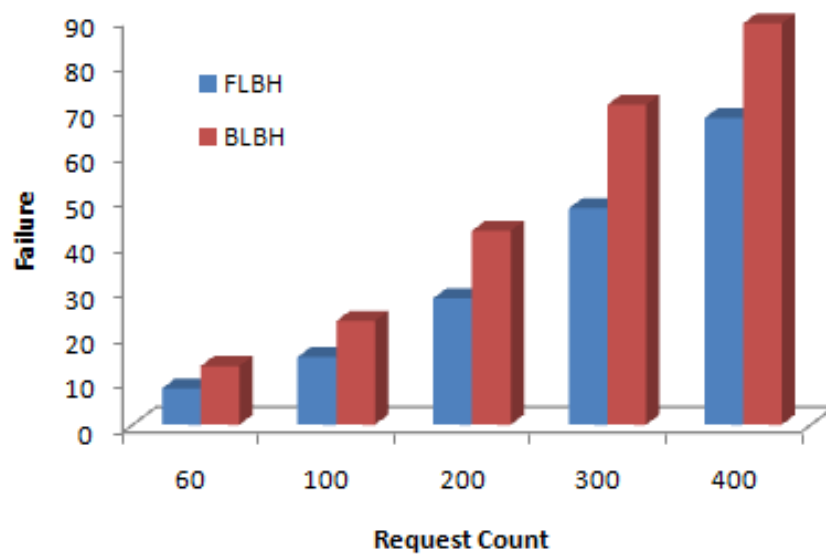


Figure 4.44 Comparison of request failure count.

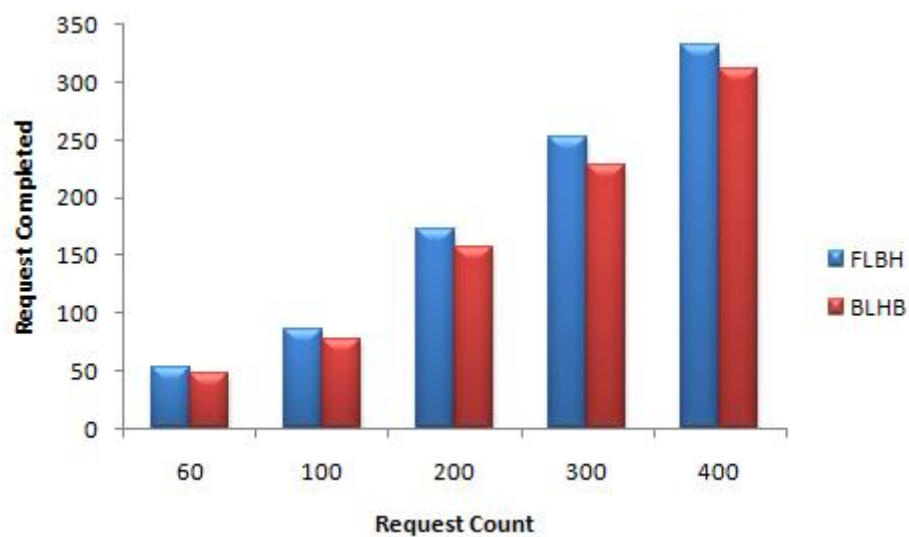


Figure 4.45 Comparison of request completed count.

Fig 4.44 shows the number of request failed using proposed and basic honey bee algorithm. Experiment shows that proposed algorithm have less number of request failures as compared to basic honey bee algorithm.

Fig 4.45 shows the number of request completed using proposed and basic honey bee algorithm, proposed algorithm proves to improve the request completion count as compared to existing algorithm. This graph shows the algorithm when tested with 60, 100, 200, 300, 400 requests. So the result shows the improvement of proposed algorithm over BLHB in fault aware environment

Table 4.5: Request failure count

	Request count				
	60	100	200	300	400
FLBH	8	15	28	48	68
BLHB	13	23	43	71	89

Table 4.6: Request completion count

	Request count				
	60	100	200	300	400
FLBH	52	85	172	252	332
BLHB	47	77	157	229	311

4.5. Comparative Analysis of Learning Based Algorithms

In this section we have performed a comparative study over all proposed learning based algorithm. The study is performed over various parameters like scheduling time, failed request count and request completed count using exiting genetic algorithm (GA) and proposed fault aware genetic algorithm, Big bang big crunch algorithm and fault aware big bang big crunch algorithm.

