HANDLING LOAD BALANCING IN CLOUD STORAGE

Project Report submitted in partial fulfillment of the requirement for the degree of

Master of Technology

in

Computer Science & Engineering

under the Supervision of

Dr. Pradeep Kumar Gupta
(Supervisor)
&
Mr. Punit Gupta
(Co-supervisor)

By

Ravideep Singh (132209)

Jaypee University of Information Technology
Waknaghat, Solan – 173234, Himachal Pradesh
May 2015

Certificate

This is to certify that project report entitled **"HANDLING LOAD BALANCING IN CLOUD STORAGE"**, submitted by **"Ravideep Singh"** in partial fulfillment for the award of degree of Master of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Sign______________________

Co-Supervisor: Mr. Punit Gupta

Assistant Professor (Grade-I)

Supervisor: Dr. Pradeep Kumar Gupta

Assistant Professor (Senior Grade)

Acknowledgement

First and foremost, I would like to express my deep sense of gratitude to my supervisor **Dr. Pradeep Kumar Gupta** and Co-supervisor **Mr. Punit Gupta** for providing excellent guidance during my research and study at Jaypee University of Information Technology, Waknaghat, Solan(H.P). Their perpetual energy, motivation, enthusiasm and immense knowledge inspired me to discipline myself in efficiently executing my multiple responsibilities simultaneously. In addition, they were always accessible and willing to help me with my queries and doubts during my research.

I would like to thank my supervisor **Dr. Pradeep Kumar Gupta** as well as my co - supervisor **Mr. Punit Gupta**, who let me, experience the research of role of Load balancing in Distributed environment, patiently corrected my writing and financially supported my research. With their guidance, I was able to collect the building data and proceed for my report.

I wish to express my gratitude and high regards to **Prof. Dr. Satya Prakash Ghrera** head of CSE department, JUIT.

I would ike to express my profound sense of gratitude to all the faculty members for their valuable suggestion and encouragement throughout my course.

I would also like to thank my family and my friends especially my colleague **Mr. Ajeet Singh** for their valuable suggestions and undue support which has been a constant source of encouragement in all educational pursuits.

Date: Ravideep Singh

Table of Contents

List of Figures

List of Tables

List of Abbreviations

QoS	Quality of service
MSP	Managed service provider
API	Application programming interface
VSA	Virtual server assignment
VST	Virtual server transfer
DFS	Distributed file system
VM	Virtual machine
OLB	Opportunistic load balancing
LBMM	Load balancing min-min
SLA	Service level agreement
ABC	Artificial bee colony
LBACO	Load balancing ant colony
FCFS	First come first serve
ACO	Ant colony
DHT	Distributed hash table
CDLBA	Capability based distributed load balancing algorithm
DDLBA	Deadline based distributed load balancing algorithm

Abstract

From few last decades, there is a huge proliferation of data in cyberspace. In order to manage data efficiently, distributed storage plays the important role. Cloud storage is one kind of the distributed storage provided by cloud computing technology. Cloud storage acts as a repository in which data is stored, managed and made accessible to end users. Largest generated application datasets can flexibly be stored or deleted in the cloud and end users can access this data using cloud storage services interface, without accessing any storage server in real. Cloud storage system is comprised of hundreds of independent storage servers which are distributed geographically and are sufficient to handle thousands of client requests concurrently. In the system few of the storage server gets huge clients requests where as other servers remain idle or least loaded. This unequal distribution of load on storage servers leads to degrade the performance of overall system and increases the response time of submitted requests. In this work, we have addressesed these issues for efficient utilization of storage servers in cloud storage. Handling various challenges related to the load balancing in the cloud storage is the one of the main objectives of this research work. We have proposed two distributed load balancing algorithms Capability based distributed load balancing algorithm (CDLBA) and Deadline based distributed load balancing algorithm (DDLBA) by exploiting the different parameter of storage server. Here, first algorithm considers the service rate, and queue length as a main parameter of the server where as the second algorithm considers parameters like service time, and deadline time of the client request. The main objective of this research is to monitor various aspects which leverage the overall performance of cloud storage. Proposed algorithms are sufficient to balance the load of storage servers and effectively utilize the server's capabilities. From the obtained simulation results, we can say that our proposed algorithms balance the load efficiently utilize the server capabilities, reduce the response time, and leverage the overall system performance.

CHAPTER 1
INTRODUCTION

Rapid growth of Internet Technologies has increased the data proliferation exponentially on the network. Cloud computing is one of the technology that provides cloud storage to manage the data. Cloud storage act as a repository in which the data is maintained, managed and is made available to the end users [1]. Large generated application datasets can flexibly be stored or deleted in the cloud and from here end users access this data by using cloud storage services interface, without accessing any storage server in real. Cloud storage system is considered of hundreds of independent storage servers which are distributed geographically [2], and thousands of client requests are handled by these storage servers concurrently. These storage servers get huge clients requests and some servers remain under loaded. Due to this unequal distribution of load on storage servers leads to degrade the performance of the overall system and increases the response time. Resources are not utilized adequately as some server gets too many requests and some remain idle. In cloud storage system, load can either be in term of requests handled by a server or storage capacity of that server or both.

In this work, we have proposed the load balancing approach to balance the load in terms of requests of overloaded servers in the cloud storage.

1.1. Cloud Computing

Cloud computing is a new style of web based computing model for providing flexible, cost-effective and on demand network access to a common pool of configurable computing capabilities that can be quickly allocated and managed with negligible administrative effort or with least interaction to cloud service provider [3]. It is a pond of manageable, virtual and highly extensible computing infrastructure which enables the customers to host their applications over internet in cost effective manner [4].

Cloud computing is emerging as the most recent disseminated computing paradigm which gives excess, reasonable and adaptable resources on request to client over the internet. This technology effectively exploits the sharable resources on internet such as memory, storage, computation power and bandwidth. Cloud service delivery is divided into three models [5]. The three service models are:

- Software as a Service (SaaS)
- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS).

Cloud computing model can be categories into three deployment models:

- Public Cloud
- Private Cloud
- Hybrid Cloud

1.1.1. Layered Architecture of Cloud Computing

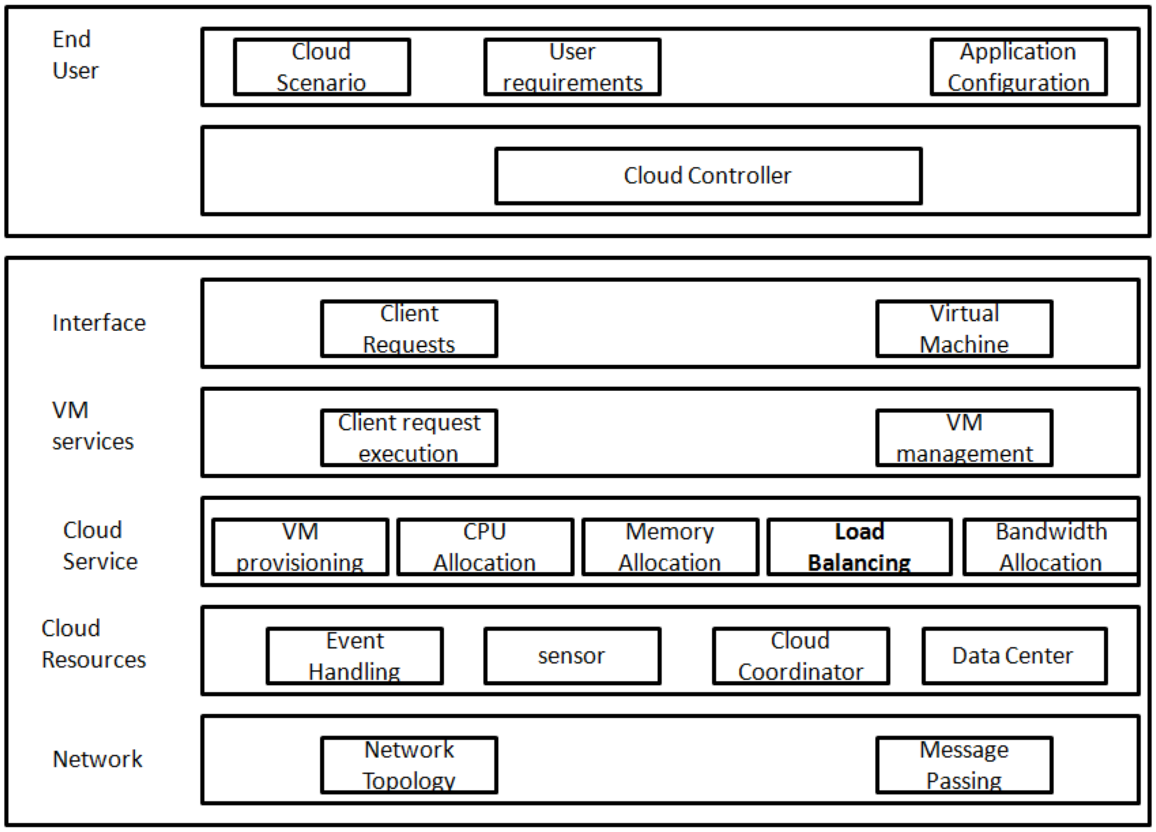


Figure 1.1: Layered Architecture of Cloud Computing

Here figure 1.1 shows the layered architecture of cloud computing. In figure 1.1, load balancing is one of the cloud services in cloud computing.

1.1.2. Core services of Cloud Computing [6]

a) *Discovery*

Cloud computing advances the reusability through discovering the various existing services. Through this service, service provider organizes the cloud infrastructure in a cost effective way.

b) *Replication*

Replication can be utilized to make and keep up duplicates of an organization's data at different locations. At the point when occasions influencing an organization's primary site happen, primary application services can viably be resumed and operate at the other site where replica is stored at negligible cost.

c) *Load Balancing*

Load balancing leverages the system performance by avoiding the bottleneck in the system due to uneven load distribution. Load balancing increases the system performance by reducing the request response time and increase the throughput of system. It also provides continuation of services through fault tolerance when one or more components of system get failed. Through this mechanism, various instances of applications can be allotted and de-allotted automatically by a load balancer without altering the network configuration. It also enhances the life of infrastructure by putting less stress on the hardware portion of each component. Scalability is another feature provided by load balancing.

d) *Resource Management*

In real time, cloud environment delivers an efficient route to using enormously scalable, sharable assets on request at moderate cost. All sorts of homogeneous as well as heterogeneous resource environments can be effectively handled by Resource management services of cloud. Scheduling of available resources and tasks, administration virtualized resources, provisioning of cloud resources with guaranteed QoS, administration of extensible resources are the concerning focuses. Dynamically scheduling the resources over a virtualized framework for such environment is another challenging task in the cloud.

1.2 Cloud Storage

Storage of data over the Internet is one of the essential application of cloud computing. Cloud storage is capable of providing the users to store the enterprise data in the different storage servers of different vendors instead of storing the data in particular storage server. Cloud storage implements the location transparency, so that user can never know where his data stored in the cloud storage but it provides the abstract view of local storage. Cloud storage is simply an alias used to pointing out to virtual storage in the cloud environment. Hence in cloud storage, client's data can be accumulated on one or many of the systems that participate in the cloud environment. But the real repository area may significantly vary from time to time or even moment to moment, as the cloud powerfully oversees accessible storage areas. Anyway, despite the fact that the storage place is virtual, the client gets a static view of his data area and can trivially work with his cloud storage which physically resides far away from the client [7].

Economically, in comparison to dedicated physical resource, virtual resources are less expensive. Concerning security, enterprise data put away in the cloud storage is safe from unintentional eradication or equipment failures, on the grounds that it is replicated over numerous physical devices. Because various replicas of the information are kept ceaselessly, the cloud storage keeps on working as typical regardless of the possibility that one or more machines get disconnected from the net. [7].

1.2.1. Evolution of Cloud storage

Since there is neither a clearly characterized arrangement of abilities nor any standard for architectural planning, so cloud storage is nebulous still now. Decisions proliferate, with numerous customary facilitated i.e. managed service provider (MSP) providing file or block storage, using conventional remote access methods [8].

Cloud storage is a transformation of the online storage technology that coats many namespaces, management tools, files virtualization, and complex APIs, across storage. There is several diverse cloud storage offered by various cloud vendors. Depend on the market requirements some have a particular purpose, for example, archiving Web email or digital images. Others are accessible to store all types of digital data. Cloud storage frameworks are divided according to their computation such as some of them are little operations, while others are huge to the point that the physical hardware can top off a whole warehouse [8].Basically cloud storage system requires at least one data storage server associated with web. A user transmits duplicates of files over the Web to the data storage server, which then stores the data. User access the data through web browser [8].

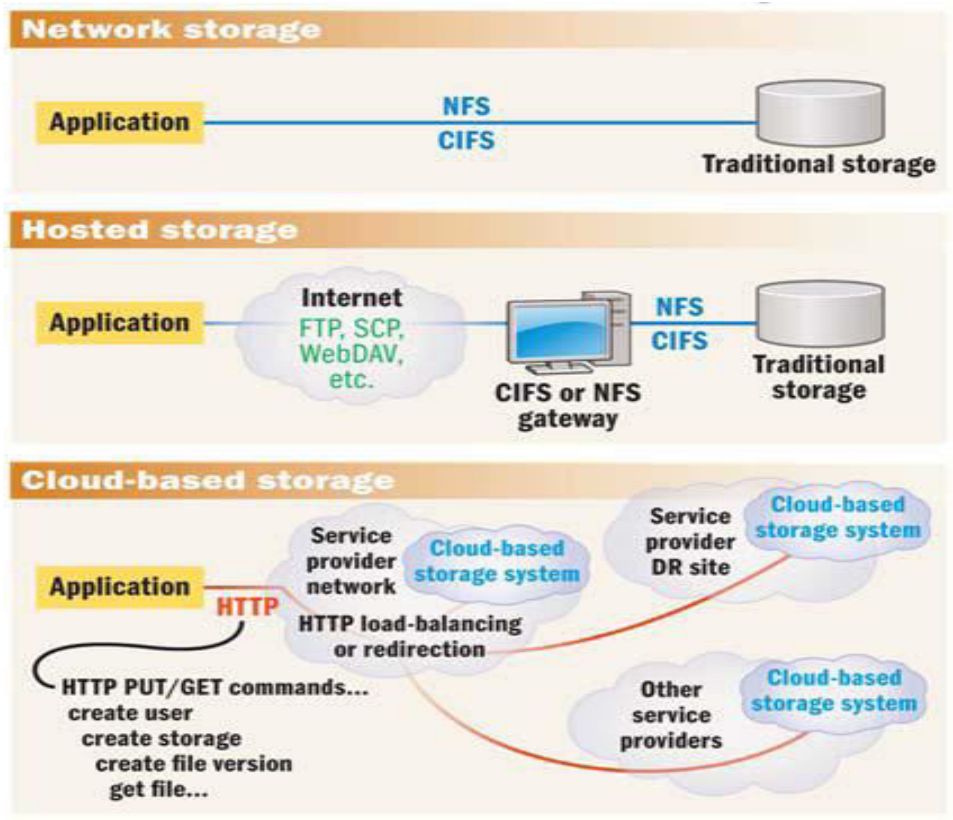


Figure 1.2: Evolution of Cloud storage [8]

Based on conventional network storage and hosted storage, Figure 1.2 shows the evolution of cloud Storage.

1.2.2. Architecture of Cloud Storage

Here, figure 1.3 shows the architecture of cloud storage system where central control server controls the several storage servers. Storage servers are connected with central control server and other storage server. Client interacts with cloud storage system through central control server which in turn decides about storage server to handle the client requests.

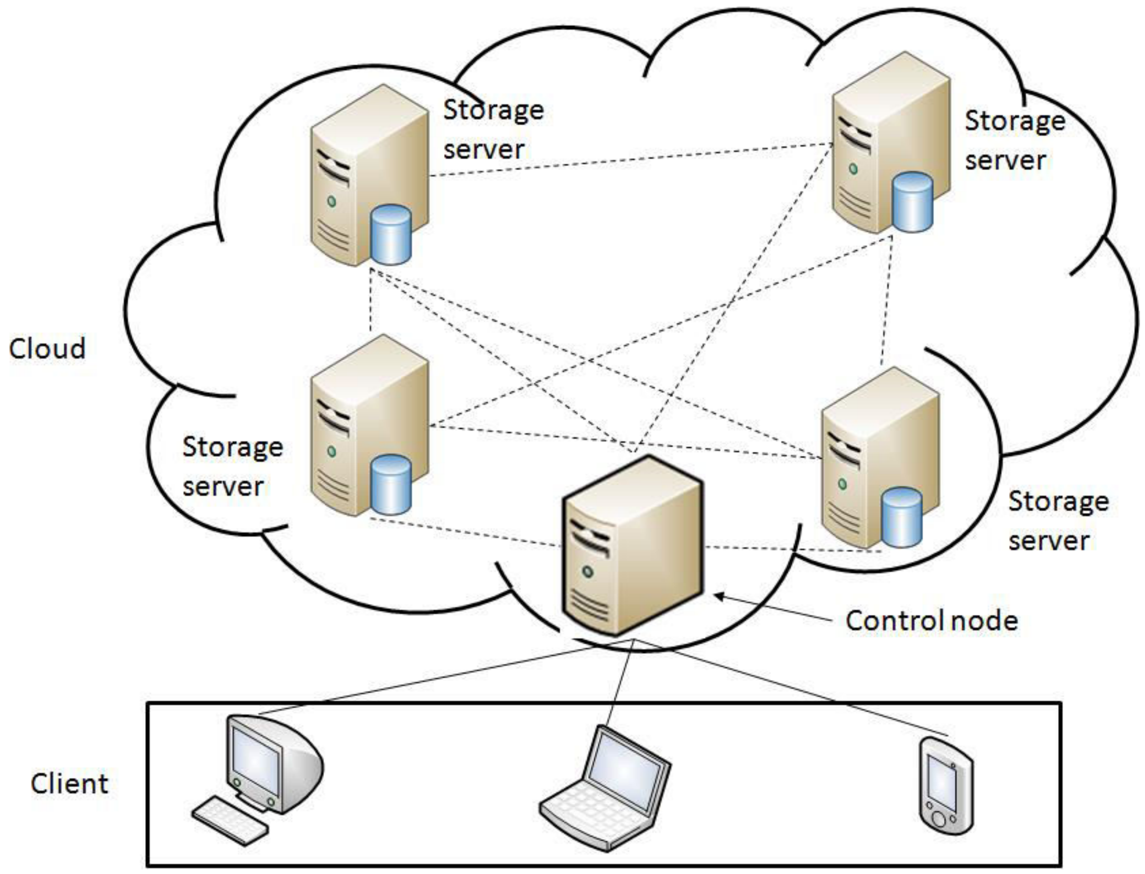


Figure 1.3: Architecture of cloud storage system [7]

1.2.3. Issues in cloud storage

Though cloud storage is getting the consideration of IT managers because of its relatively low cost and capacity to effectively change limit offers reduction in the capital venture cost, still clients has to face issues at different levels [9].

a) *Control over the Data*

   Since the information is dwelling outside the enterprise’s infrastructure, it is seen that the enterprise might loss the control over data. In spite of the fact that the concerns are highly psychological and hypothetical than real, due to the immaturity of cloud services, benchmarks on the conveyance of services and their advancing plan of action, clients may have authentic concerns about the service provider’s practicality and operational procedure.

b) *Interoperability & Control*

   The unpredictability of utilizing cloud storage is be something numerous clients underestimate "It's not plug-and-play." Each vendor has distinctive access methods, nonstandard APIs that make incorporating applications, for example, storing or sharing in cloud storage are complex and expensive. The absence of standard protocols for using the cloud storage implies there will be no interoperability between cloud storage providers, incredibly confusing the data migration.

c) *Performance*

   Access to cloud information is clearly constrained by system throughput and latency, and in spite of intense enhancement in Internet performance, it is still poor in correlation to local storage. Since some vendors try to upgrade throughput with various local caching strategies and compression strategies, these don't enhance Internet latency.

d) *Security*

Data security is also one of the main issues in cloud storage. It is because clients store their data on remote location which is highly susceptible and insecure. If there is any possibility leakage, both in exchange and inside a shared infrastructure, specialist concur that utilizing encryption on all data in cloud storage

1.3 Load Balancing in cloud storage

Load balancing is a strategy to disseminate amount of work across two or more computing resources such as servers, network devices, etc in order to get more work at same amount of time. Load adjusting is utilized to verify that none of current resources are idle while others are being used [10]. The target of load balancing is to build the throughput by using most extreme resources accessible in the system. In cloud datacenters, typical storage server architecture depends on huge, robust, powerful computing hardware and network framework. They all are under the considerable risks associated with physical devices including hardware failure, power failure, network failure/congestions, and resource limitation during high demand [10].