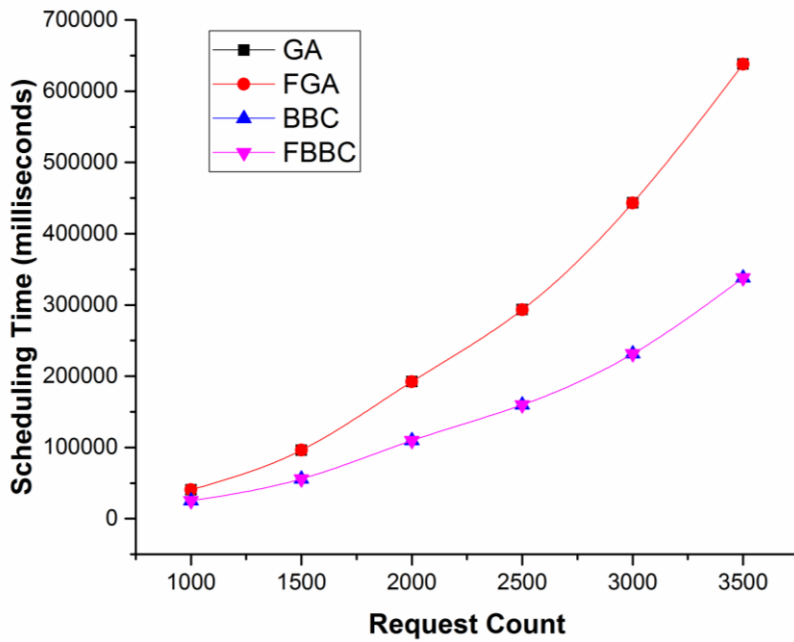


Figure 4.46 Comparison of scheduling delay

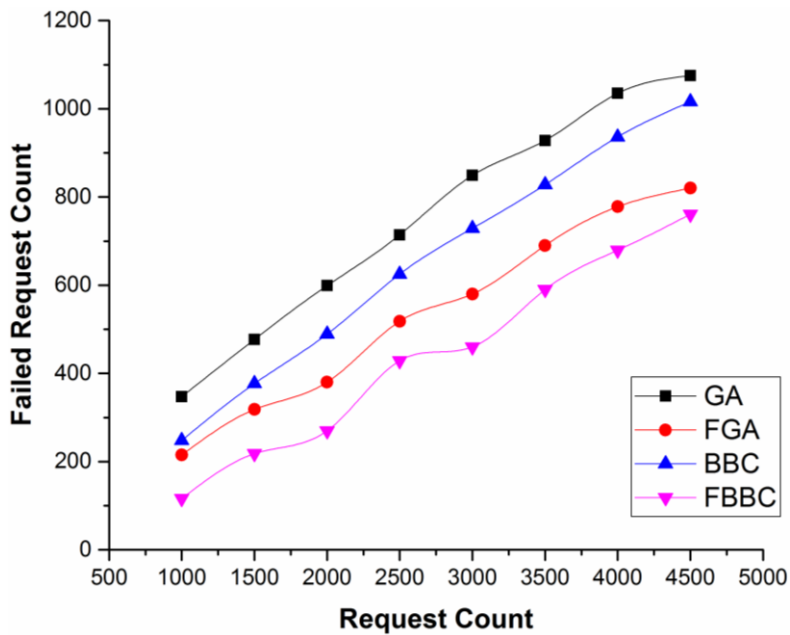


Figure 4.47 Comparison of failed request count

Figure 4.46 shows the comparison of scheduling delay by all four stated algorithms. Figure shows that Genetic algorithm, Fault aware GA takes same scheduling delay where as BBC and fault aware BBC takes almost same delay. BBC proves to be better in term of having least scheduling delay. Figure 4.47 compares the request failure count using all four algorithms with varying request count. Comparison shows the order of improvement in proposed algorithms which shows Fault aware BBC(FBBC) as best having least request

failures and the in the list is BBC, the is Fault aware GA(FGA). Existing GA proves to perform worst with highest request failure count.

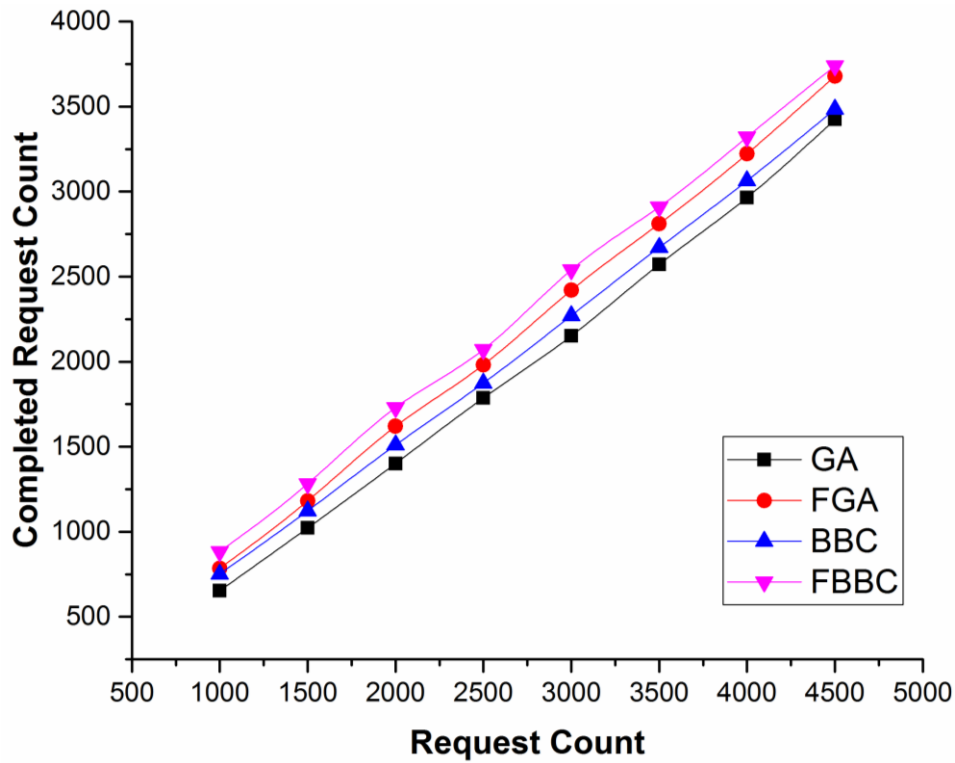


Figure 4.48 Comparison of completed request count

Figure 4.48 compares the performance in term of request completion count over all stated algorithms. FBBC proves to perform best with highest request completion count and at second number BBC proves to have better performance then FGA algorithm. GA algorithm proves to perform worst with least number of requests completed.

4.6. Conclusion

The main achievement of this work is to find the rich literature and solve the issue of resource allocation in fault aware cloud environment. The results obtained with our approach were very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed resource allocation algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, reduced average utilization, Scheduling delay, high request completion count, Failure probability and improved reliability of system.

CHAPTER 5

FAULT AWARE POWER EFFICIENT SCHEDULING

A Cloud computing is now a trending way of computing tasks and more general these days. Cloud computing is adopted by many firms like Google, Amazon, Microsoft and many more for reliable and efficient computing. But as the cloud size increases with expand in number of datacenters and vigorously increasing consumption of power over a data center. Also with increasing request load over a server the computing load on server's increases, leading high power consumption. So there is a need to balance the request load in such a manner to effectively improve the resource utilization, load with reducing request failure and power consumption.

Cloud computing has made it complicated with variable length request whose proportion may increase or decrease effecting the cloud. Recent surveys show that the power consumption of a datacenter increases linearly with increase in utilization due to request load over a datacenter. This results in high request failure and decreasing power efficiency of system. Resource allocation done without having knowledge of load and power efficiency of a datacenter will increase the power consumption of system and high request failure count. So to overcome these issues a fault and power aware resource allocation and scheduling algorithms are proposed to improve the power efficiency, failure count and average load over a datacenter. A proposed algorithm shows improved performance in term of average load and power efficiency as compared to existing algorithms for cloud infrastructure.

5.1. Approach 1: Power and Fault Aware Reliable Resource Allocation for Cloud Infrastructure

The problem with existing algorithm is that they used for simple task scheduling to improve resource utilization or power efficiency in cloud and manage quality of service of a datacenter. Existing algorithms also assumes cloud as non faulty in nature so do not takes fault occurring in the system for scheduling and only taken load over da datacenter, which is in sufficient to provide better QoS to the user. So to conquer these issues a power and fault aware resource allocation algorithm is proposed. Proposed algorithm utilizes linear power model or evaluation of power efficiency of datacenter. Failures over a datacenter occur randomly that may be due to network or storage failure. Proposed VM allocation algorithm aims to minimize the power consumption of the system and reduce request failure count. Proposed algorithm is based on fitness value which is evaluated using power efficiency and failure probability of datacenter.

Parameters to evaluate fitness value:

PD_i : i^{th} Data center.

PE_i : power efficiency of i^{th} host in a data center .

U_i : Current Utilization of i^{th} host in a data center.

FR_i : Fault rate that is the number of request failed due to System failure over time t.

FP_i : Failure probability over a Host i.

F_i : Fitness value of i^{th} host.

By Applying liner power utilization of PE_i can be calculated.

$$PE_i = LineaPower \left(\frac{(P_{max} - P_{min}) * U_i}{100} \right) \quad (5.1)$$

Where P_{max} & P_{min} = maximum and minimum power consumed by PD_i respectively.