Load balancing plays a key role in the cloud storage. It improves the overall performance by balancing the workload over the entire distributed storage nodes in cloud. For cloud computing applications, load imbalanced scenario occurs frequently even though the workload was distributed evenly before. In term of cloud storage load balancing signify even distribution of workload as well as efficient utilization of all the storage nodes. In cloud computing environment, storage servers are geographically distributed across the globe. Load balancing algorithm in the cloud environment differs from classical view of load-balancing architecture and implementation by employing commodity servers to achieve the load balancing. There are four basic steps that are common in nearly all load balancing algorithms [11].

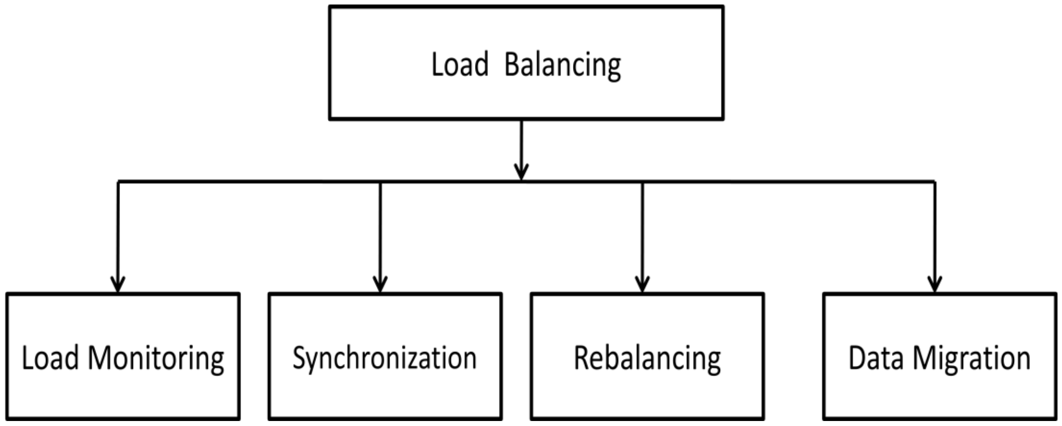


Figure 1.4: Four basic steps in Load balancing [11]

- Load Monitoring –Observing the load and state of resources.
- Synchronization –Interchanging load and state information between resources.
- Rebalancing Criteria -Compute the new task distribution and executing balancing decisions.
- Data Migration –Actual migration of load among resources.

1.3.1. Types of Load balancing algorithms

Different types of load balancing algorithm have been proposed depending on their nature of implementation. Load balancing algorithms can be categorized into three categories as follow [12][13]:

- *Sender Initiated:* In this algorithm, sender triggers the load balancing mechanism.
- *Receiver Initiated:* In this algorithm, receiver triggers the load balancing mechanism.
- *Symmetric:* In this algorithm, both sender and receiver can triggered the load balancing mechanism.

Relying on the available condition of the system, balancing of load can be categorized into two categories as follow [12]:

- *Static:* In this algorithm, prior knowledge of system in required instead of current state of system. In static load balancing when a new server connects to the system, it tries to find an intensely stacked server and assume control over a portion of the heap from that server and when a server exit from the system, it hunt for a least loaded server to pass its current load on that server.

- *Dynamic:* In dynamic load balancing, balancer operates when a server who is currently present within the system becomes overloaded or least loaded. In dynamic load balancing, methodology work when servers that have effectively joined the framework get to be over-burden or least loaded. The overloaded server searches for a lightly loaded server to balance their load and the under loaded server searches for a heavily loaded server to balance their load.

In distributed system, dynamic load balancing algorithm is of two types:

a) *Distributed:* In this balancing algorithm, balancing of load is carries out by all servers available in the distributed system and load balancing job is distributed amid them. In order to achieve load balancing, the collaboration among servers can be taken in two structures: cooperative and non-cooperative [12].

- In the cooperative structure, the servers perform task parallel to accomplish a common goal, e.g. to enhance the general reaction time.
- In the non-cooperative strucutre, each server performs tasks autonomously toward a local goal, e.g. to enhance the general reaction time of a local job.

Due to the distributed nature of dynamic load balancing algorithms, communication overhead in distributed dynamic load balancing are generally more than the non-distributed dynamic load balancing algorithms. But it provides more fault tolerance because if one of the serverss failed, it won't bring about the aggregate load balancing procedure to end. But it would affect the overall system performance to some extent.

b) ***Non distributed:*** Non distributed dynamic load balancing algorithm performs occupation of load adjusting on either one server or set of servers. Non distributed one further divided into two types:

- *Centralized:* The load balancing is completed just by a solitary server in the entire system i.e. the master server. This server is totally in charge of load adjusting of the entire system. Alternate servers correspond just with the master server.

- *Semi distributed:* In this type, servers in the system are classified into clusters, where each cluster performs own load balancing in centralized type. A central node is voted in each cluster by relevant election process which performs the load balancing within that cluster. Hence load balancing of whole system is executed via the central node of each cluster.

1.3.1.1. Polices in dynamic load balancing

In dynamic load balancing algorithm, there are four policies as shown in figure 1.5 [12]

Transfer Policy: This module picks a task for migrating from a local node to a remote neighbor node.

Selection Policy: It defines the nodes involved in the load migration.

Location policy: This module picks a destination node for load migration.

Information Policy: This module responsible for gathering status information regarding nodes in the system.

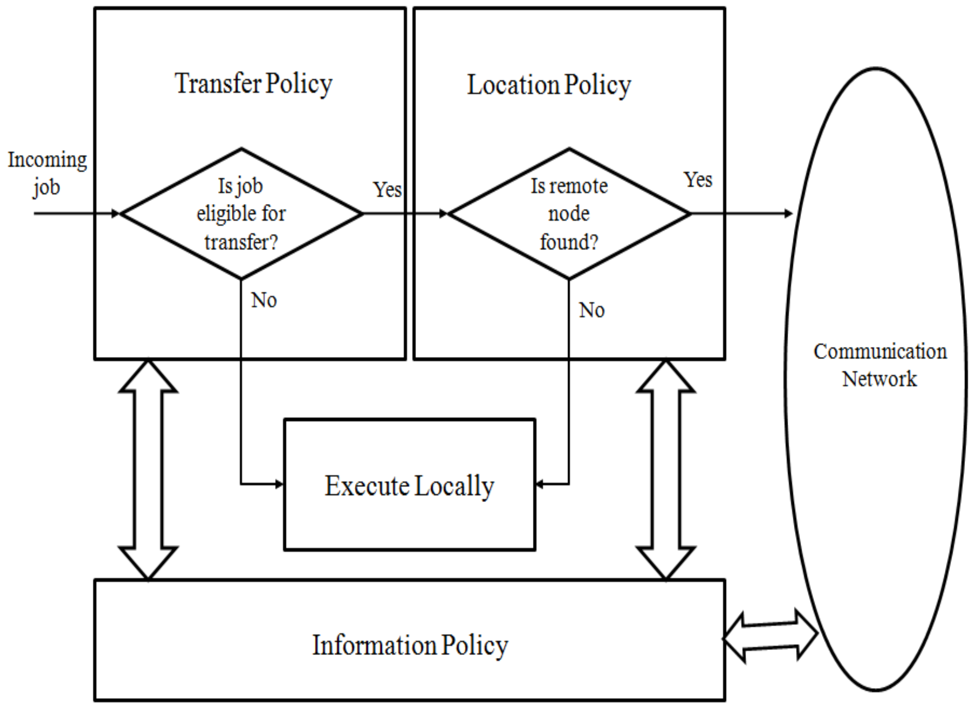


Figure 1.5: Interactions of components in Dynamic load balancing algorithm [12]

1.3.2. Metrics for Load Balancing

Various important matrices used for load balancing algorithms are discussed as follow: [11]

- ***Throughput*** is utilized to ascertain the number of job whose processing has been accomplished. It ought to be maximized to enhance the execution of the system.
- ***Overhead Associated*** decides the measure of cost of actualizing a load balancing technique in term of time. It is made out of extra cost because of migration of jobs, between various process interactions and processors. This ought to be least so a load adjusting strategy can perform effectively.
- ***Fault Tolerance*****:** The capacity of an algorithm to perform uniform load balancing despite subjective nod or connection failure. The load balancer ought to be a fault tolerant strategy.
- ***Migration time*****:** The total duration to move the tasks or processes among the nodes available in the system. It ought to be least in order to upgrade the execution of the system.
- ***Response Time*****:** The measure of time elapsed to react by a specific load adjusting mechanism in a disseminated system. It ought to be least.
- ***Resource Utilization*****:** It is utilized to analyse the use of resources. It ought to be advanced for an effective burden adjusting.
- ***Scalability:*** It is the capacity of an algorithm to operate load balancing efficiently the size of system increased or decreased. It ought to be enhanced.
- ***Performance:*** It is utilized to check the effectiveness of the system. This must be enhanced at a sensible expense, e.g. minimize tasks response time while keeping worthy deferrals.

1.4. Problem Statement

As we can see that cloud storage provides remote storage services to subscribed clients through Internet. These storage services enable the users to use storage space remotely. As the user sends a request for online storage space, it is redirected to the nearest storage server that queue received request and process them as server gets idle. But in real, handling of these user requests is different as some servers are get overloaded with huge client requests, and some remains idle. Due to this reason, overloaded servers become potential spots to enhance the overall performance of the system. This problem can be illustrated clearly through Figure 5. 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 that how much number of requests processed by a server simultaneously. Initially at time t=0, we assume that each server receives an approximately equal amount of requests. In the above figure, we took 8 requests to illustrate the scenario. In the second case after time t=2, each server processed the client according to its service rate S_r. As shown in the Figure 5, server S1 request is empty, and they become idle. At the same time, server S3, S5 are fully loaded. In this situation fully loaded, server takes time to process all requests while other servers are idle. In cloud storage, many times server are not utilized 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 are required to migrate to idle server or under loaded server. There is a need of a mechanism that can adequately transfer client requests to these available servers and another problem is that servers have a limit to store the incoming requests.

So when the input buffer of server gets that overloaded then the server start discarding the client requests. This will lead to poor performance of the system as well as poor response time. Our aim is to avoid such situations and efficiently utilizes the capability of each server in the network. We have proposed a mechanism, which can evenly distribute the client requests among various servers to leverage the system performance by reducing the response time.

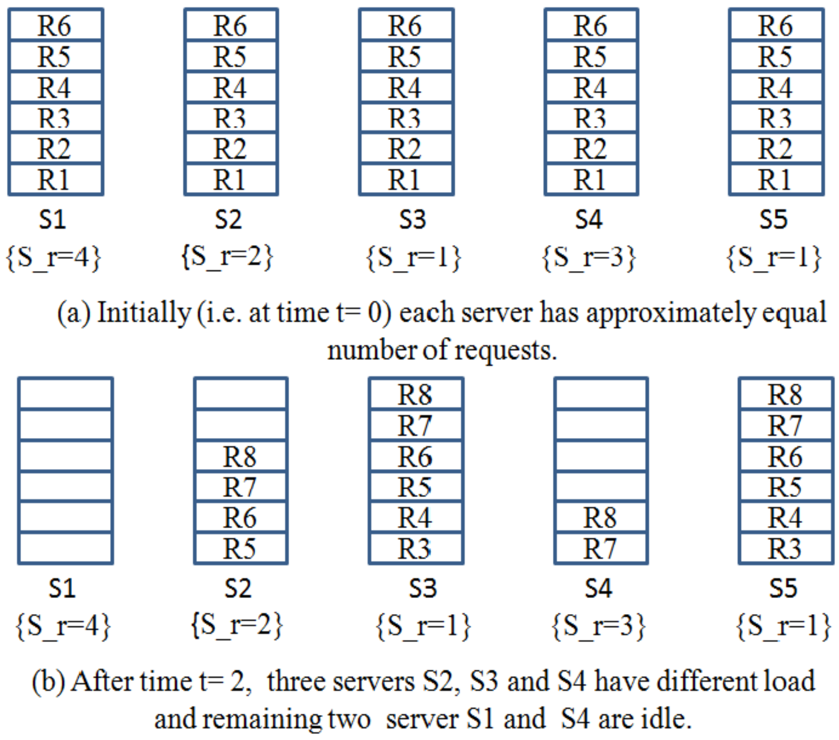

R6
R5
R4
R3
R2
R1
S1
{S_r=4}
S2
{S_r=2}
S3
{S_r=1}
S4
{S_r=3}
S5
{S_r=1}
(a) Initially (i.e. at time t= 0) each server has approximately equal number of requests.
R8
R7
(b) After time t= 2, three servers S2, S3 and S4 have different load and remaining two server S1 and S4 are idle.

Figure 1.6: Load balancing problem in cloud storage

1.5. Objective

- Designed a distributed load balancing algorithm for cloud storage.
- Reduce the waiting time of client requests in server queue for processing.
- Enhance the utilization of server.
- Reduce the overall response time of system.

1.6. Organization of report

This report is organized into five chapters. Chapter 1 describes what is cloud computing, various core services provided by cloud computing, cloud storage, evolution of cloud storage and its architecture, various issues in cloud storage, load balancing in cloud storage, types of load balancing algorithms in cloud storage, problem statement and objective of report. Chapter 2 describes about the previous research work related to the proposed problem statement. Chapter 3 describes about the proposed work, which consists of assumptions, proposed system and proposed algorithms. Chapter 4 describes about the simulation environment and results. Chapter 5 describes the conclusion of the report and future work.

CHAPTER 2
LITERATURE REVIEW

The following sections describe the literature background for the proposed problem statement given in chapter 1. Here, various authors had proposed their approaches to solve the various issues related to proposed problem statement related to our problem statement till now. We have categories the literature into two different sections: Centralized load balancing algorithms and Distributed load balancing algorithms.

2.1 Centralized load balancing algorithms:

Zhu et al. [14] have proposed an efficient, proximity-aware load balancing algorithm by introducing the concept of virtual servers. They have proposed the concept of proximity relation between various servers for load balancing. Their contributions are three fold: 1) they use fully distributed, self-organized, k-ary tree structure developed on top of a DHT. So load balancing is attained by aligning the two skews, load distribution and node capacity which are inherent in P2P systems, that is, the node which have higher capacity carry more loads, 2) Minimizing the cost of load movement to perform load balancing efficiently. This can be done with the help of proximity information which is used to guide virtual server reassignments. So that reassignment and transportation of virtual servers occurred between geographically close lightly loaded nodes and heavily loaded node and 3) Obtained simulation results shows that the proposed algorithm reduces the load migration cost by 11-65% for all the federation of load dissemination to virtual servers, node capacity profiles, and representative network topologies.

Load balancing Scheme consists of four phases as shown in below figure 2.1 proposed by Zhu et al. [14].

Load balancing information (LBI) aggregation: Accumulate load as well as capacity information about complete system.

Node classification: Distinguished which node overloaded, which node is under-loaded and which one is neutral based on their load and capacity.

Virtual server assignment (VSA): This phase is an important phase because in this load balancer identify from which overloaded server virtual server is migrated to under-loaded server. In this phase proximity information is used.

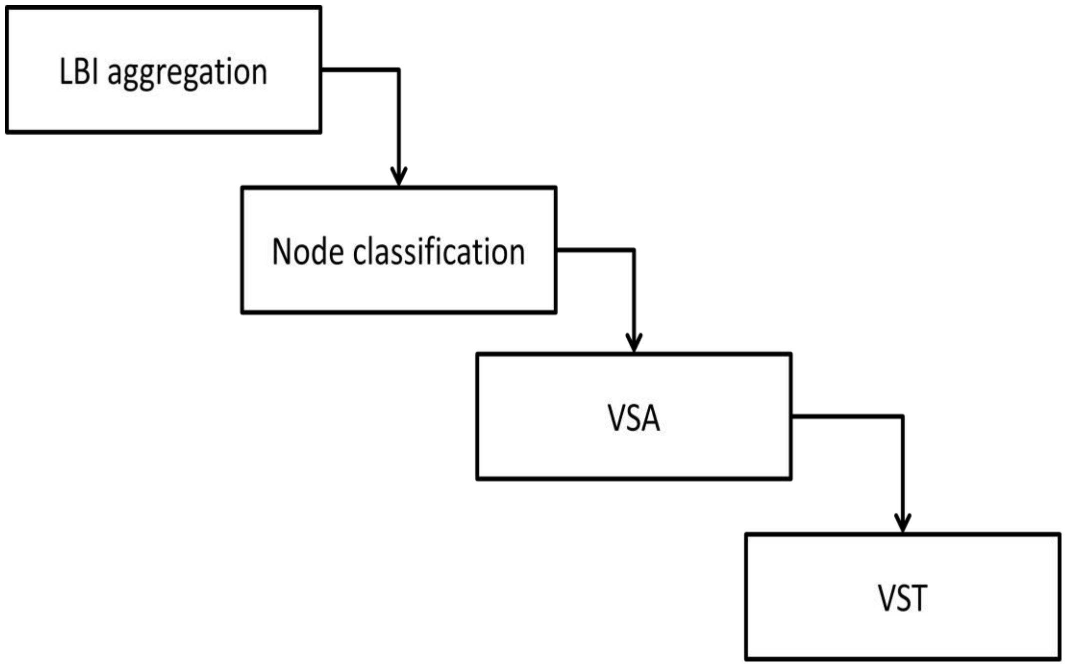


Figure 2.1: Phases in load balancing

Virtual server transfer (VST): In this phase, virtual server migrated from overloaded server to under-loaded server.

Zeng et al. [15] have purposed a load rebalancing algorithm in the distributed file system in order to fix the load balancing issues between various chunks of servers. Authors have also focused on the reliability and fault tolerance by ensuring that one chunk of a file and its replicas are stored in 3 different chunk servers simultaneously. Here, authors have proposed that large scale distributed file system formulated in a tree like structure and root of the tree act as a global namespace define a group of files. Each namespace can accommodate more than one file, and each file exactly belongs to only one namespace.

Each file is divided into fixed size chunks, and each chunk has two replicas. Figure 2.2 shows the organization of files in large scale distributed file system.