Utilization of Data Center can be calculated by

$$U_i = \left(\frac{(Total_MIPS - Allocated_MIPS)}{Total_MPIS} \right) \quad (5.2)$$

Since faults over a data center are random in nature and follows Poisson distribution, which over a period of time t and $t + \Delta T$ can be defined as:

$$FPi(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(-\lambda(t + \Delta T))}{\exp(-\lambda t)} \quad (5.3)$$

$$FPi(t) = (1 - \exp(-\lambda \Delta t)) \quad (5.4)$$

$$\text{//Fitness value} \quad Fi = PEi + FPi(t) \quad (5.5)$$

As in above formula of U_i is calculated by getting total utilization from total MIPS allocated by data center PD_i . Once calculate utilization of data centers then calculating power consumed by these data centers and using linear power efficiency formula as above. To get power efficiency of data centers as well to allocation resources for requests is done with below steps. On the other end we need to calculate the fault rate over a data center PD_i , which depend on the number of request failed on a data center over a period of time 't'. Since fault is random in nature so the probability of failure can be found using poison distribution as shown in equation 5.3. Equation 5.3 defines the probability of failure at data center PD_i . Base on the above defined parameters fitness value of each datacenter is calculated, as shown in equation 5.5 which is sum of power efficiency and probability of failure in fraction which range from 0 to 1.

5.1.1. Proposed algorithm

Algorithm :PFARA

1. PFARA (PD, Q_{length})
- Input:** Datacenter List PD and Queue length Q_{length}
2. $PD \leftarrow$ Host List
3. $i \leftarrow$ No. of Data Centers
4. $Q_{length} \leftarrow$ current queue size
5. $PE_i \leftarrow$ Power Efficiency of Host Pd_i
6. $Fpi \leftarrow$ Failure Probability of Host Pd_i
7. If($Q_{length} \neq 0$) then
8. Allocate_Resources (Req)
- Output:** All request scheduled.

Figure 5.1 Proposed PFARA algorithm initialization

Algorithm : Find fittest host

```

1. Allocate_Resource( Req r)
Input: Requests list r
2. Host_list= gethostlist();
3. Sorted_host = Sort_Fitness (Host_list)
4. for (Host h: Sorted_host)
5.   Fi=PEi + FPi (t);
6.   If( h.isSuitable() && h.fitness_Least() )
7.     selected_host= h;
8.   end for
9. if( selected_host != NULL) then
10.  allocate( request, selected_host)
11. else
12.  printf("cannot find suitable server")
Output: The server with minimum fitness value.

```

Figure 5.2 Proposed PFARA algorithm resource allocation

Figure 5.1 & 5.2 shows the pseudo code of proposed trust and deadline aware ant colony algorithm. Figures show various phases of algorithm of initialization and evaluation of fitness value and final selection.

5.1.2. Experimental and Results

Proposed power and fault aware VM allocation algorithm is simulated using CloudSim 3.0 and power module package. CloudSim provides a benchmark for simulation of cloud platform and also provides Linear power model for simulation of power model. Proposed algorithm is being tested under various request count with 4 servers S1, S2, S3, S4 & S5. Linear power model directly depend on utilization of servers. Proposed algorithm is compared with basic DVFS (Dynamic voltage and frequency scaling) scheduling [80]. Compression of proposed algorithm is performed for 200, 400, 600, 800, 1200 and 1400 set of requests. These set of request contributes of various type of short, average and large requests sizes. System configuration taken into consideration is as follows:

Table 5.1: Experimental parameters used for simulation environment

Server	RAM(Mb)	MIPS	Storage (Gb)	core	PE	HOST
S1	2000	10000	100000	4	10	2
S2	2000	10000	100000	6	10	2
S3	2000	10000	100000	6	10	2
S4	2000	10000	100000	6	10	2
S5	2000	10000	100000	6	10	2

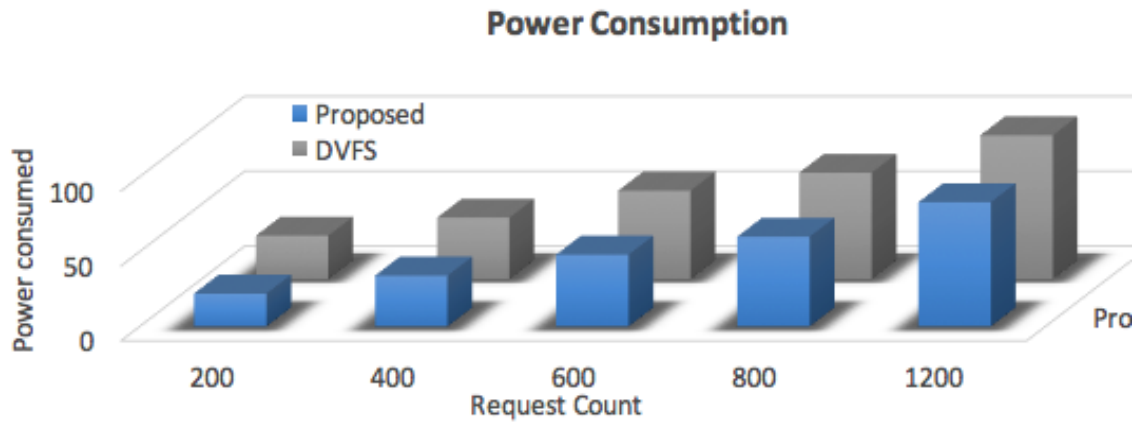


Figure 5.3 Power consumption

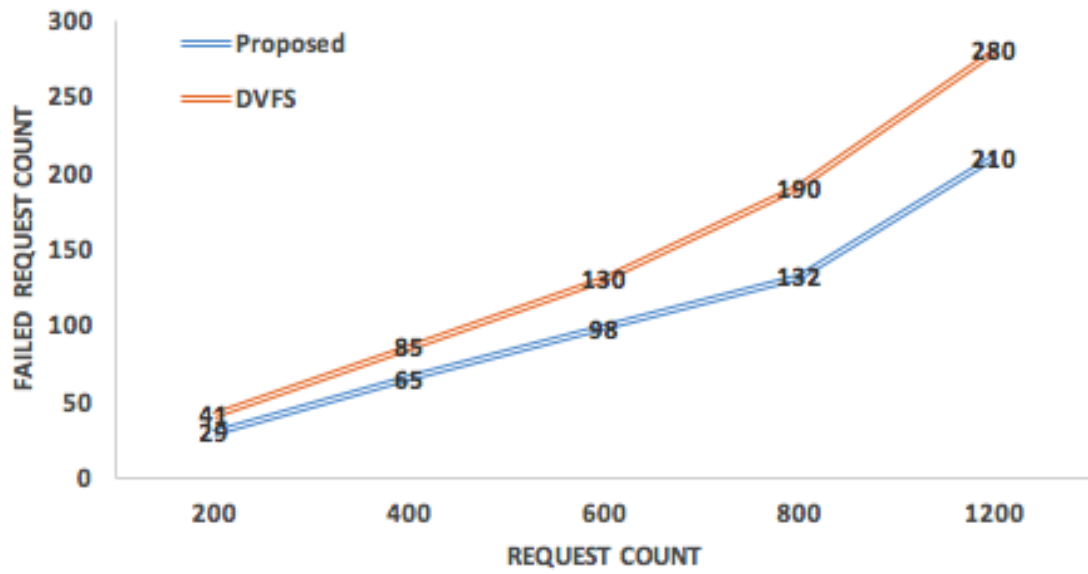


Figure 5.4 Comparison of request failure count

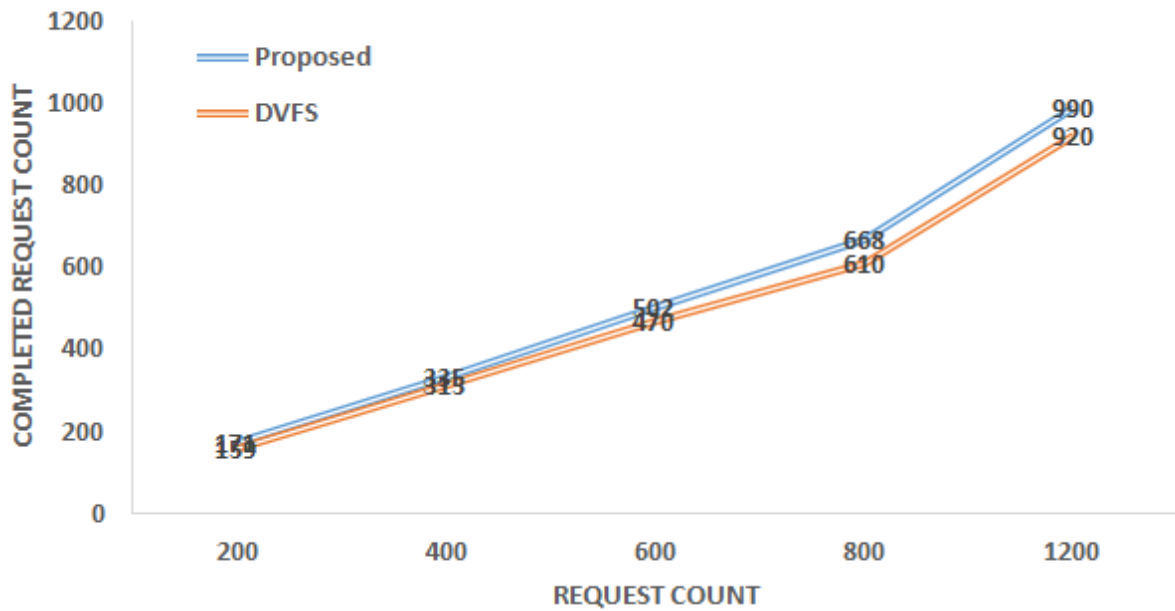


Figure 5.5 Comparison of request completed count

Figure 5.3 shows the improvement in power consumption by proposed algorithm over DVFS. Figure 5.4 shows the improvement in number of request failed by proposed algorithm over DVFS in various test cases. Figure 5.5 shows the improvement in number of request completed by proposed algorithm over DVFS in various test cases. From the experiment it is shown that proposed algorithm performs better than DVFS in term of failed request , power efficiency and completed request count.