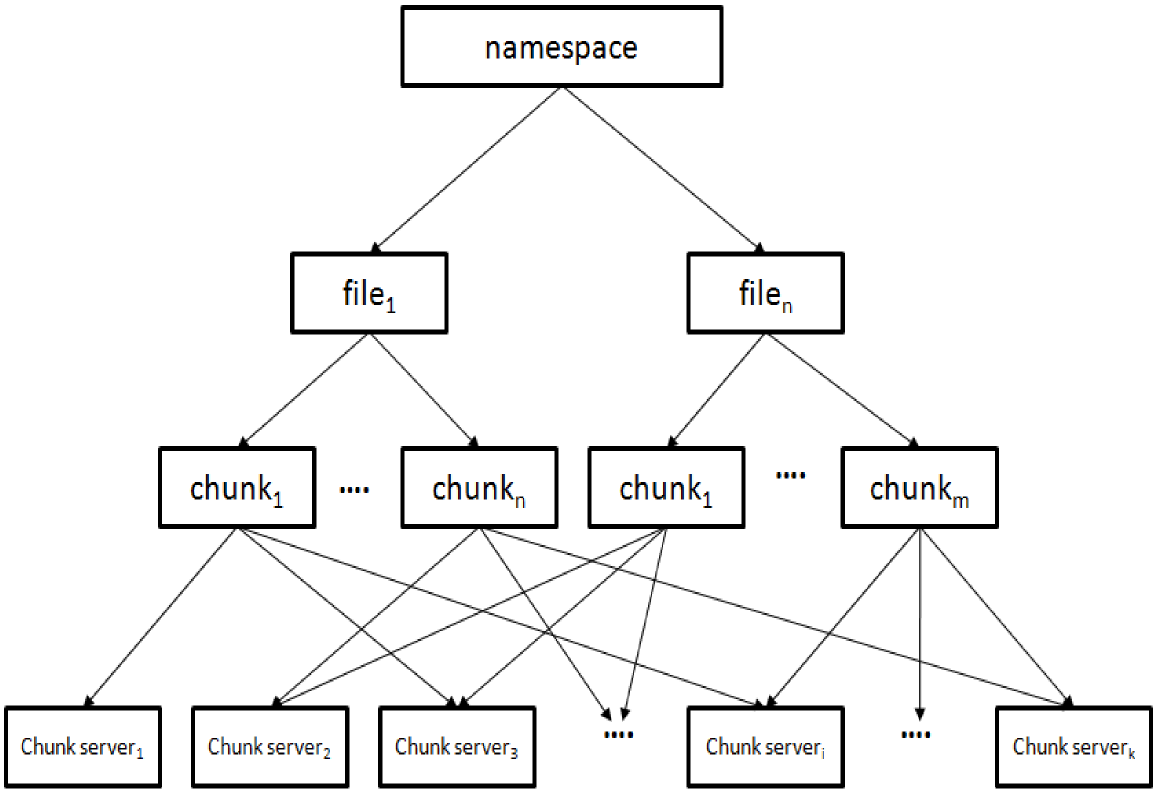


Figure 2.2: File organizational structure of large scale DFS [15]

As shown in Figure 2.3 rebalancing model of large scale distributed file system, master server act as a coordinator node that periodically checks the load of each chunk server. If load distribution unbalancing occurs, master server performs the load rebalancing task. Authors have calculated the load of each chunk server based on the following: average bandwidth utilization, average CPU utilization, average disk utilization, and chunk capacity in the chunk server. In this paper, Authors defines two load thresholds value: high and low.

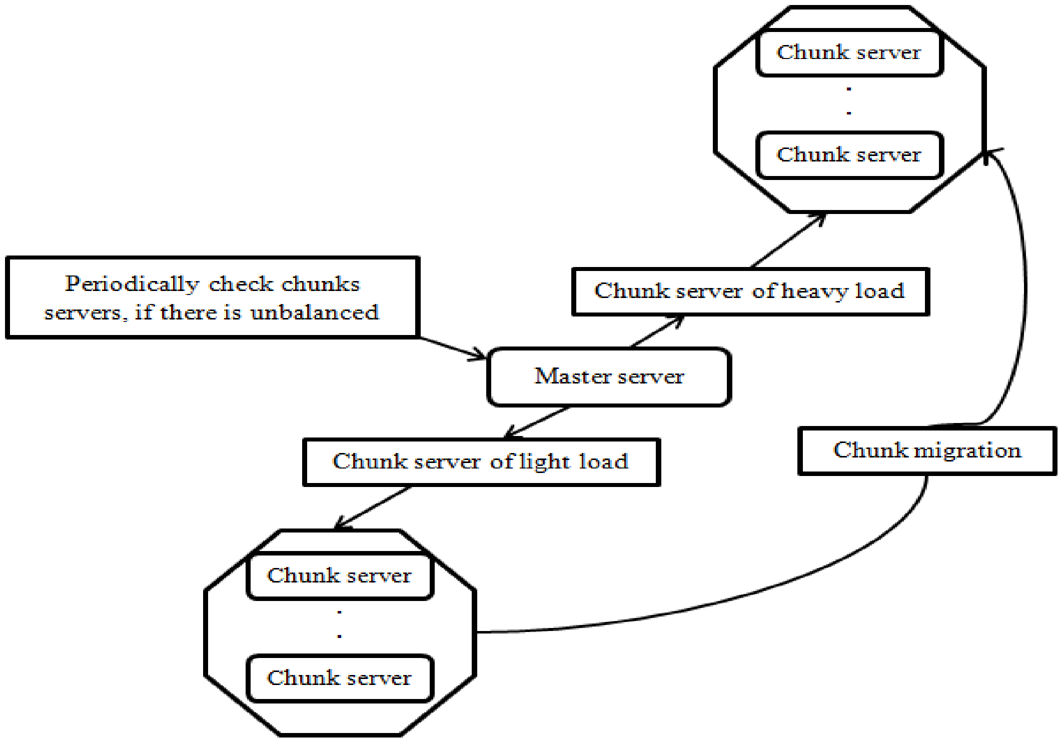

Chunk server
Chunk server
Periodically check chunks servers, if there is unbalanced
Chunk server of heavy load
Master server
Chunk server of light load
Chunk migration
Chunk server
Chunk server

Figure 2.3: Rebalancing model of large scale DFS [15]

A chunk server whose load above the high can be considered as heavily loaded chunk servers and below the threshold considered as lightly loaded chunk servers. Authors have concluded that load rebalancing is one of the powerful techniques to improve the overall performance of the system.

Hu et al. [16] have proposed a randomized load balancing algorithm to leverage the utilization of cloud resource through virtualization. Authors have used genetic algorithm to efficiently utilize the virtual machine which deployed on the physical server. They have focused on minimizing the migration cost. Proposed approach is centralized in nature. Authors have mainly focused on how efficiently mapping the various virtual machines among the available physical servers and reducing the migration cost of virtual machines.

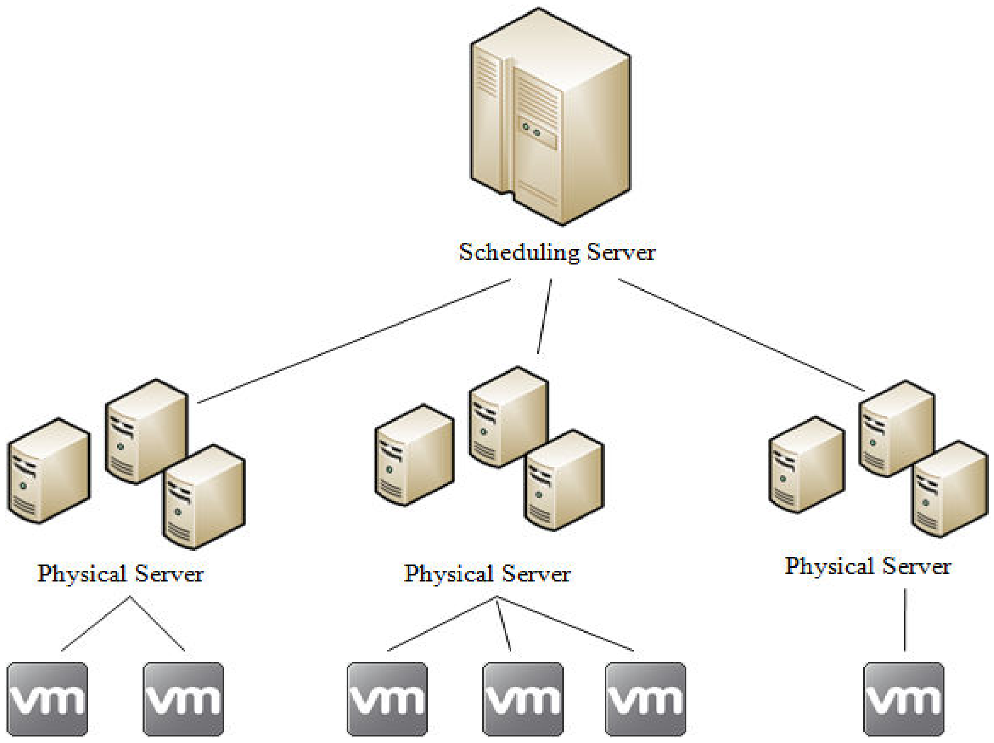


Figure 2.4: System structure [16]

Figure 2.4 depicts the system structure of proposed model of Hu et al. Here authors have described the mapping of virtual machine with physical servers in abstract manner. Figure 2.4 clearly shows that scheduler server maintains the global information of all physical servers and their respective VMs. Scheduler server perform load balancing. Finally, Authors have compared their results with least loaded and rotating algorithm in term of load and migration cost.

Wang et al. [17] have proposed a load balancing algorithm under the three levels in cloud computing environment. Authors have integrated OLB (Opportunistic Load Balancing) and LBMM (Load Balance Min-Min) scheduling algorithms to propose their algorithm to leverage utilization of executing efficiency and balance the load.

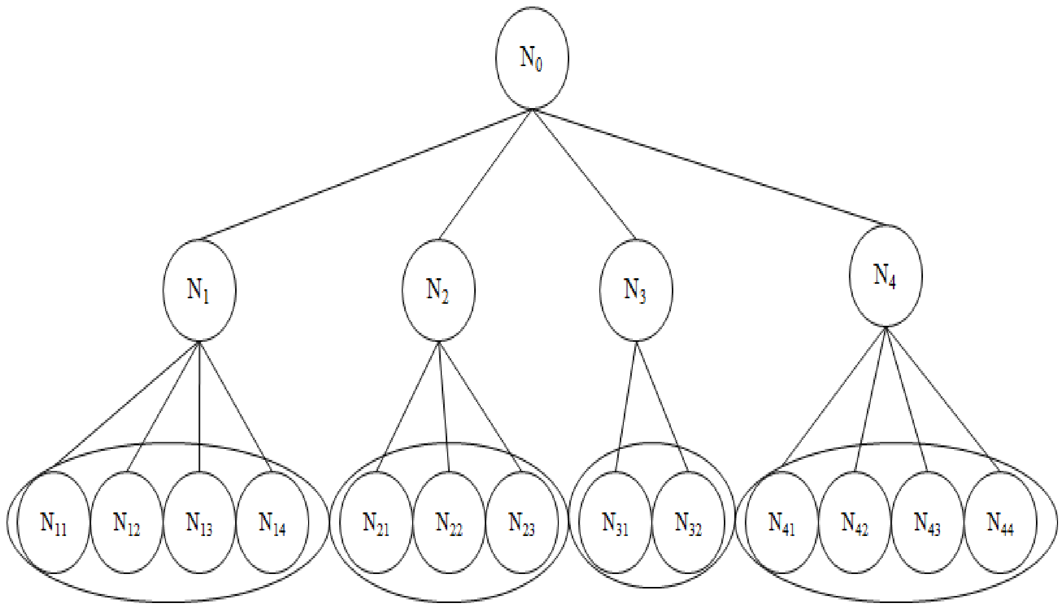


Figure 2.5: Three-level framework [17]

Figure 2.5 shows three-level hierarchical framework that used in the proposed approach. The three levels in the figure are described as follows:

- Lowest level represents the service node to process subtasks.
- Center level represents the service manager to partitions the jobs into free subtasks.
- And top most level represents request manager to map the task with appropriate service manager.

LBMM scheduling technique is utilized to allocate job to every service manager in the form of some subtask. These subtasks are executed in respective service node. Finally, algorithm balances the load by keeping the minimum execution time for task.

Tian et al. [18] have proposed a compelling and incorporated resource scheduling algorithm (DAIRS) for Cloud datacenters. DAIRS integrates network bandwidth, CPU and memory for both virtual servers and physical servers. They have devised an integrated measurement for mean asymmetry level of each server as well as overall asymmetry level of a Cloud datacenter. Integrated load balance measurement can be computed as follow:

$$V = \frac{1}{(1 - CPU_u)(1 - MEM_u)(1 - NET_u)}$$

Where CPU_u, MEM_u, NET_u are average utilization of CPU, memory and network bandwidth respectively.

Authors have considered the following resources:

- *Physical server:* physical processing machines which construct a datacenter. Each one can have multiple VMs, CPUs, memory devices, network devices, etc.
- *Physical clusters*: number of physical servers, network devices and storage device organized into a group called cluster.
- *Virtual servers:* it is a virtual processing component running on a physical server through virtualization software.

Scheduling of tasks in Cloud datacenter is shown in the following Figure 8 depicts a referred architecture of Cloud datacenters and key operations of scheduling:

- User requests: user sends requests.
- Scheduling management: decision taken by the scheduler based on user request type.
- Feedback: response provided by resource scheduler algorithm to the users.
- Executing scheduling: scheduling results are pipelined to next stage.
- Updating and optimization: updating the resource information and optimize the scheduling process.

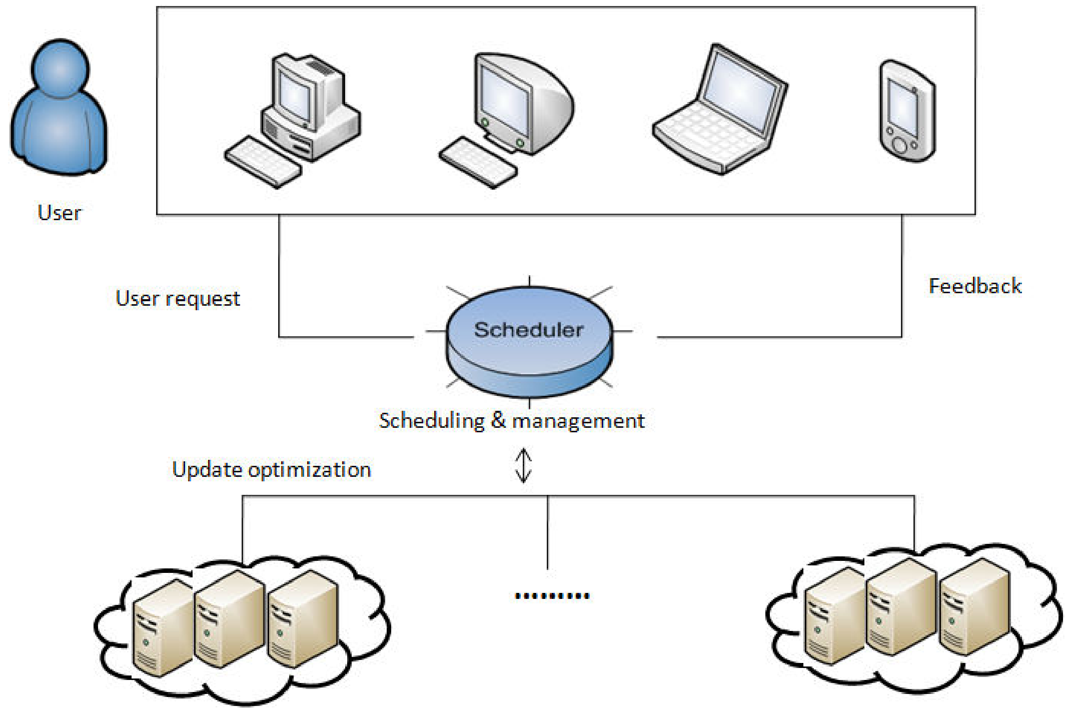


Figure 2.6: referred architecture of Cloud datacenters and key operations of scheduling [18]

Lee et al. [19] have proposed feasible resource aware load balancing algorithms to leverage SLA by using existing technology. Authors have proposed two load balancing models. These model schedule workload based on latest available resource utilization by dynamically comparing them in each server. Authors have assumed that two models of servers namely Model-H and Model-L, have different resource capacity. Model-H represents higher capacities whereas Model-L represents lower capacity. LB_n load balancer gathers client requests regularly. Using random policy, authors have formulated the probability of Model-H as follow:

$$f(LB_H) = \frac{C(N_H, LB_H) * C(N_L, LB_L)}{C(N_H + N_L, LB_H + LB_L)}$$

$$LB_n = LB_H + LB_{HN}$$

$$C(n,r) = \frac{n!}{(n-r)! * r!}$$

Where N_H represents higher capability of Model-H, N_L represents low capability of Model-N, LB_H and LB_L are load balancer of respective model. Authors have proposed two new resource aware techniques: 1) Resource best and 2) Resource fit.

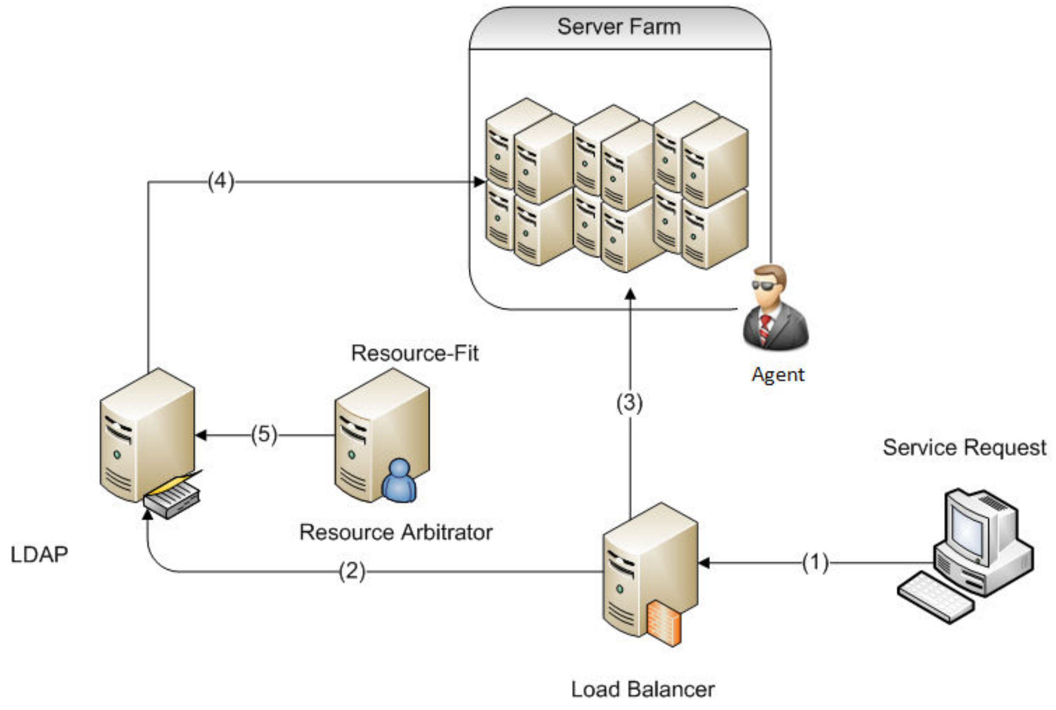


Figure 2.7: Component in the solution framework [19]

Figure 2.7 depicts the various components of the proposed models by Lee et al. In the Figure 2.7, when a session starts, a huge amount of request arrived at server farm, then load balancer initiate the arbitrator by transferring the information about the requested application to it. Then the arbitrator consult the application policy settings, selects the servers according to their recent resource capacities, then push the server candidate list into an internal queue, and then extract it and again submitted it back from the queue to the load-balancer.