5.2. Approach 2: Trust and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization

This paper proposed a trust and deadline aware algorithm that uses various parameters to evaluate the trust value for a host, on that trust value we have proposed and VM allocation policy to maximize the utilization of resources available in data center. Flow chart of proposed algorithm is below. As can see in flow chart we begin with task pool, here we look for task, if task pool is empty then do nothing but if there is some request in task pool for completion then we proceed to collect the information of each data center. Trust can be defined as an indirect reliability or a firm believe over a host based on its past performance parameters.

Trust is based on:

Start time: Time taken by host to initialize a virtual machine (VM).

Processing Speed: Total number of MIPS of a machine i.e. Number of processor * number of MIPS in each processor

Fault Rate: This can be defined as the total count of request failed over a period of time T

Utilization: this is the current utilization of that host in real time.

Power Efficiency: the ratio of the output power over the input power i.e. the percentage power consumed over a period of time.

For scheduling algorithm, we have proposed an ant colony based VM allocation algorithm which uses a fitness function based on above discussed trust value and deadline to find the fittest host among all.

Steps for proposed algorithm are as follows:

Step 1: Initialize datacenters and host

Step 2: Initialize search ants equal to number of hosts.

Step 3: Assign ants to search randomly and evaluate their fit nesses for a request on each host.

Step 4: Stop when all ants have arrived, otherwise wait for all ants for a fixed time.

Step 5: evaluate the trust value for each host and sort them in descending order.

Step 6: find the fittest host with highest trust value and can fulfill the task with deadline.

Step 7: if found, update pheromone value table with updated trust value that will be used for evaluation of fitness function for other requests.

Step 8: Assign bees to search randomly and evaluate their fitness and find new best solutions.

Step 9: Stop, when no more requests.

Trust value (T_i): Trust value for host i .

$$T_i = \alpha_1 * Initial_i + \alpha_2 * PS_i + \alpha_3 * \frac{1}{Fault_i} + \alpha_4 * \frac{1}{Utilization_i} + \alpha_5 * \frac{1}{PE_i} \quad (5.6)$$

$$\alpha_1 < 1, \alpha_2 < 1, \alpha_3 < 1, \alpha_4 < 1, \alpha_5 < 1 \quad (5.7)$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \quad (5.8)$$

where

$Initial_i$: initialization time of host i .

$$PS_i : PE_i * MIPS_i \quad (5.9)$$

$Fault_i$: fault rate over host i .

$Utilization_i$: utilization of host i at current point of time.

PE_i : Power efficiency of host i .

Fitness function $F(n)$:

$$F(n) = \min(T_i) \text{ \& } D_j < \text{computation time}_i \quad i=0 \dots n \quad (5.10)$$

j is request id and computation time is the time to compute the request over host ' i '. Let take PE_i (power efficiency) and U_i (Utilization) of data Centers($i, e; i=1,2,3,4, \dots, n$). By Applying linear power utilization of PE_i can be calculated.

$$PE_i = \text{LineaPower} \left(\frac{(P_{\max} - P_{\min}) * U_i}{100} \right) \quad (5.11)$$

Where P_{\max} & P_{\min} = maximum and minimum power consumed by PD_i respectively.
 U_i =Utilization of Data Center can be calculated by

$$U_i = \left(\frac{(\text{Total_MIPS} - \text{Allocated_MIPS})}{\text{Total_MIPS}} \right) \quad (5.12)$$

To get power efficiency of each data center first calculating utilization of PD_i then using Power Liner model to calculate power efficiency of that data center. The proposed algorithms Pseudo is below, this Pseudo code shows that request allocation based on power efficiency of data center minimize power loss and increase utilization of resource that implies, throughput of data center is increasing.

Pseudo code of TDARP (Trust and Deadline Aware Resource Allocation Policy) algorithm takes data centers list, queue length of task in task pool, Power Efficiency of data centers, as shown in pseudo code if task pool is not empty, then calculate the power efficiency on the basis of their utilization.

5.2.1. Proposed Algorithm

Algorithm :TDARPA

1. TDARP (Host List PD and Q_{length})
- Input:** Host List PD and Queue length Q_{length}
2. $PD \leftarrow$ Host List
3. $i \leftarrow$ No. of Data Centers
4. $Q_{\text{length}} \leftarrow$ current queue size
5. $PE_i \leftarrow$ Power Efficiency of Host PD_i
6. $F_{pi} \leftarrow$ Failure Probability of Host PD_i
7. $\text{Initial}_i \leftarrow$ Start time of Host PD_i
8. If($Q_{\text{length}} \neq 0$)
9. Sent_Search_ant(Reqtype);
10. find Fittest host(Reqtype);
11. Update pheromone();
12. $Q_{\text{length}} --$;
13. If($Q_{\text{length}} > 0$) then goto step 1;
14. End
- Output:** The server with minimum fitness value.

Figure 5.6. Proposed TDARPA algorithm (1)

Algorithm : Find fittest host

1. Find_Fittest host(Reqtype r)
- Input:** Request type r
2. Compute PE_i for each host
3. Compute utilization for each host
4. Compute T_i for each host
5. Sort_descending(T_i)
6. Find host with high trust value and fit's deadline
7. Return selected host

Output: The server with minimum fitness value.

Figure 5.7. Proposed TDARPA algorithm (2)

Figure 5.6 & 5.7 shows the pseudo code of proposed trust and deadline aware ant colony algorithm. Figures show various phases of algorithm of initialization and evaluation of fitness value and final selection.

5.2.2. Experiment and Results

Proposed power based trust and deadline aware allocation algorithm is simulated using CloudSim 3.0 and power module package. Linear power model is used for simulation of power model. Proposed algorithm is being tested under various request count with 4 servers S1, S2, S3 & S4. Linear power model directly depend on utilization of servers. Proposed algorithm is compared with basic DVFS (Dynamic voltage and frequency scaling) scheduling [80]. Compression of proposed algorithm is performed for 1000, 1500, 2000, 2500 and 3000 set of requests. System configuration taken into consideration is as follows:

Table 5.2: Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	core	PE	HOST
S1	3000	2000	100000	4	10	4
S2	3000	2000	100000	6	10	4
S3	3000	2000	100000	4	10	4
S4	3000	2000	100000	4	10	4

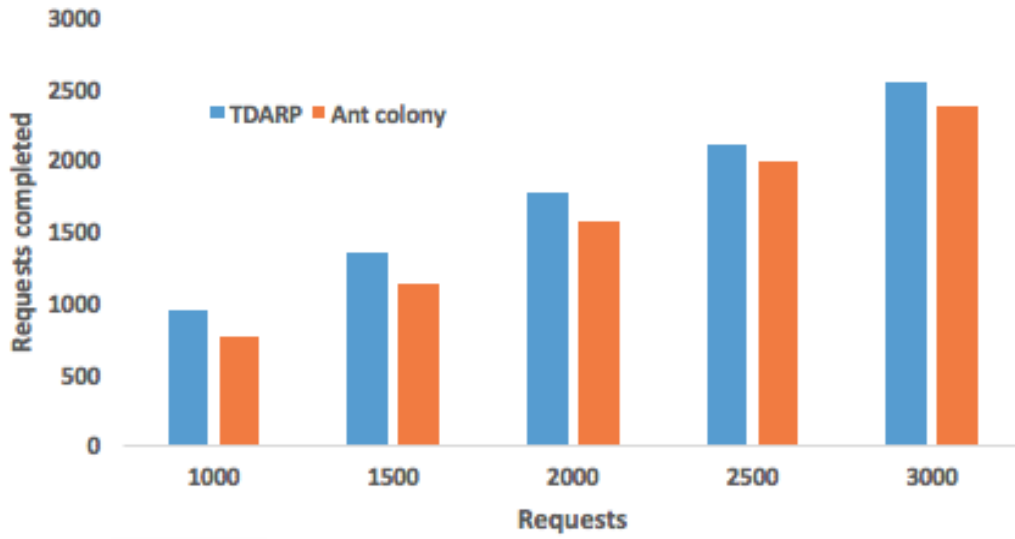


Figure 5.8 Comparison of request completed

Figure 5.8 and 5.9 shows the improvement in number of request complited and failed using proposed and ant colony based base algoithm [81]. Figure also proves that the proposed algorithm completes more requests than existing algorithm . Figure 5.9 shows the improvement in power comsumption when tested over 1000, 1500, 2000, 2500 and 3000 set of requests. figure 5.10 compares the power efficiency of proposed algorithm over exeisting algorithm over increasing request load. Proposed algorithm proves to consume less power as compared to existing algorithm.



Figure 5.9 Comparison of request failed



Figure 5.10 Comparison of power consumed in Kwh

In performance section, it is clear that TDARP is giving high performance as compare to previous proposed algorithm. The main of this algorithm in cloud computing is to complete the request as possible as minimum power and full utilization of resource, proposed algorithm shown that it can maximize throughput and minimize the requests failure count and computation power.