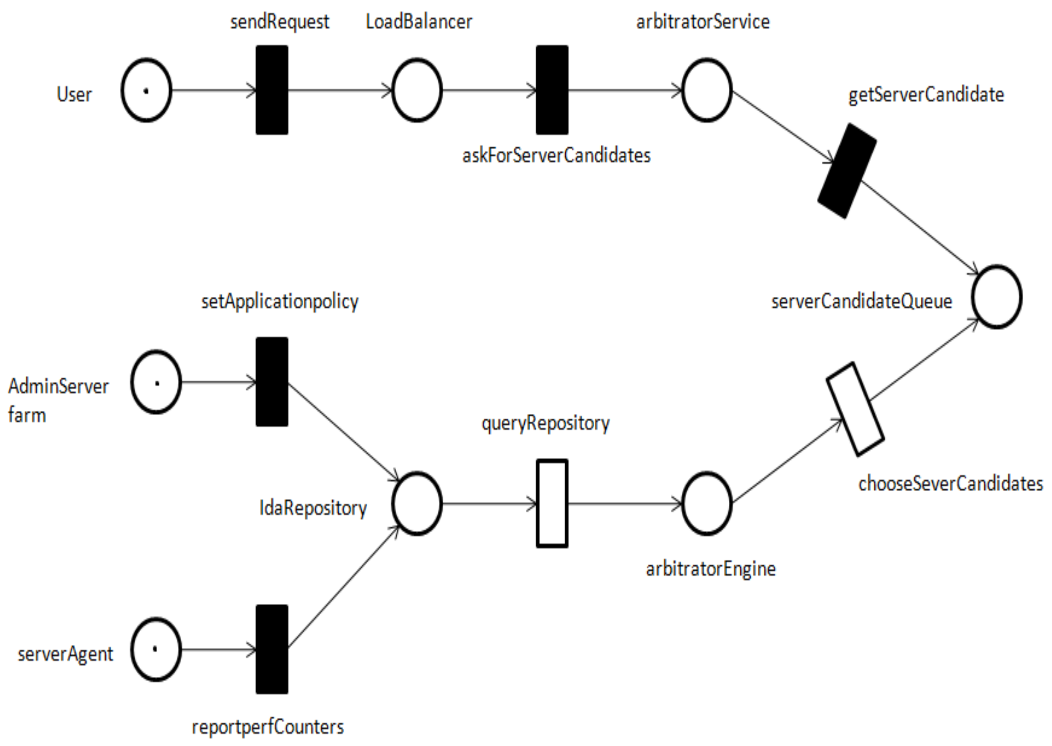


Figure 2.8: The process of load balancing in the solution framework [19]

Arbitrator will act like an effort-save manner. Instead of searching new servers to reply back to the next arbitration request generated by the load-balancer, the arbitrator will regularly use the last finding by returning the chosen servers from the queue unless it is empty. The load-balancer schedules workload to the recommended servers. Then inform their weighted performance counters to the arbitrator regularly to update their status.

Branko et al. [20] have analyzed the issues of load balancing in the Cloud computing environment and proposed a new load balancing algorithm which incorporates information from virtualized environments and end user experience.
Figure 2.9 shows the proposed load balancing model:

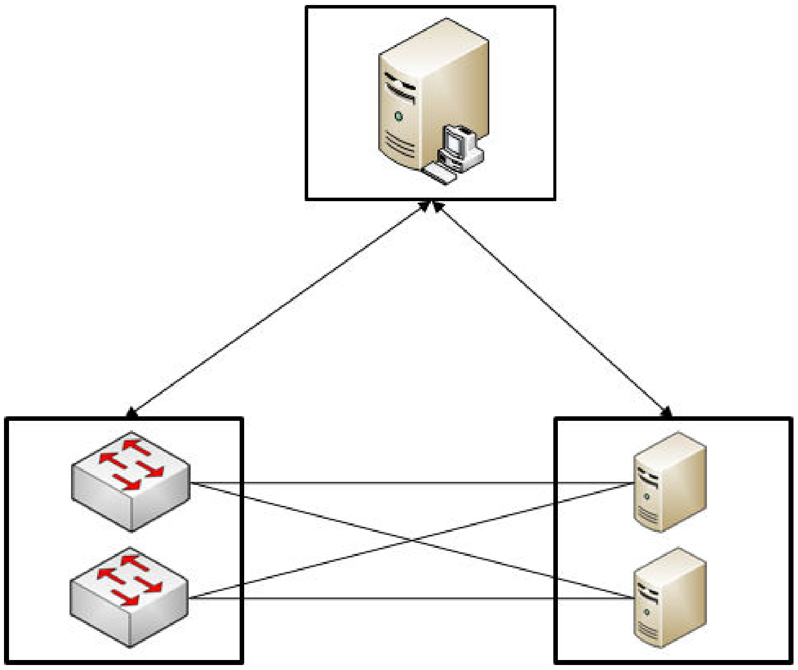


Figure 2.9: Placement of Central Load Balancing Decision Module (CLBCM) in a computer system [20]

In figure 2.9, authors have introduced a central module that influences the decision taken by load balancers. The objective of this module is to monitor all parts of system. After that, based on gathered information and internal calculation, CLBDM will influence the decision of load balancers.

2.2 Distributed load balancing algorithms:

Yemanto et al. [21] have proposed a load balancing algorithm based on replication method in P2P network. They said, simple replication method is not effective in distributed system where some nodes degree is too high, leads to the wastage of storage and processing capability. Simple replication also did not consider the read and write operation overhead. So Yemanto et al. have devised two efficient replication based approach to balance the load more effectively in term of read and write operation overhead.

- *Path Random Replication.*
- *Path Adaptive Replication.*

1.) Path random replication: To reduce the wastage of storage capacity and processing capability, authors have introduced the concept of replication ratio. Replication ratio is proportion of generated replicas to all the moderate peers on the way for every requested data. The replication ratio computed ahead of time. The generation and placement of replica in the intermediate peer is based on the probability of pre-computed replication ratio. Authors major concern is deciding a sufficient replication ratio that will alleviate the convergence of load on the few high degree peers.

2.) Path adaptive replication: Based on the resource status and pre-computed replication ratio, probability of replication in each peer is determined. Probability is represented in each peer as an f(x), here x is storage utilization.

$$f(x) = 1 - \frac{F(x)}{f(1)} = 1 - \frac{1 - e^{-\lambda x}}{1 - e^{-\lambda}}$$

$$\int_0^1 f(x)dx = Ratio$$

Randles et al. [22] have comparatively analyzed the distributed load balancing algorithms in cloud computing. Authors have compared the following distributed load balancing algorithms:

- *Honeybee Foraging Behavior*
- *Biased Random Sampling*
- *Active Clustering*

Honeybee Foraging Behavior: This algorithm inspired by behavior of honeybees foraging and harvesting food. This approach is employed as a searching technique. In honeybee load balancing approach, set of servers are divided into virtual servers. Each virtual server is serving a virtual service request queue. To measure the bee’s quality, cost of serving the request is calculated which gives the profit.

Biased Random Sampling: in this approach, load of a server is measured by its connectivity in a virtual graph. Initially a network is created with virtual nodes that represent each server. Degree of each server node is mapped to available resources.

Number of incoming edges gets connected with randomly selected nodes. Through this edge dynamics load allocation is required for load balancing. When a node process a new task, it deletes an inward edge, represent reduction in tasks. Adversely, when a node completes its task, a new incoming edge is added. The process of increment and decrement is performed via Random Sampling. During sampling, at each step node select its one of neighbor randomly.

Active Clustering: It is self-aggregation algorithm to reconstruct the network. This approach works on the principle of similarity group. Active clustering consists of following iteration:

- At any time (random), a node acts as an initiator and select randomly different type of nodes from its neighbors.
- The matchmaker node leads to a link to be created between one matchmaker nodes.
- The matchmaker deletes that links.

Yao et al. [23] have proposed an improved Artificial Bee Colony algorithm. They have experimentally represented that ABC based load balancing algorithm outperform the basic ABC algorithm [24, 25]. Authors have said that previous load balancing algorithms consider only lightly loaded node and execute a lot of requests e.g. newly arrived request and requests coming from heavily loaded nodes. This leads to load imbalance again.

Kun-Li et al. [26] have proposed a scheduling algorithm based on the Load Balancing Ant Colony Optimization (LBACO) which is an enhanced version of simple ACO algorithm[27,28]. Authors have tried to balance the load and minimize the response time of a tasks. Authors have simulated the approach using the Cloudsim simulator [29,30] and compared it with FCFS and basic ACO algorithm

Lu et al. [31] have proposed a hybrid control mechanism for load balancing and dynamic migration technique in cloud storage. Le et al. main motive is to minimize the overall response time and efficiently adjust the global load.

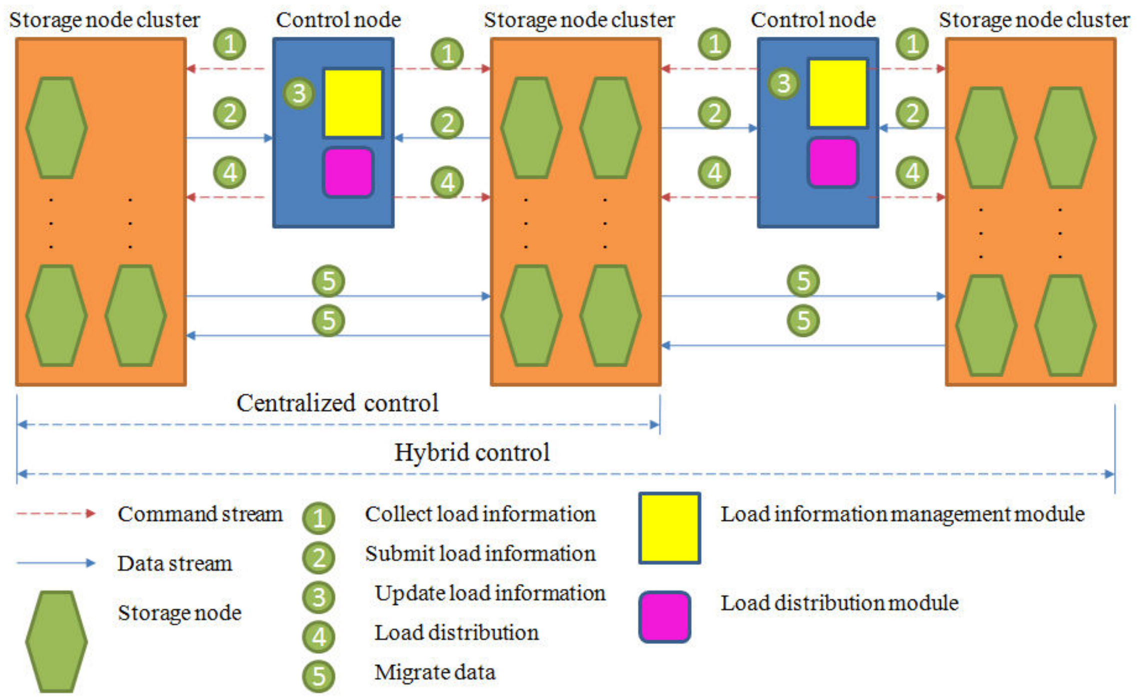


Figure 2.10: Hybrid control strategy for load balancing [31]

Figure 2.10 shows the process of hybrid control strategy for load balancing proposed by Lu et al.

1. The load information management module regularly distributes the information to the storage node.

2. The load information accumulated and updated by the storage node which will submit to the control node.

3. The original load information and the recently submitted load information will integrated by the management module.

4. The distributing module responsible for load migration. The information contains the sender node, the receiver node, the migration amount and the migration quota.

5. The sender node and receiver node establish.

Hung-Chang et al. [32][33] have proposed a distributed algorithm for load rebalancing in distributed file systems in cloud environment. They have enhanced the performance of Hadoop distributed file system by implementing load balancer algorithms. Authors have devised a distributed load rebalancing algorithms to eradicate the above mentioned issues. In order to tackle with the load imbalance problem, authors have implemented their approach by shifting the load rebalancing task to storage nodes which spontaneously migrates the load to reach balance state. This removes the dependence on central nodes. In their approach, the storage nodes are organized as an overlay network based on distributed hash tables (DHTs). A file chunk can be discovered by refer to rapid key lookup in DHTs.

Author have proposed that a DFS is said to be in load balance state if each chunk server host not more than average number of chunks say A. Proposed algorithm follows two properties:

- *Low movement cost*
- *Fast convergence rate*

Authors have also reduced the time complexity of proposed algorithm by pairing top-j under-loaded chunk server with top k overloaded chunk server.

Figure 2.11 depicts a working example of proposed algorithm. There are $n = 10$ chunkservers in the system; the initial loads of the nodes are shown in Figure 2.11(a). Let us consider $\Delta L = \Delta U = 0$. Then, nodes $N1$, $N2$, $N3$, $N4$ and $N5$ are under-loaded nodes, and $N6$, $N7$, $N8$, $N9$, and $N10$ are heavily loaded nodes. Suppose that $N1$ performs the load balancing algorithm. Note that each node performs its load balancing task independently. $N1$ first enquiries the loads of $N3$, $N6$, $N7$, and $N9$ selected randomly from the system Figure 2.11(b).

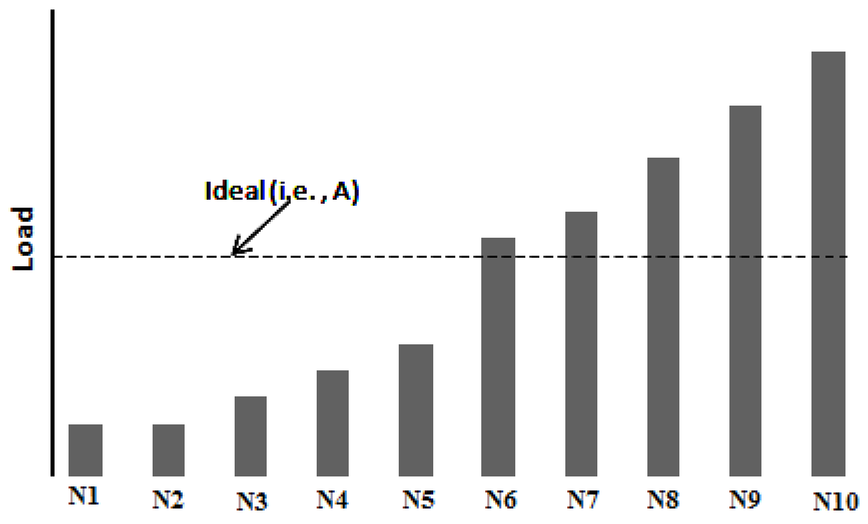
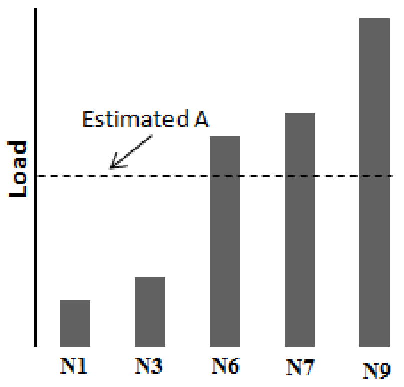


Figure 2.11(a): Initial loads of chunk-servers



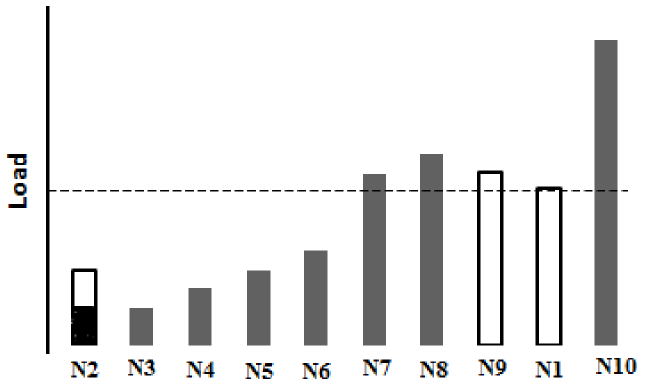


Figure 2.11 (b): *N*1 samples the load   Figure 2.11 (c): Load migration

Based on the gossip based aggregation protocol, *N1* got a sample of randomly selected nodes as shown in figure 2.11(b). *N*1 calculates the ideal load *A* (i.e., *AN*1 $= \frac{LN1+LN3+LN6+LN7+LN9}{5}$) that it needs to host. It then finds that it is a light node. *N*1 then leaves the system by transferring its load to its successor *N*2. As *N*1 is the lightest among *N*1 and its sampled nodes *{N*3, *N*6, *N*7, *N*9*}*, it rejoins as the successor of the heavy loaded node (i.e., *N*9) as shown in figure 2.11(c). *N*1 allocates N9 - *AN*1 chunks from *N*9. Authors conclude that their devised load balancing algorithms harmonized the loads of nodes and reduce the claimed displacement cost as much as possible. Authors have also compared their proposal with centralized algorithm in the Hadoop HDFS productions.

Prabavathy et al. [1] proposed a dynamic load balancing algorithm to balance the load across the various storage servers when cloud storage expands. Here, authors have attempted to balance the load during the data placement as well as in any later situation that lead to imbalance. Authors have also proposed suitable algorithms for data placement, rebalancing and data migration to achieve load balancing in the private cloud storage. Author proposed that one of machine act as a centralized coordinator. It acts as an interface between commodity machines and the client. As storage space within the private cloud is limited, so de-duplication approach is used for efficient usage of the storage space. This approach attempts to find the duplicate content across the files. Figure 2.12 represents the Load balancer model:

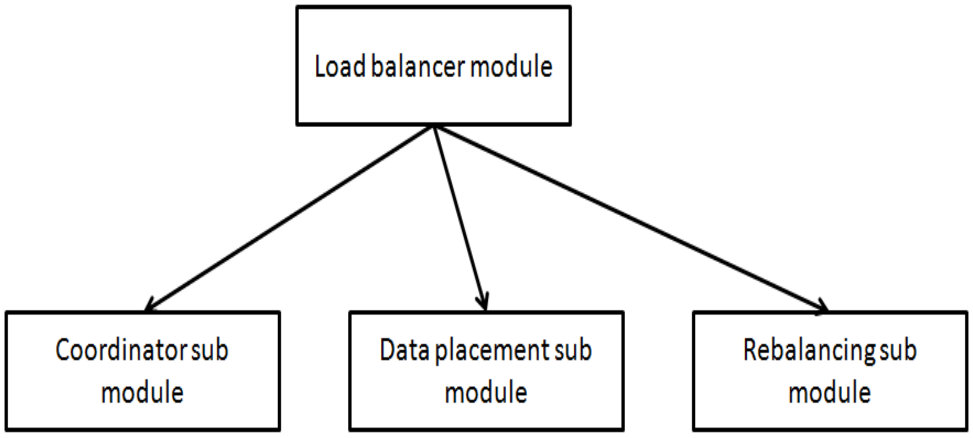


Figure 2.12: Load balancing model

Load balancer consists of coordinator, data placement and load rebalancing sub modules.

Coordinator sub module: This module acquires all the status information from various registered storage nodes which is represented in a vector. It gets the unique chunks from de-duplication engine.

Data placement sub module: This module acquires required information regarding to storage cluster as well as the file chunks from the coordinator sub module.

Load rebalance sub module: This module periodically checking storage clusters to check which storage nodes are lightly or heavily loaded. Then according to that information it migrate the chunks of files to respective storage node.