5.3. Conclusion

The main achievement of this study and work was to study the rich literature and solve the issue of resource allocation in fault aware cloud environment. The results obtained with our approach were very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed resource allocation algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, high request completion count, and Power consumption of system.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1. Conclusion

The work puts its eye to achieve fast, optimal and fault tolerant algorithm for load and resource allocation for cloud. Novel approaches to improve the performance of system in variant load and cloud environment have been presented. The issues of resource optimization with high resource utilization are optimized with proposed load balancing techniques. Proposed algorithm also optimizes the scheduling time to allocate all resource with high resource utilization by promising global best schedule with least failure probability. Finally, as depicted by the experiments, proposed approaches are proven to be superior as compared to existing algorithm. The efforts are made to design algorithms in such a way to improve over system performance rather than improving a specific parameter like cost, deadline or power. Proposed multi objective resource allocation algorithm reduces the request failure count in fault aware environment and also resource utilization at the same time and improves the reliability of the system in efficient way.

Proposed work for fault ware load balancing algorithm for storage cloud reduces the deadline failure over the system and improves the reliability of the system, reduces the waiting time of request and reducing the average queue length of each server. Average utilization of system has improved in worst case scenario with high request rate. In an fault aware storage cloud proposed system has better overall response time as compared to least loaded algorithm. The power efficient learning based algorithms improves the reliability of system and increasing power efficiency of system as compared to DVFS (Dynamic voltage and frequency scaling) algorithm. The proposed algorithms for hybrid cloud provided better QoS for private and public requests in hybrid cloud environment, makes its reach to different business models in cloud.

The overall probability of request failure has decreased, improving the reliability of system that may be private public or hybrid cloud environment. The research work also takes into consideration improvement done by static, dynamic and learning based algorithms for resource allocation and load balancing for reliable and energy efficient system for Green Computing.

6.2. Future Scope

The advantages of fault aware and reliable algorithm has improved the performance of cloud environment, this can further be improved by incorporating fault tolerant feature to other algorithms like migration and replica management algorithm. This may improve the performance of the system up to a great extend and shall increase the reliability of the system.

The proposed solutions for resource allocation and load balancing can further be improved by introducing further parameter like cost and scalability to improve the system performance. The proposed solutions are restricted to private cloud and can be extended to public or hybrid cloud. The techniques of fault aware learning based algorithm can be used to find better solution in replica management, scalable algorithms, migration algorithms and many more.

LIST OF PUBLICATIONS

Journal Publications

- [1] Gupta P, Ghrera SP. Load Balancing Algorithm for Hybrid Cloud IaaS. International Journal of Applied Engineering Research (IJAER). 2015 10:69 :257-261. [*SCOPUS, EBSCOhost, GOOGLE Scholar*] (*SJR: 0.13*) (*Cite 1*)
- [2] Gupta P, Ghrera SP. Fault and Load Aware Load Balancing in Cloud Storage. International Journal of Applied Engineering Research (IJAER). 2015 10:69 :280-285. [*SCOPUS, EBSCOhost, GOOGLE Scholar*] (*SJR: 0.13*) (*Cite 1*)
- [3] Gupta P, Ghrera SP. Deadline-Aware Load Balancing of Distributed Servers in Distributed. International Journal of Applied Engineering Research (IJAER). 2015 10:69 :268-273[*SCOPUS, EBSCOhost, GOOGLE Scholar*] (*SJR: 0.13*)
- [4] Gupta P, Ghrera SP. Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure. Indian Journal of Science and Technology (IJST). [*SCOPUS, ISI Thomson Reuters, DOAJ, Copernicus Poland*] (*SJR: 1.3*)(*Accepted*)
- [5] Gupta P, Ghrera SP. Fault Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure. Indian Journal of Science and Technology (IJST). [*SCOPUS, ISI Thomson Reuters, DOAJ, Copernicus Poland*] (*SJR: 1.3*)(*Accepted*)
- [6] Gupta P, Ghrera SP. Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure. International Journal of Advanced Intelligence Paradigms,(Accepted).[*SCOPUS, ACM Digital Library, DBLP, GOOGLE Scholar*] (*SJR: 0.16*)
- [7] Gupta P, Ghrera SP. QoS Aware Grey Wolf Optimization For Task Allocation In Cloud Infrastructure. . International Journal of Advanced Intelligence Paradigms, (Communicated).[*SCOPUS, DBLP, GOOGLE Scholar*] (*SJR: 0.12*)

Conference Proceedings

- [8] Gupta P, Ghrera SP. Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure. InProceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014 2015 :135-143. Springer International Publishing. [*SCOPUS, ISI Thomson Reuter, DBLP*](*Cite 3*)

- [9] Gupta P, Ghrera SP. Power and Fault Aware Reliable Resource Allocation for Cloud Infrastructure. *Procedia Computer Science*. 2016 Dec 31;78:457-63.[*SCOPUS, Proceedings Citation Index, GOOGLE Scholar*] (*Impact: 0.705*)
- [10] Gupta P, Ghrera SP. Trust and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS) 2016 Feb.[*SCOPUS*]