Manfredi et al. [34] have devised an effective distributed control law for load balancing in content delivery network. Authors have derived the proposed law using fluid flow behavior model of the network of servers. Authors have devised and verified a lemma for network queue equilibrium. Authors have used this lemma to develop a unique distributed time continuous load balancing algorithm. Authors have compared their algorithm performance with RR algorithm, random algorithm, least loaded algorithm and 2RC algorithm in term of average queue length, response time and scalability, which shows great improvement. Authors have described three fundamental load balancing models [35]:

- *Queue adjustment model*
- *Rate adjustment model*
- *Hybrid adjustment model.*

Queue adjustment model: In this model, client requests are directly inserted into the server queue, to be scheduled by scheduler. Scheduler is placed between server queue and server itself. Scheduler pops the requests from queue and decides whether schedule to local server or remote server. The working of model depicted in the following figure 2.13:

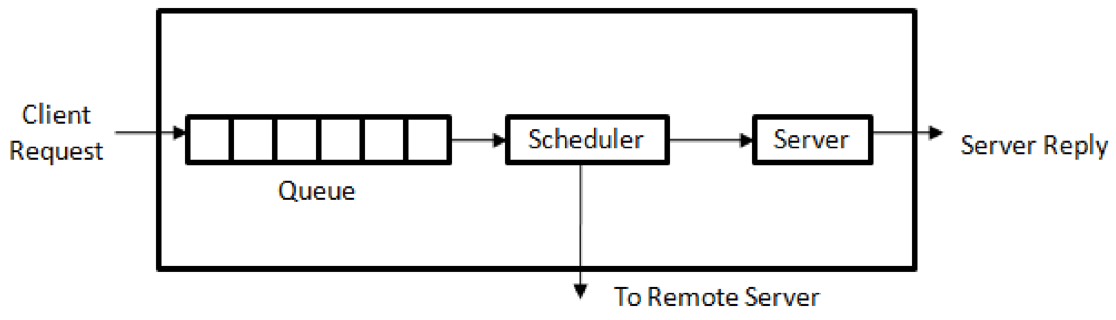


Figure 2.13 Queue adjustment model

Rate adjustment model: In this model, arrival of client requests is firstly scheduled by the scheduler and then push into the queue of local server for further processing. If server is busy then arrived requests scheduled to remote server. Generally, scheduler placed in front of local server queue. Figure 2.14 represents the rate adjustment model:

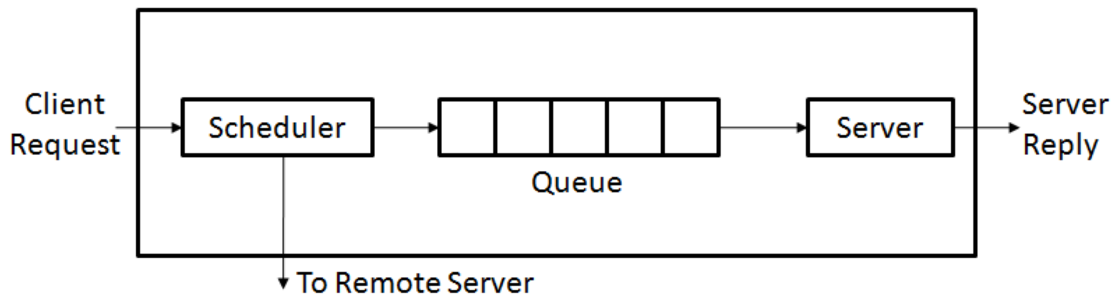


Figure 2.14: Rate adjustment Model

Hybrid adjustment model: In this model, scheduler can manage both local server queue length and incoming client requests of a server. This approach is more effective in load balancing in dynamic environment. The model is shown in figure 2.15:

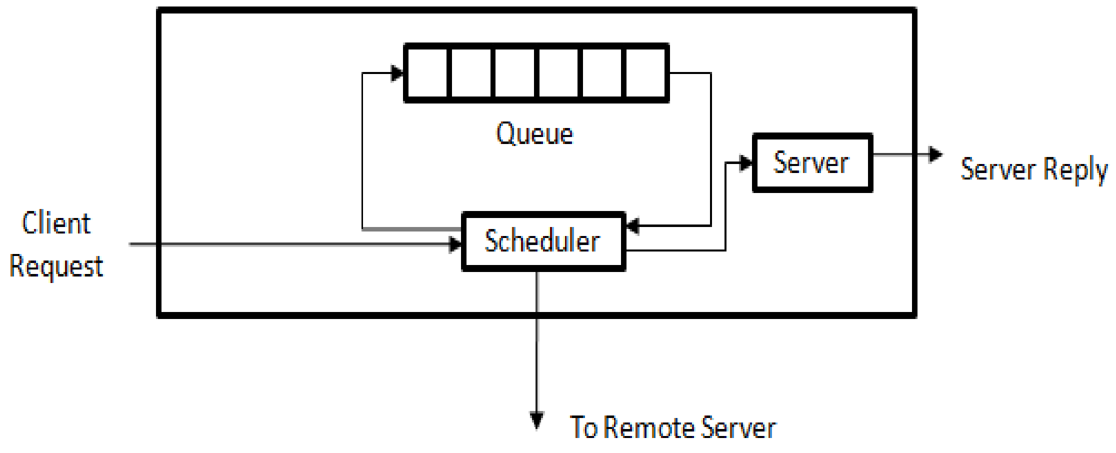


Figure 2.15: Hybrid adjustment model

Authors have used the continuous fluid flow model of server queue to formulate their approach. This model represents the dynamic nature of server queue. Authors have modeled their approach to stabilize the local instability in each server which leads to the global stability of system. Authors have assumed that global resources of network are near to saturation and primarily emphasis on the critical condition i.e. input rate is greater than the output rate.

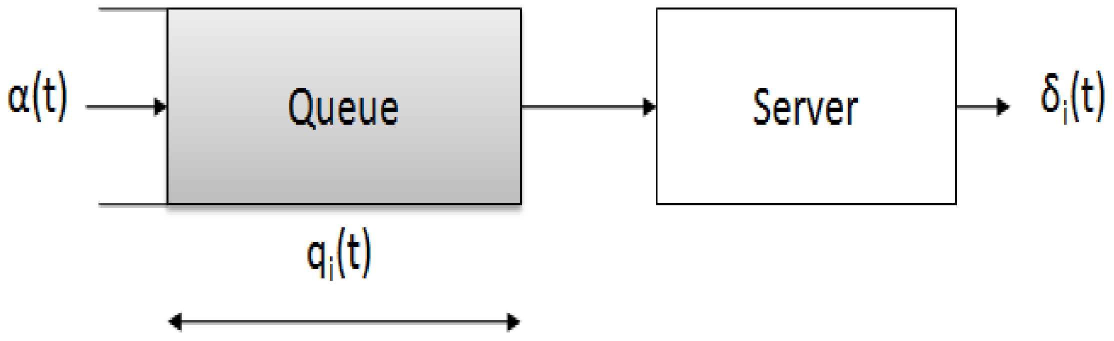


Figure 2.16: Fluid queue model

Figure 2.16 shows the dynamic nature of a server queue whose length varies like movement of fluid.

2.3 Conclusion

In the literature review, some authors have proposed centralized load balancing approach and some authors have proposed decentralized load balancing approach to solve the issues related to proposed problem statement. Both types of approaches have some pros and cons. So the applicability of the any approach depends on the scenario used by the authors. The main motive of literature review is to thoroughly analyze the research work related to proposed problem statement and design an approach to solve the issues related to proposed problem statement.

CHAPTER 3

PROPOSED WORK

3.1. Proposed approach

We have proposed two distributed load balancing algorithms in cloud environment. In the first proposed algorithm, we have considered that each server has different service rate, and queuing capacity but same service time for client request and simple client request. Proposed algorithm balances the load of each server using server parallel processing capability and another proposed algorithm, we have considered that each server has different service rate, different queuing capacity and different service time for client requests and deadline based client request.

Here we have implemented queue adjustment model [35]. In this model, client request directly inserted into the server queue, after that they are scheduled by a scheduler. Scheduler is placed between server queue and server pops the requests from queue to decide whether schedule to local server or remote server. Depicted working model in shown in the figure 3.1:

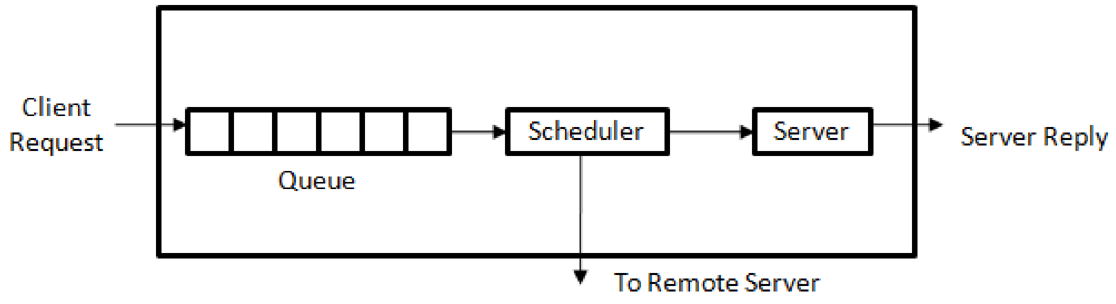


Figure: 3.1 Queue adjustment model

3.2 Capability based Distributed Load balancing Algorithm (CDLBA)

In our first proposed algorithm, we have considered two main parameters of a server, 1) Server request queue size which represents the buffer space to store the incoming client requests handle by server, 2) Service rate λ which represents the number of processing elements (PE) available for processing the client request in a server. Here each server maintains a request queue to buffer the client requests and processed them. Now a day modern servers are equipped with many features like multiple processing elements (PE), large storage, high I/O capability etc. We have selected multiple processing elements (PE) as a main parameter for implementing proposed load balancing approach. We have taken the advantages of this feature to select the appropriate server during functioning of our proposed approach.

Following are the assumptions that have been considered for proposed approach:

- We have assumed a situation where some storage servers get huge client requests and some remains in idle state.
- We have assumed that all servers are strongly connected with each other through high speed dedicated network which shows that the network latency is very low and does not affect the performance of during implementation of proposed approach.
- We have assumed that each server maintains a global view. Here global view represents the status information and load information of its neighbors through control server.

Fig.3.2 shows the organization of distributed storage servers in cloud. In fig.3.2, there are N servers where N$\in \{1,2,3 \dots . n\}$ present in the system. Each server has a queue to buffer the incoming requests, number of processing elements for parallel processing and some storage space. We have assumed all servers are geographically distributed across the globe. In the fig.3.2, servers are connected with each other through a high speed dedicated network. Each server is used by their respective region requests. Due to overloading of requests from any region, incoming request rate increases exponentially on a particular server.

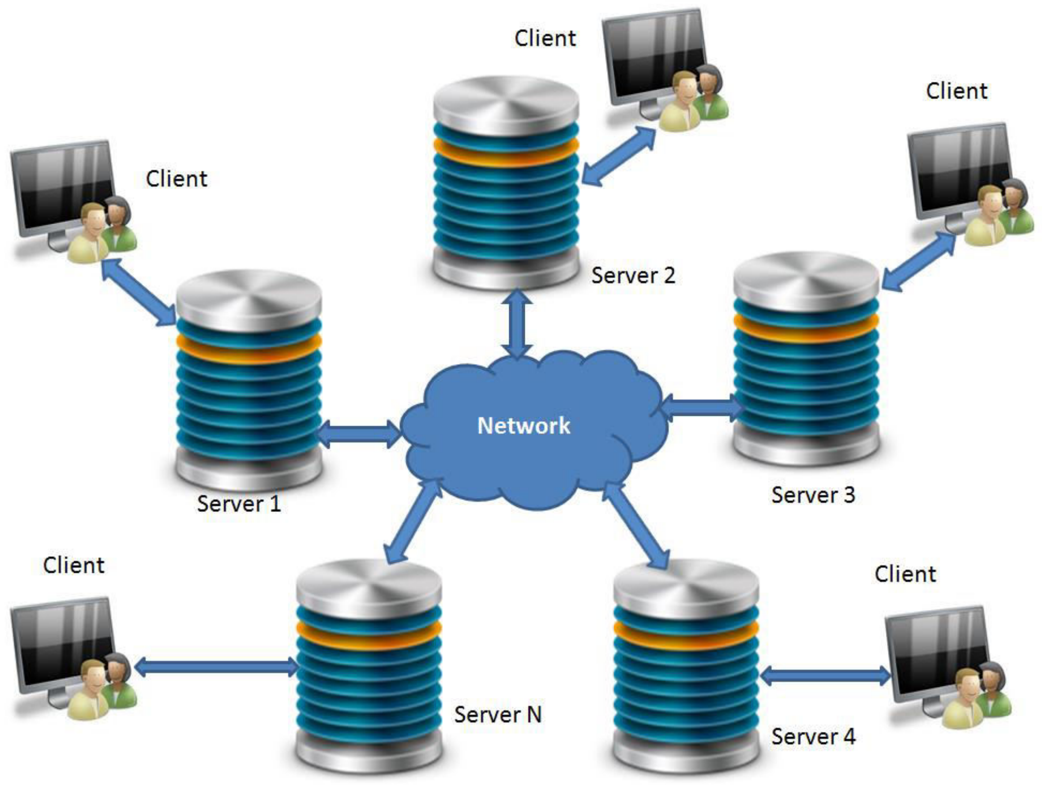


Fig.3.2: Organization of storage server

When a server gets a request, it processes them if it is idle or least loaded otherwise server stores them to their queue. In case, if request rate is higher than the service rate then its queue size increase exponentially and system becomes unstable. In order to maintain the stability of the whole system, every server sets its threshold limit, to acknowledge whether it is least loaded or overloaded. When the size of queue is greater than threshold limit then server is considered as an overloaded. Once the server gets overloaded, it triggers the load balancing mechanism. Load balancer classifies the least loaded server based on their request queue capacity as well as processing capability. Then it searches for the highest least loaded server. When a load balancer found the highest least loaded server then overloaded server migrate its load to the available server. In this way, load balancer balances the load of overloaded servers. Various notations have been used in the proposed approach which are listed below:

ρ: Queue size of server.

λ: Service rate that is number of request processed simultaneously by a server.

$Q\_L_{threshold}$: Threshold limit of server queue.

$Q\_L_{Current}$: Exixting capacity of server queue at time t.

ΔL_i: extra amount of load on server i.

We have considered the real world scenario so that the request queue size and service rate changes with respect to time t, which are represented as $\delta\rho$ and $\delta\lambda$ respectively

$$\delta\rho = \frac{\rho}{\delta t}\ and\ \delta\lambda = \frac{\lambda}{\delta t} \tag{3.1}$$

Storage server is said to be overloaded if:

$$Q\_L_{current} > Q\_L_{threshold} \tag{3.2}$$

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

$$\Delta L_i = Q\_L_{current} - Q\_L_{threshold} \tag{3.3}$$

When a overloaded server triggers its load balancer module, then the conditions to trigger the load balancer on server *i* is given as follows

$$T(i) = \begin{cases} 1, & \Delta L > 0 \\ 0, & otherwise \end{cases} \tag{3.4}$$

Where *T(i)* is trigger function running at ith server.

As the load balancer module gets triggered, server *i* find the least loaded or idle server which can accommodate its load and adequately process them. For this, load balancer calculate the fitness F_j value of neighbors of server *i*. Fitness value F_j where $j \in \{1,2,3 \ldots . n-1\}$can be calculated using fitness function as followed

$$F_j = \alpha_1 \Delta M_j + \alpha_2 \lambda \tag{3.5}$$

$$\Delta M_j = Q\_L_{threshold} - Q\_L_{current} \tag{3.6}$$

Where ΔM_j is free request queue of server j. If ΔM_j is negative, then server j request queue is overloaded otherwise ΔM_j is positive and server j request queue is least loaded.

Where α_1 and α_2 are constant which can vary according to scenario such that

$$\alpha_1 + \alpha_2 = 1 \tag{3.7}$$

Here we have considered the value of α_1 and α_2 is 0.5 for the discussed scenario. This is because both parameters play equal role in load balancing. In this way, load balancer calculate the fitness value of each neighbor server of *i* and select that server which has maximum fitness F_j value and migrate ΔM_j amount of load to server j.

3.3. Flow Chart of CDLBA

Here Figure. 3.3 shows the flow chart of the capability based distributed load balancing algorithm.

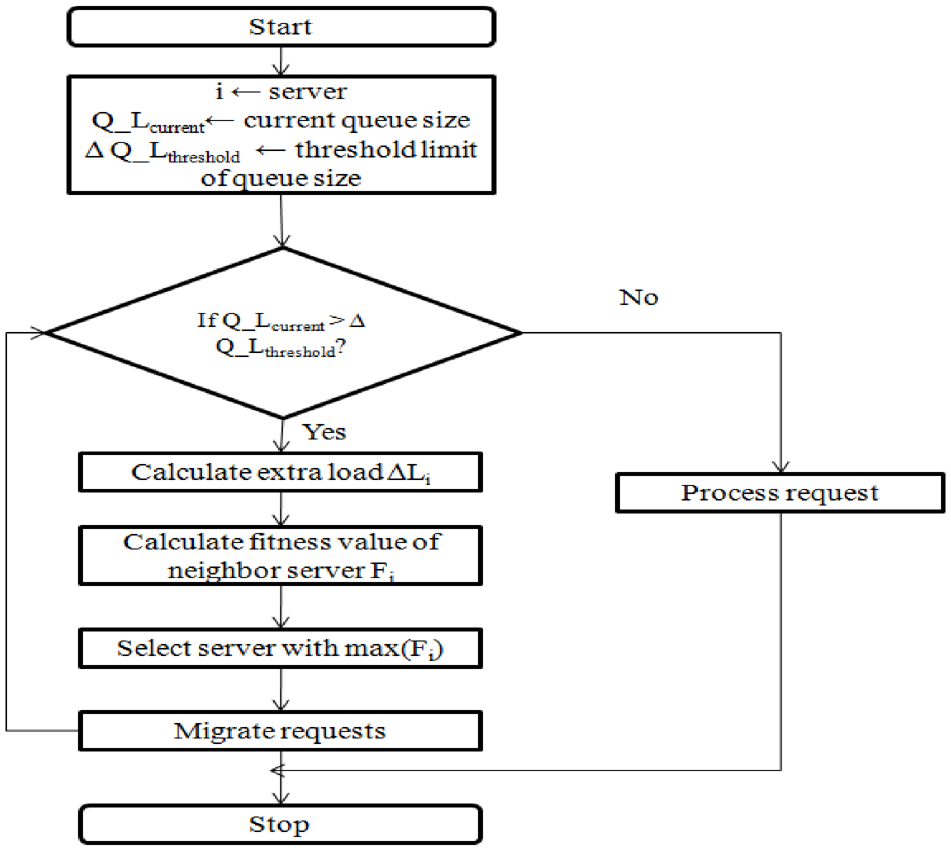

Start
i ← server
Q_L_current ← current queue size
Δ Q_L_threshold ← threshold limit of queue size
If Q_L_current > Δ Q_L_threshold?
No
Yes
Calculate extra load ΔL_i
Calculate fitness value of neighbor server F_i
Select server with max(F_i)
Migrate requests
Process request
Stop

Fig. 3.3: Flow chart of CDLBA

- As shown in the Figure 3.3, initially balancer selects a server *i* whose current queue size is $\Delta Q\_L_{current}$. Let $Q\_L_{threshold}$ be threshold limit of server queue size. The threshold limit signifies that either server is overloaded or least loaded.
- Load balancer continuously checks the server request queue status.
- When a client request arrives at the server, it checks whether current request queue size is greater than threshold limit defined by the server.
- If request queue is available then, add the request and to process them.
- If it is not, then server is considered overloaded.
- It calculates the extra load the server and load balancer searches the least loaded server from its neighbor list.
- It checks its neighbor server and then selects the server whose request queue is least loaded as well as maximum processing elements based on fitness value F_j.
- After selecting the least loaded server it transfers the load to that server. In this way each server balances its load.

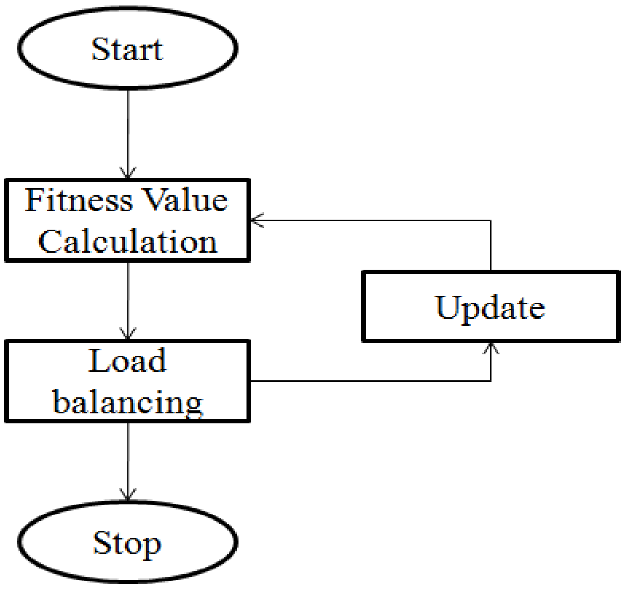


Figure 3.4: Updating of Fitness function

Figure 3.4 shows the flowchart of how fitness value is updated during load balancing. Objective to do this is to reduce the overhead of calculation of fitness value during load balancing.

- Initially server computes the fitness.
- Load balancer uses this fitness value for decision making.
- Server periodically updates fitness value when it is idle.

3.4. Proposed Algorithm 1

Algorithm 1. CDLBA(Server s, $Q\_L_{current}$)

Input : Server s and $Q\_L_{current}$

```
 1. s← server ;
 2. $Q\_L_{current}$ ← current queue size;
 3. $\Delta QL_{threshold}$ ← threshold limit of queue size;
 4. if ($Q\_L_{current} < \Delta QL_{threshold}$ ) then
 5.       //  check server queue status.
 6.       Add request to queue;
 7.        Process_request();
 8.       // processing the client request.
 9. else
10.      //server is overloaded
11.     S ← Find_server( server_neighbour_list L );
12.     // find underloaded server.
13.     S ← migrate request;
14. Stop;
```

Find_server(server_neighbour_list L)

Input : server_neighbour_list L

```
1. for k=1 to L.size();
2.     S1←L.get();
3.      if ($S1.Q\_L_{current} < S1.Q\_L_{threshold}$) then
4.      List L2← Add(S1);
5. for  n=1 to L2.size();
6.      $F_n = \alpha_1 \Delta M_n + \alpha_2 \lambda_n$;
```

```
7.        // calculate fitness value of server n.
8.        temp_list t← $F_k$ ;
9.        // Add to temporary list.
10. for j=1 to t.size();
11.        S2← max($F_j$);
12.       // select maximum fitness value.
13. return S2;
```

In proposed algorithm1, load balancer balances the load of storage server in terms of client requests. When a client or any application sends a request to storage server to acquire online storage space, then server processes the client request and allocates the required storage space. But in real, handling of client requests is different as some servers gets overloaded with huge number of received requests. In this situation, response time of server increases which leads to degrade the QoS. So we have proposed an approach to cope with such a situation. Following steps describe the algorithm in detail:

a) In the proposed load balancing algorithm, load balancer periodically checks the load of server using Eq. (3.2) when the number of client requests received by the server. Proposed algorithm tries to avoid the situation of load transferring as much as possible by utilizing the idle server present in the system. When a request is arrived at server, load balancer module checks the status of the request queue and also checks whether the server request queue is overloaded or not using Eq. (3.2). Here each server maintains its data structure "queue" to buffer all its client requests as for a given instant of time hundreds of client's requests may arrived to the server.
b) If the current queue length is less than the threshold limit of queue length, then it adds the client request to server queue. If any CPU is available, then the request gets processed otherwise that have to be waiting for CPU to be available.

c) If request queue size is greater than the threshold limit, then load balancer assumes that this server is overloaded and triggered a module called *find server()*.

d) In this module, load balancer searches for the least loaded server present in the system by computing the fitness value using Eq. (3.6) for every neighbor server and stores them in a list. From this list load balancer selects the server which has highest fitness value.

e) As proposed algorithm is designed to reduce the response time so it chooses the server on the basis of maximum service rate. We have assumed that each server has multiple processing elements to process large number of requests, which correspond to service rate. Fitness value is calculated using service rate and available queue of server.

f) Once the least loaded server is select then it transfers the load to available server using Eq. (3.5). Load balancer transfers only that amount of load, which are sufficient to balance the load of that server.

3.5. Deadline based Distributed Load balancing algorithm (DDLBA)

This algorithm utilizes the service time along with previously discussed parameters of each server for load balancing. Apart from that we have also considered that client requests have some time constraint called as deadline time under which they have to be processed by server. This proposed algorithm evaluates the utilization of each server when service time of each request is heterogeneous in nature. This algorithm DDLBA has classified the servers into least loaded and overloaded server using following Equation:

$$W_k = \frac{Q\_L_{current\_k}}{\lambda_k} \times S\_T_k \tag{3.8}$$

Here W_k: Waiting time in server *k*.

$Q\_L_{current\_k}$: Current queue capacity of server k.

λ_k: Service rate of server *k*.

S_T_k: Service time of server *k*.

A server is said to be overloaded if

$$W_k > DL_i \tag{3.9}$$

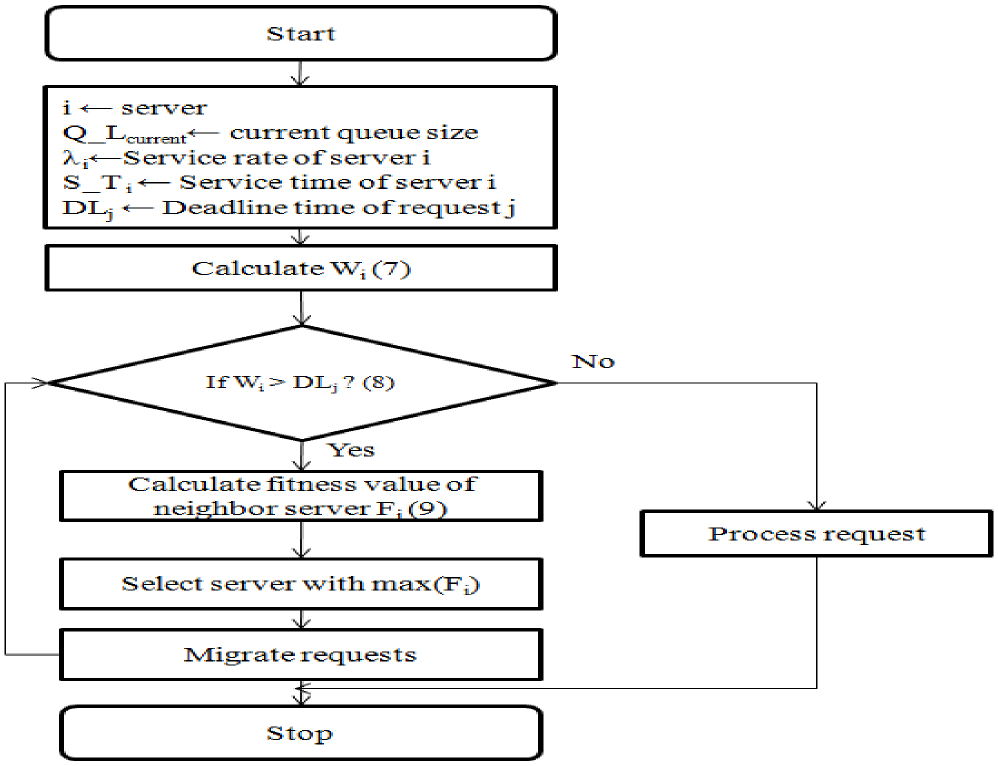
DL_i: Deadline time of request *i*.

When a server is overloaded then it triggers its load balancer module using Eq. (3.4). Load balancer calculates the fitness value F_j value of neighbors of server *i*. Fitness value F_j where $j \in \{1,2,3 \ldots . n-1\}$ can be calculated using fitness function as follow:

$$F_j = \left(DL_i - W_j\right) \; if \; and \; only \; if \; DL_i > W_j \tag{3.10}$$

In this way, load balancer calculate the fitness value of each neighbor server of *i* and select that server which has maximum fitness F_j value and transfers the sufficient client requests to server *j*.

Here Figure 3.5 shows the flow chart of deadline based distributed load balancing algorithm.



Start
i ⟵ server
Q_L_current ⟵ current queue size
λ_i ⟵ Service rate of server i
S_T_i ⟵ Service time of server i
DL_j ⟵ Deadline time of request j
Calculate W_i (7)
If W_i > DL_j ? (8)
No
Yes
Calculate fitness value of neighbor server F_i (9)
Select server with max(F_i)
Migrate requests
Process request
Stop

Figure 3.5: Flow chart of DDLBA

- Here in Figure 16, initially balancer selects a server i whose current queue size is $\Delta Q\_L_{current}$, service rate is λ_i, service time is S_T_i and deadline time of request j is DL_i.
- Load balancer continuously checks the server request queue status. Server i compute the waiting time $W_{k.}$ using Eq. 3.8.
- When a client request arrived at the server, it checks whether waiting time $W_{k.}$ is greater than deadline time of incoming request j.
- If server is least loaded, then add the request and process them.
- If it is not, then server is considered to be overloaded.
- It calculates the extra load on the server. Then load balancer searches the least loaded server from its neighbor list.
- It checks its neighbor server and then computes the fitness value F_j using Eq. 3.9 with its neighbor.
- After selecting least loaded server it transfers the load to the server. In this way, each server balances its load.

Algorithm 2 DDLBA (Server s, $Q\_L_{current}$, $\lambda_k, S\_T_k,$ DLi)

```
Input : Server s and $Q\_L_{current}$, $\lambda_k$, $S\_T_k$, $DL_i$
1. s← server ;
2. $Q\_L_{current}$ ← current queue size;
3. $\lambda_k$ ← Service rate of server k;
4. $S\_T_k$ ← Service time of server k;
5. $DL_i$ ← Deadline time of request I;
6. Compute $W_k$;
7. if ($W_k < DL_i$ ) then
8.     // check server queue status.
9.     Add request to queue;
10.      Process_request();
11.     // processing the client request.
12. else
```

```
10.     //server is overloaded
11.     S ← Find_server( server_neighbour_list L );
12.     // find underloaded server.
13.     S ← migrate request;
14. Stop;
```

Find_server(server_neighbour_list L)

```
Input: server_neighbour_list L
1. for k=1 to L.size();
2.     S1←L.get();
3.     if ($S1.W_k < DL_i$) then
4.     List L2← Add(S1);
5. for n=1 to L2.size();
6.     $F_n = (DL_i - W_j)$;
7.     // calculate fitness value of server n.
8.     temp_list t← $F_k$ ;
9.     // Add to temporary list.
10. for j=1 to t.size();
11.     S2← max($F_j$);
12.     // select maximum fitness value.
13. return S2;
```

Here, we have utilized service time and service rate of server as a key parameter for load balancing in the distributed environment where client request associated with some deadline time. In cloud storage, storage servers are heterogeneous in nature with respect to various parameters such as service rate, service time, storage queue capacity etc. The main objective of proposed algorithm is to utilize the server in the cloud storage that is heterogeneous in service time. Following steps describe the working of DDLBA:

a) DDLBA algorithm tries to avoid the situation of load transferring as much as possible by effectively utilizing the service time of every server in the system. In the first step, load balancer calculates the current queue capacity of server. Then load balancer compute the waiting time using Eq. (3.8). Load balancer performs the task of continuously monitor the load status of server using Eq. (3.9). So when a request arrived at a server, load balancer module checks the waiting time for incoming request using Eq. (3.7). It checks whether the server is overloaded or not using Eq. (3.9). Since each server maintains a data structure “queue” to buffer its client requests because at any instant of time hundreds of client’s requests arrived at the server. And server cannot process all the requests simultaneously.

b) If the waiting time is less than the deadline time of incoming request, then it adds the client request to server queue. If any of CPU currently available, then that request processed. Otherwise that has to be waiting for CPU time.

c) If request waiting time is greater than the deadline time of request, then load balancer assumes that server is overloaded and triggered a module called *find server()*.

d) In this module, load balancer searches for the idlest server present in the system by computing the fitness value using Eq. (3.10) for every neighbor server and stores them in a list. From this list load balancer select that server who has highest fitness value.

e) Since proposed algorithm main motive is to reduce the response time so it selects the server with least utilization of CPU. We have assumed that each server has multiple processing elements to process large number of requests, which correspond to service rate and different service time. So fitness function computes the fitness value using server current queue capacity, service rate and service time.

f) Load balancer transfer only that amount of load, which makes the server balanced.

In this way when any of servers get overloaded in terms of client requests, load balancer balances the load to migrate its requests to the least loaded server that can process with in deadline of request leads to increase the QoS.

CHAPTER 4

SIMULATION AND RESULT ANALYSIS

4.1 Simulation Environment

We have analyzed the performance of proposed algorithm using simulations. We have designed our simulation environment in Java for which we have used Netbeans7.0 tool. To provide real world scenarios, we have created storage servers and client requests were sent using multi threading. Here, all the storage servers executes in parallel and client requests also generated in parallel. We have generated some set of client requests for particular server which is represented as end users. Servers are associated with a queue that stores the client's requests and has storage capacity. Each server has multiple processing elements to serve the client requests concurrently. In our simulation, we have generated load in server in term of client requests. We have created a situation of system in stability where the load is transferred across the servers. To effectively analyze the performance of our proposed algorithms, we have generated the load only for half of storage servers in the system. This will create the situation of load unbalancing because some servers remain idle at the start. Our motive is to equally distributetheclientrequeststoeachserversothatnooneservergetsoverloaded.Wehave created following two test-beds to regressively analyze our proposed algorithm.

a) We have considered six servers whose queue length and service rates are different but each server may or may not have equal processing time.

b) We have increased the number of servers to 12.

We have simulated the performance of our algorithms by using wide range of client requests. We have compared our simulation result with the least loaded load balancing algorithm.

4.2. Simulation Environment for CDLBA

We have simulated the performance of proposed algorithm CDLBA through two testbeds. The configuration parameter of testbed-1 is shown below in table 4.1.

Table 4.1: Configuration of testbed-1

	Server Id					
	S1	S2	S3	S4	S5	S6
Queue Length(requests)	10	10	20	20	30	30
Service Rate(req./time)	5	7	9	11	13	15

Here, Table 4.2 shows the test case used for the simulation of proposed algorithm CDLBA. We have simulated our proposed algorithm under five test cases which are shown in the table 4.2.

Table 4.2: Test cases for CDLBA

Test case	No. of Requests	No. of Servers	Storage Server
1	600	6	500GB
2	800	6	500GB
3	1000	6	500GB
4	1200	6	500GB
5	1400	6	500GB

4.3.1. Simulation results and discussion

We have compared the performance of our proposed CDLBA algorithm with least loaded algorithm in context to number of requests completed, number of delayed requests, overall response time of system, and average server utilization. Figure 4.1 shows the comparison of number of completed requests in CDLBA with least loaded algorithm. Figure 4.1 also shows the number of client requests which are processed by server in their time means these are the requests which never wait for the CPU availability. We have compared the obtained results with least loaded algorithm. In least loaded algorithm when any server gets overloaded then load balancer selects the server where request queue is least loaded without considering its service rate. But we have also considered the service rate parameter while comparing the algorithms. We have also evaluated obtained results on different set of client requests. This can be seen from figure 4.1 that proposed algorithm has outperformed the least loaded algorithm. In all test cases, proposed algorithm processed more number of client's request in a given time as compare to least loaded algorithm. The below Table 4.3 displays the summary of results.

Table 4.3: Comparison of request completed for testbed-1

Algorithm	Client requests				
	600	800	1000	1200	1400
CDLBA Algorithm	176	145	187	145	157
LL Algorithm	111	108	122	76	97

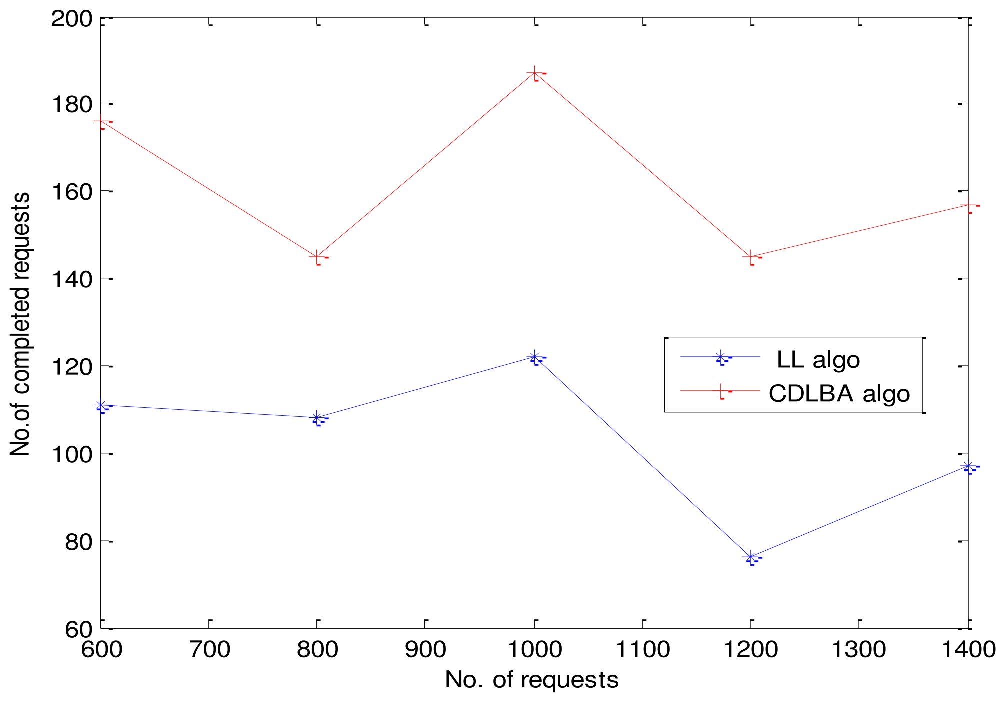


Figure 4.1: Comparison of completed requests for testbed-1

Here, Figure 4.2 shows the comparison of number of delayed request of proposed CDLBA with least loaded algorithm. From Figure 4.2 we can conclude that numbers of delayed requests are less in CDLBA as compare to least loaded algorithm in all test cases thus proposed algorithm outperforms the least loaded algorithm. Table 4.4 displays the summary of obtained results.

Table 4.4: Comparison of delayed requests for testbed-1

Algorithm	Client requests				
	600	800	1000	1200	1400
CDLBA Algorithm	424	655	813	1051	1243
LL algorithm	489	692	878	1124	1303

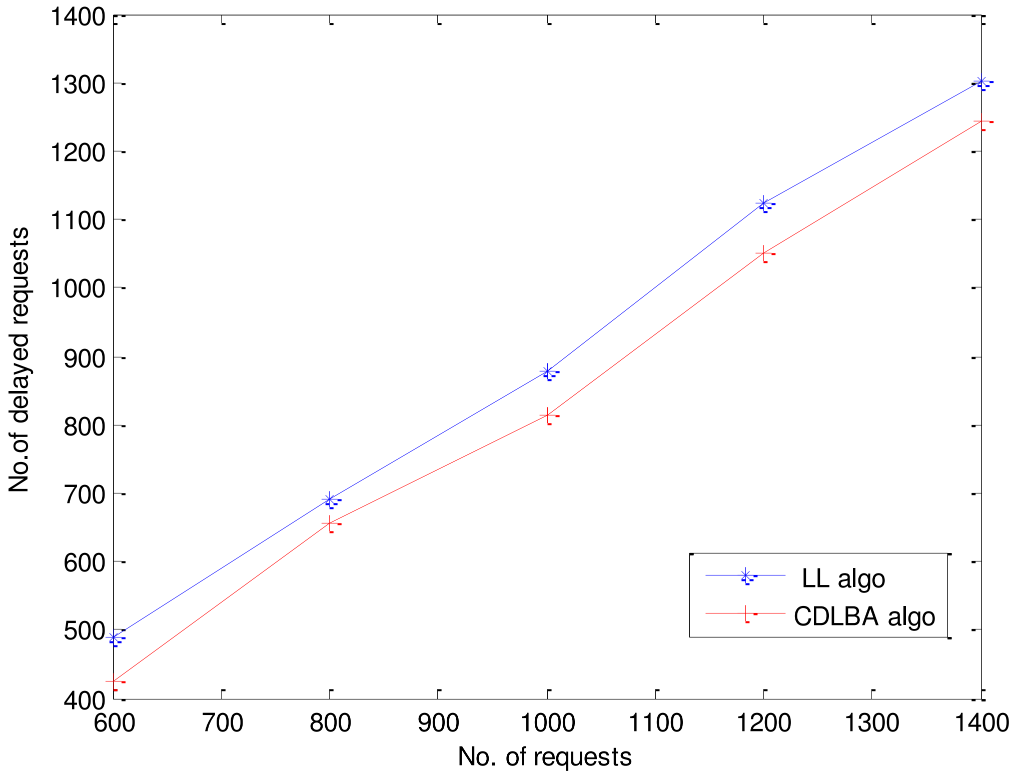


Figure 4.2: Comparison of delayed requests for testbed-1

Figure 4.3, shows the comparison of proposed CDLBA algorithm with least loaded algorithm with respect to overall response time. From figure 4.3 we can say that proposed algorithm outperform here too. In all test cases, response time of proposed algorithm is less than least loaded algorithm. Table 4.5 display the summary of results obtained.

Table 4.5: Comparison of response time for testbed-1

Algorithm	Overall Response time(ms)				
	600	800	1000	1200	1400
CDLBA Algorithm	45502	62000	71104	90044	92517
Least Loaded Algorithm	52672	64837	77898	108859	119817

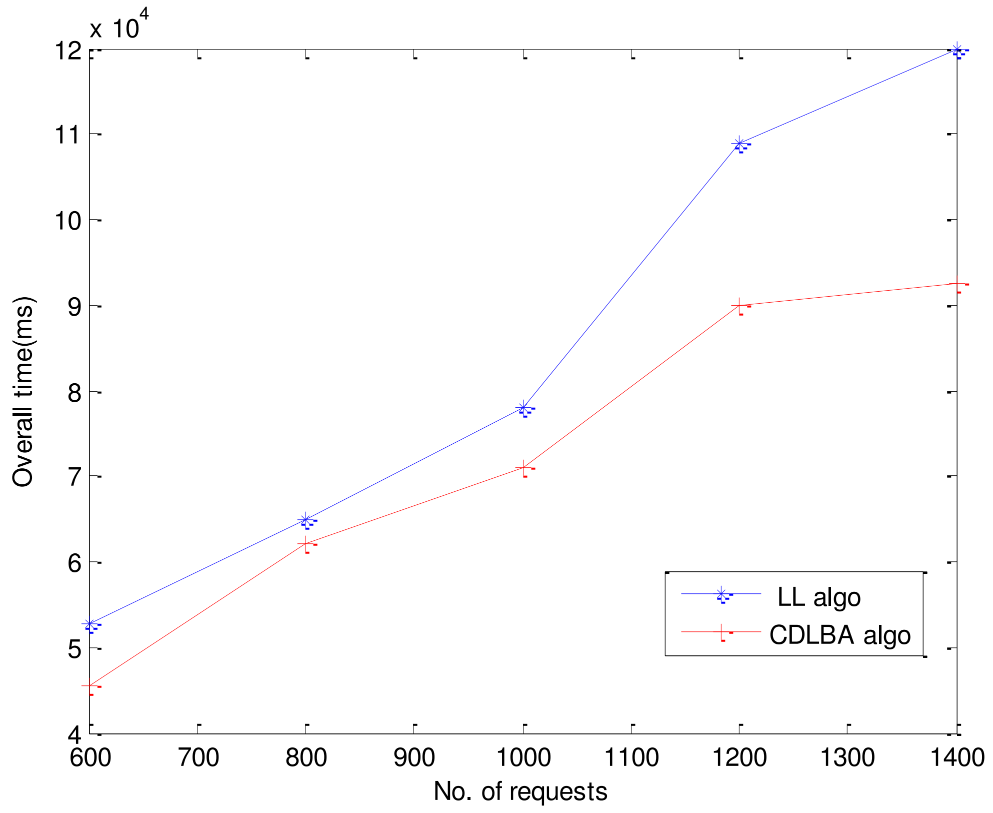


Figure 4.3: Overall response time for testbed-1

Figure 4.4 shows the comparison of proposed CDLBA algorithm with least loaded algorithm in term of server utilization. From figure 4.4, we can say that by increasing the number of requests the server utilization is also increases. Table 4.5 displays the summary of results. Formula for computing the server utilization is as follows:

$$\text{Server total service time(ms)} = \frac{No.of\ requests\ processed\ by\ server}{service\ rate} \times service\ time(ms) \quad (5.1)$$

$$Sever\ Utilization\ (\%) = \frac{Server\ total\ service\ time}{Overall\ system\ time} \quad (5.2)$$

Table 4.6: Comparison of average server utilization for testbed-1

Algorithm	Average utilization (%)				
	600	800	1000	1200	1400
CDLBA Algorithm	35.5	36.3	37.9	39.4	40.2
LL Algorithm	31.4	31.6	32.0	32.1	32.4

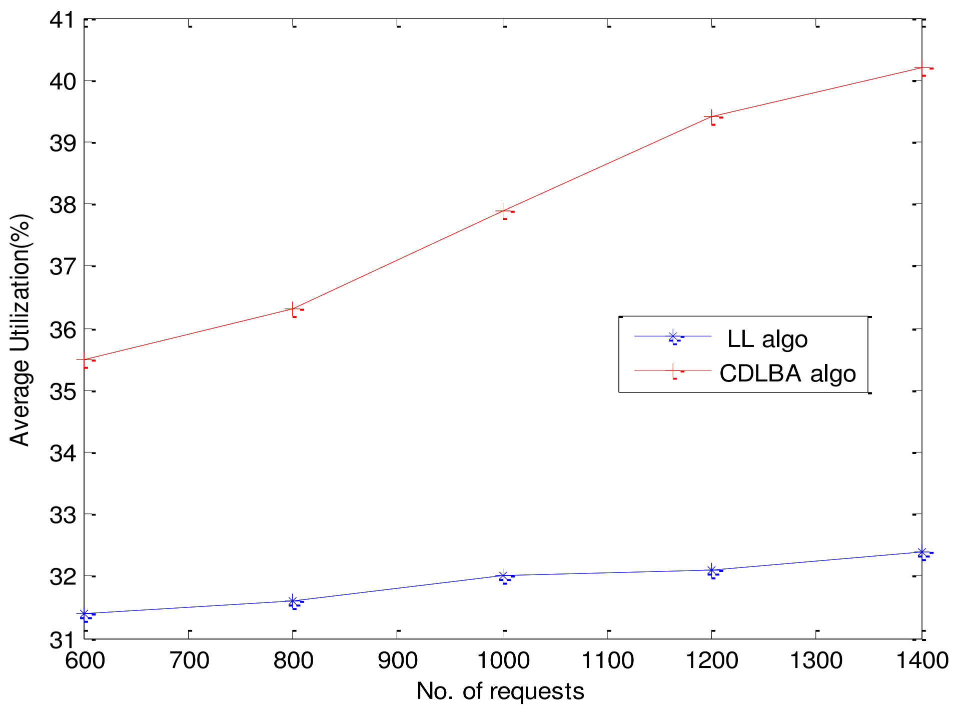


Figure 4.4: Server utilization for testbed-1

We have rigorously analyzed proposed algorithm and simulated it under different test case scenarios. Following table depicts the test case scenario.

Table 4.7: Configuration Parameter for testbed-2

	Server Id											
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
Queue length(request)	10	10	10	20	20	20	30	30	30	50	50	50
Service Rate (req./time)	5	5	6	6	7	7	8	8	10	10	15	15

We have rigorously analyzed CDLBA algorithm. For that we have simulated our proposed algorithm under different test case scenario. Following table depicts the test case scenario.

Table 4.8: Test cases for testbed-2

Test case	No. of requests	No. of Servers	Storage
1	800	12	500GB
2	1000	12	500GB
3	1200	12	500GB
4	1800	12	500GB
5	2400	12	500GB

Figure 4.5 shows the comparison of number of completed requests in CDLBA with least loaded algorithm for testbed-2. Figure 4.5show the number of client requests which gets processed by server in their time i.e. these are the requests which never wait for the CPU availability. The below table 4.9 display the results in tabular format.

Table 4.9: Comparison of request completed for testbe-2

Algorithm	Client requests				
	800	1000	1200	1800	2400
CDLBA Algorithm	651	755	837	967	913
LL Algorithm	577	593	765	723	813

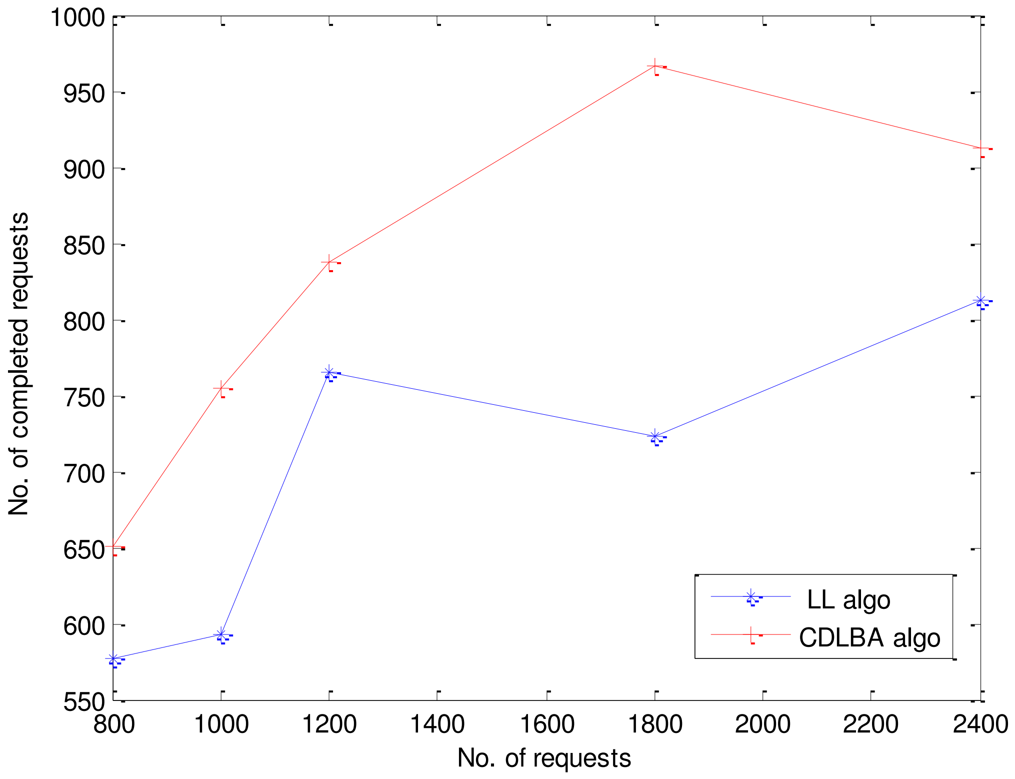


Fig.4.5: Comparison of completed requests for testbed-2

Here figure 4.6 shows the comparison of number of delayed request of CDLBA with least loaded algorithm. From figure 4.6 we can conclude that numbers of delayed requests are less in CDLBA as compare to least loaded algorithm in all test cases. This shows that our proposed algorithm are efficient than least loaded algorithm. The below table 4.10 displays the summary of results.

Table 4.10 Comparison of number of postponed for testbed-2

Algorithm	Client requests				
	800	1000	1200	1800	2400
CDLBA Algorithm	149	275	363	833	1487
LL algorithm	223	407	435	1077	1587

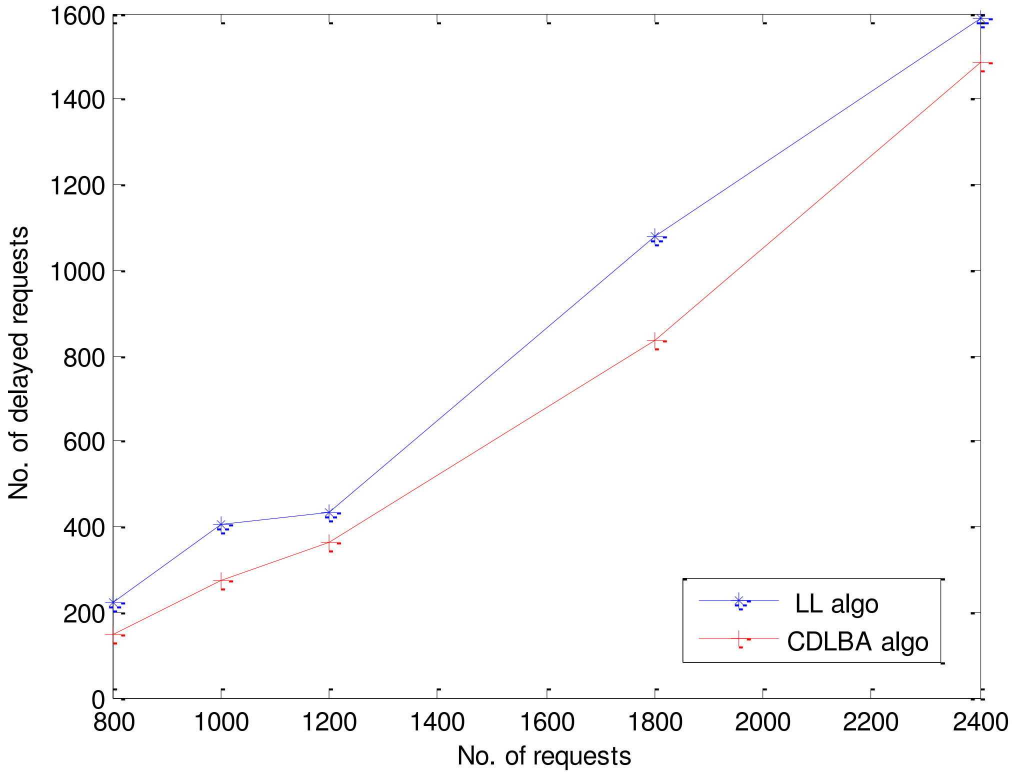


Figure 4.7: Comparison of delayed requests for testbed-2

Here, Figure 4.7 shows the comparison of our proposed CDLBA algorithm with least loaded algorithm in term of overall response time. From figure 4.7 we can conclude that our algorithm outperform the least loaded algorithm. In all test cases, response time of our algorithm is less than least loaded algorithm. The below table 4.11 displays the summary of results.

Table 4.11 Comparison of response time for testbed-2

Algorithm	Overall Response time(ms)				
	800	1000	1200	1800	2400
CDLBA Algorithm	61505	70338	78219	110914	160629
LL Algorithm	66458	83525	97414	127074	172044

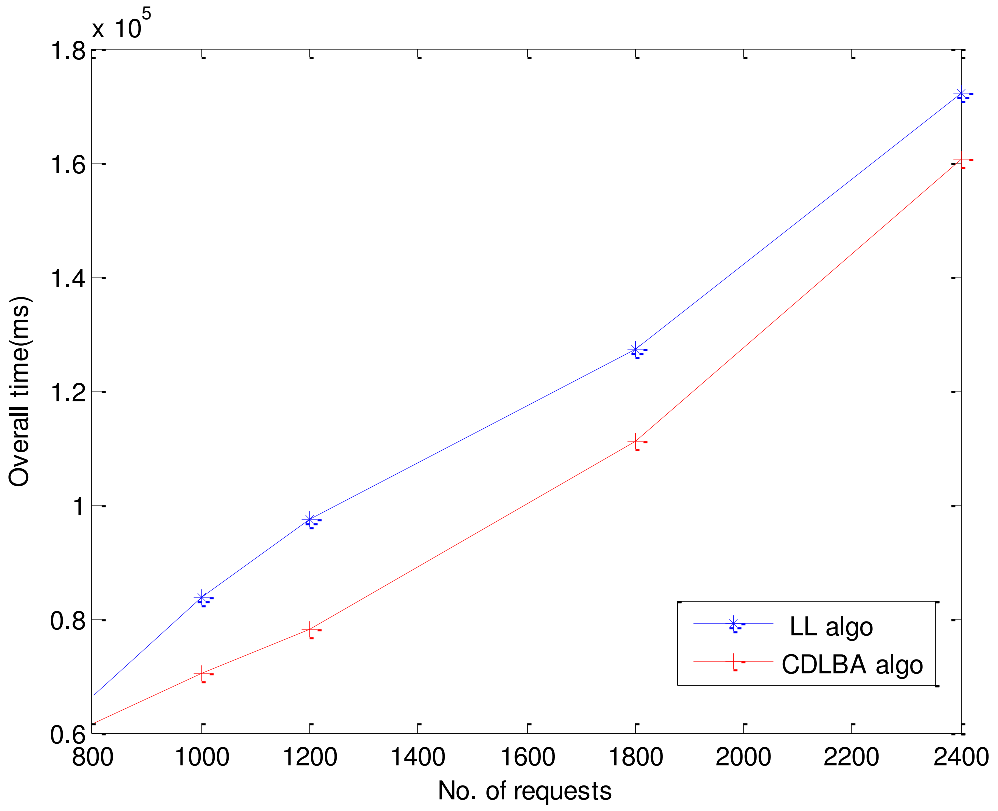


Figure 4.7: Overall response time for testbed-2

Here Figure 4.8 shows the comparison of our proposed CDLBA algorithm with least loaded algorithm with respect to server utilization. From figure 4.8, we can say that increasing in the number of requests increases the server utilization. The below Table 4.12 display the summary of results. Server utilization can be computed using Eq. (5.1) and (5.2).

Table 4.12: Comparison of average server utilization for testbed-2

Algorithm	Average utilization (%)				
	800	1000	1200	1800	2400
CDLBA	27.5	29.0	31.0	33.0	30.0
LL Algorithm	23.0	21.0	24.0	21.0	19.0

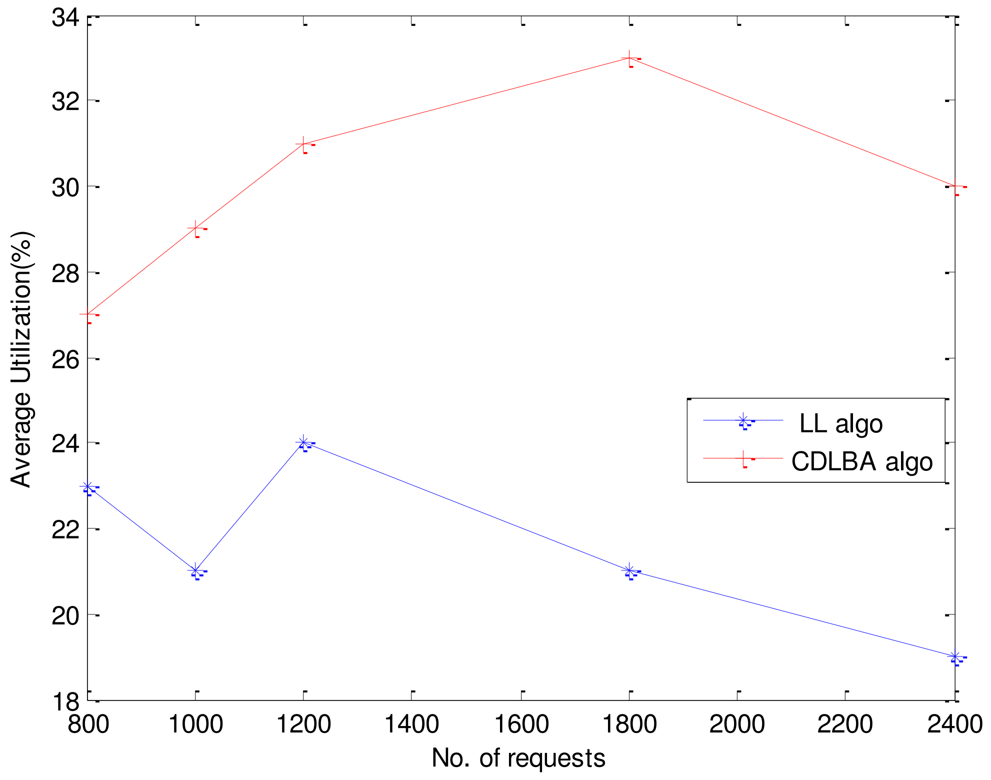


Figure 4.8: Server utilization for testbed-2

4.3. Simulation environment for DDLBA

We have simulated the performance of our second proposed algorithm DDLBA through using two test-beds. For DDLBA we have considered the different service time of server. So we have included one more parameter in the simulation environment. The configuration parameter of testbed-3 is shown below in Table 4.13.

Table 4.13: Configuration of testbed-3

	Server Id					
	S1	S2	S3	S4	S5	S6
Queue Length (request)	10	10	20	20	30	30
Service Rate(req./time)	5	7	9	11	13	15
Service time(s)	1.5	0.5	1	1.5	0.5	1

Table 4.14 shows the test case used for the simulation of proposed algorithm DDLBA. We have simulated proposed algorithm under five test cases which are shown in the Table 4.14.

Table 4.14: Test cases for DDLBA

Test case	No. of Requests	No. of Servers	Storage
1	600	6	500GB
2	800	6	500GB
3	900	6	500GB
4	1000	6	500GB
5	1200	6	500GB

4.3.1. Simulation results and discussion

We have compared the performance of our proposed DDLBA algorithm with least loaded algorithm in term of number of completes requests, number of delayed requests, overall response time of system and average server utilization. In DDLBA algorithm, we have utilized the service time of server for load distribution. Figure 4.9 shows the comparison of number of completed requests in DDLBA with least loaded algorithm. Figure 4.9show the number of client requests which are processed by server in their time i.e. these are the requests which never wait for the CPU availability. We have compared our results with least loaded algorithm. In least loaded algorithm when any server gets overloaded then load balancer selects the server of which request queue is least loaded without considering the service rate. But we have also considered the service rate as well service time parameter for the proposed algorithm. We have evaluated our results on different set of client requests. As shown in the Figure 4.9, our algorithm has outperformed as compare to least loaded algorithm. In all test cases, our algorithm has processed more number of client's request in a given time as compare to least loaded algorithm. Table 4.15 displays the summary of obtained results.

Table 4.15: Comparison of request completed for testbed-3

Algorithm	Client requests				
	600	800	900	1000	1200
DDLBA Algorithm	92	85	91	118	109
LL Algorithm	66	60	56	66	56

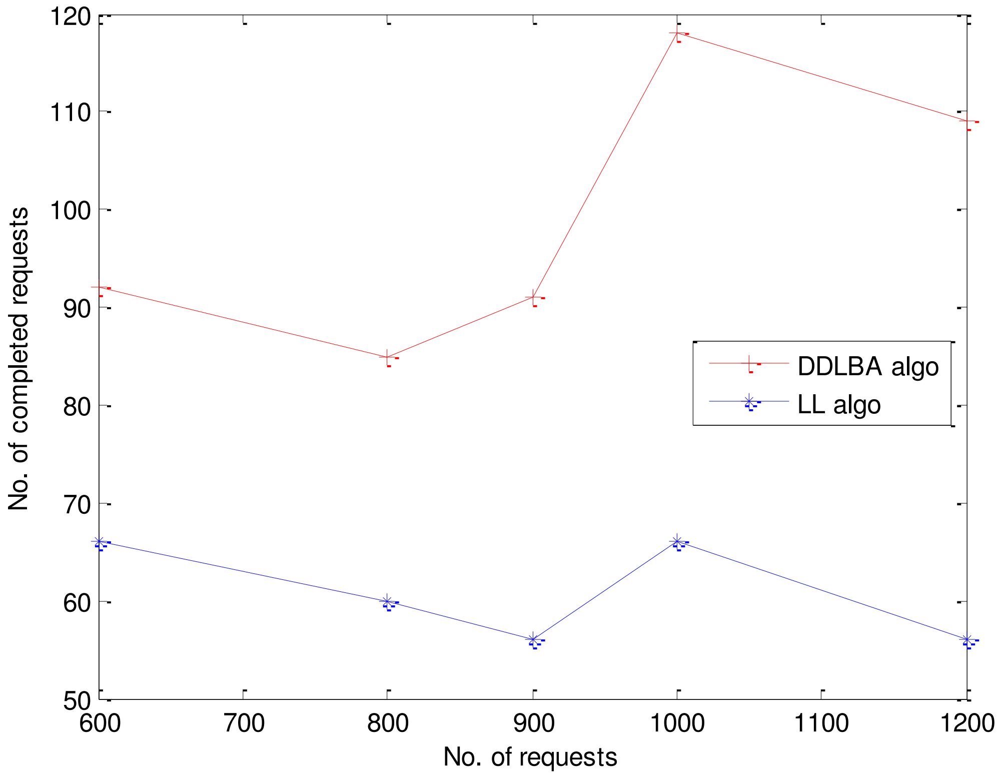


Figure 4.9: Comparison of completed requests in testbed-3

Here figure 4.10 shows the comparison of number of delayed request of DDLBA with least loaded algorithm. From figure 4.10 we can conclude that numbers of delayed requests are less in DDLBA as compare to least loaded algorithm in all test cases. This shows that our proposed algorithm are efficient than least loaded algorithm. The below table 4.16 displays the summary of obtained results.

Table 4.16: Comparison of delayed requests for testcase-3

Algorithm	Client requests				
	600	800	1000	1200	1400
DDLBA Algorithm	508	715	809	934	1144
LL algorithm	534	740	844	882	1095

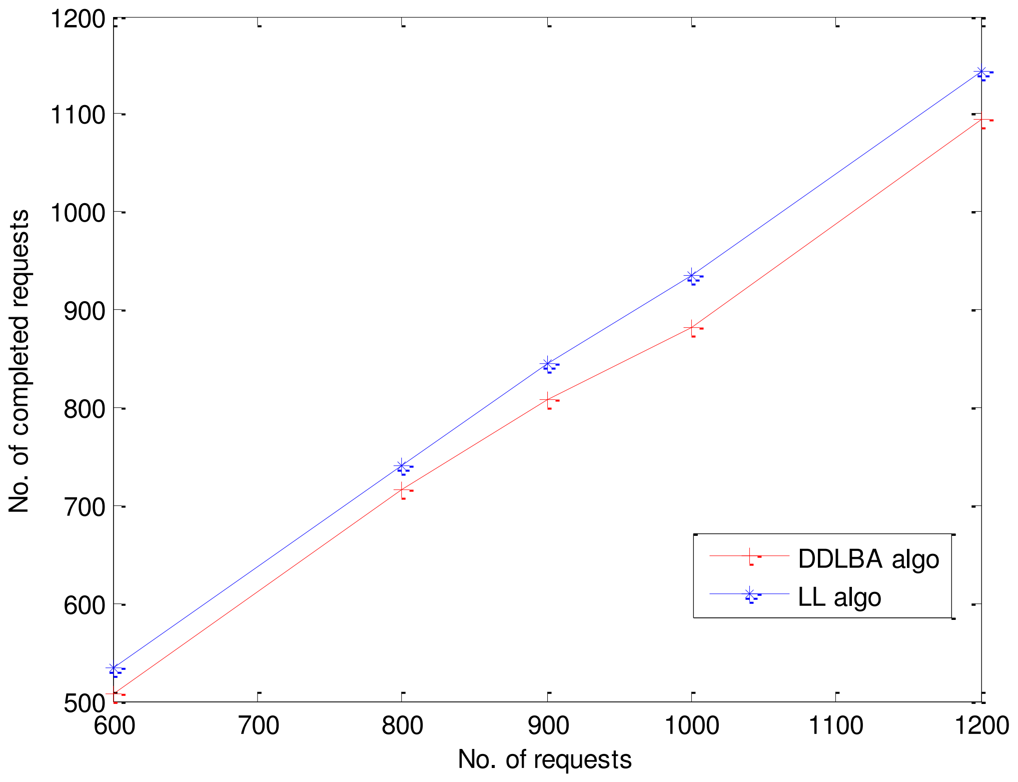


Figure 4.10: Comparison of delayed requests for test-bed-3

Figure 4.11 shows the comparison of proposed CDLBA algorithm with least loaded algorithm in term of overall response time. From figure 4.11 we can say that our algorithm outperform the least loaded algorithm. In all test cases, response time of our algorithm is less than least loaded algorithm. Table 4.17 displays the summary of obtained results.

Table 4.17: Comparison of response time for testbed-3

Algorithm	Overall Response time(ms)				
	600	800	1000	1200	1400
DDLBA Algorithm	46782	70307	86479	91812	109817
LL Algorithm	48690	72550	90333	102639	124580

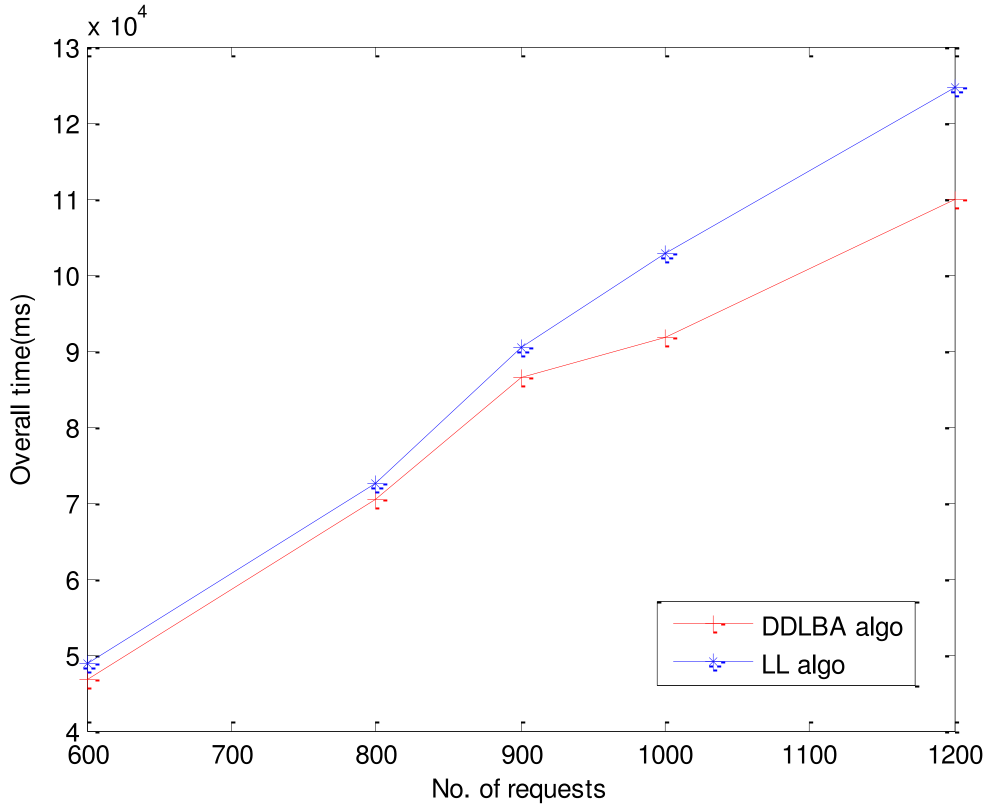


Figure 4.11: Overall response time for testbed-3

Figure 4.12 shows the comparison of our proposed DDLBA algorithm with least loaded algorithm with respect to server utilization. From figure 4.12, we can say that increasing in the number of requests also increases the server utilization whereas in least loaded server utilization decreases. Table 4.18 displays the summary of obtained results. Server utilization can be computed using Eq. (5.1) and (5.2).

Table 4.18: Comparison of average server utilization for testbed-3

Algorithm	Average utilization (%)				
	600	800	1000	1200	1400
DDLBA Algorithm	39.1	41.2	41.9	43.4	45.1
Least Loaded Algorithm	32.1	31.2	31.0	30.0	29.0

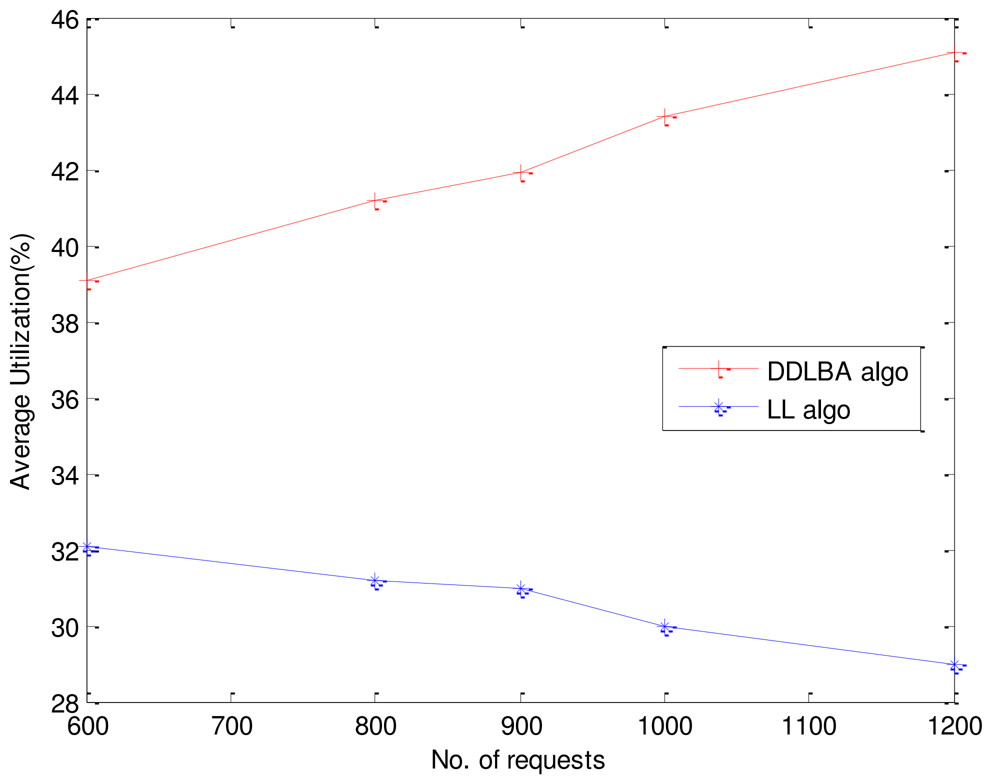


Figure 4.12: Server utilization for testbed-3

To regressively analyze the performance of DDLBA, we have simulated our proposed algorithm on second test-bed. The configuration parameters are displayed in Table 4.19.

Table 4.19: Configuration Parameter for testbed-4

	Server Id											
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
Queue length (request)	20	20	20	25	25	25	40	40	40	50	50	50
Service Rate (req./time)	5	5	5	7	7	7	10	10	10	15	15	15
Service time (sec)	1.5	0.5	1	1.5	0.5	1	1.5	0.5	1	1.5	0.5	1

We have rigorously analyzed the proposed algorithm. For that we have simulated proposed algorithm under different test case scenario. Following Table 4.20 depicts the test case scenario.

Table 4.20: Test cases for testbed-4

Test case	No. of Requests	No. of Servers	Storage
1	800	12	500GB
2	1000	12	500GB
3	1200	12	500GB
4	1800	12	500GB
5	2400	12	500GB

Figure 4.13 shows the comparison of number of completed requests in DDLBA with least loaded algorithm for testbed-3. Figure 4.13 shows the number of client requests which are processed by server in their time i.e. these are the requests which are not wait for the CPU availability. The below table 4.21 display the results in tabular format.

Table 4.21: Comparison of request completed for testbed-3

Algorithm	Client requests				
	800	1000	1200	1800	2400
DDLBA Algorithm	220	250	172	213	157
LL Algorithm	190	202	73	177	86

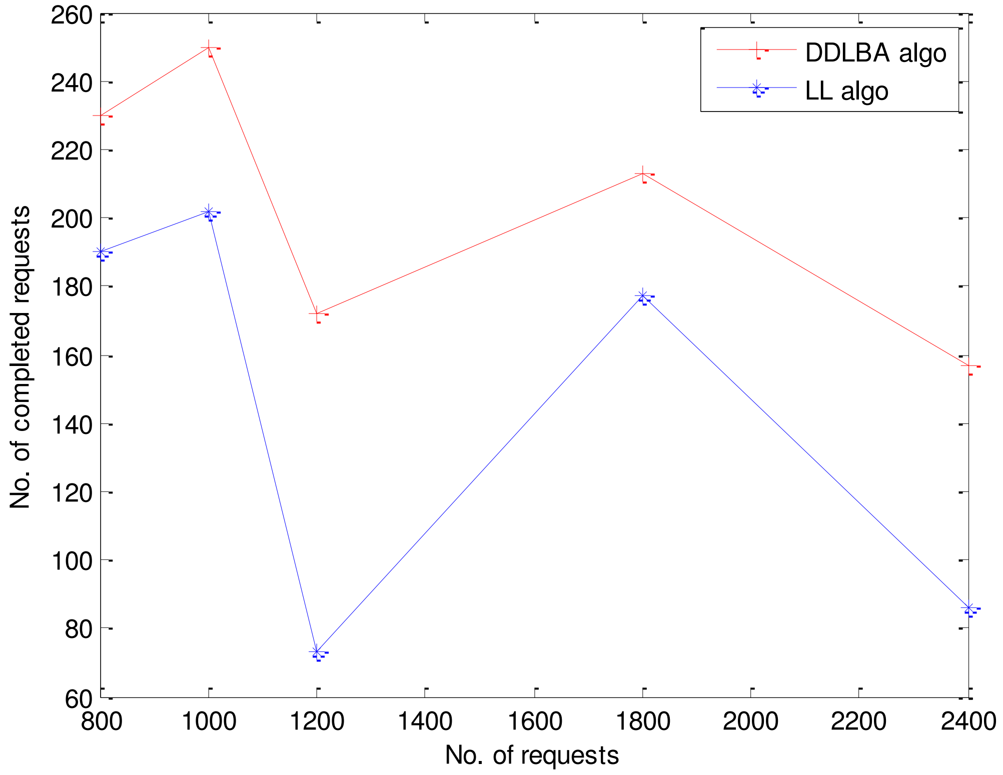


Fig.4.13: Comparison of completed requests for testbed-4

Figure 4.14 shows the comparison of number of delayed request of DDLBA with least loaded algorithm. From Figure 4.14 we can say that numbers of delayed requests are less in DDLBA as compare to least loaded algorithm in all test cases. This shows that our proposed algorithm are efficient than least loaded algorithm. Table 4.22 displays the summary of results.

Table 4.22 Comparison of number of postponed for testbed-4

Algorithm	Client requests				
	800	1000	1200	1800	2400
DDLBA Algorithm	570	750	1028	1587	2243
LL algorithm	590	798	1127	1623	2314

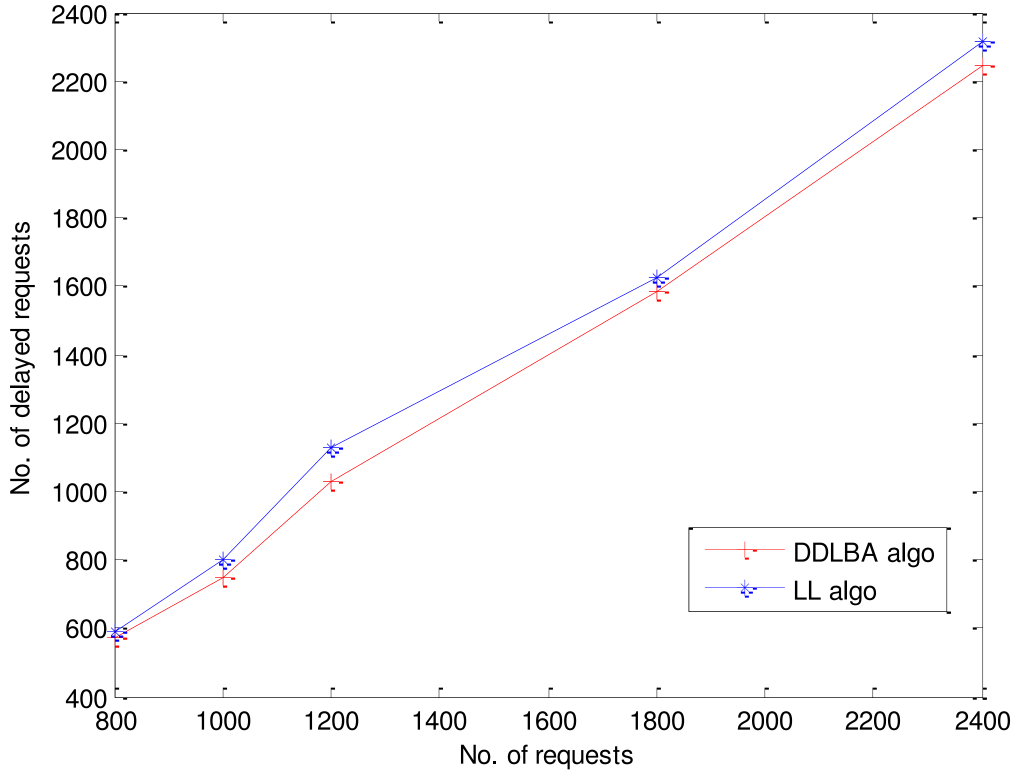


Figure 4.14: Comparison of delayed requests for testbed-4

Here figure 4.15 shows the comparison of our proposed DDLBA algorithm with least loaded algorithm in term of overall response time. From Figure 4.15 we can say that our algorithm has outperformed the least loaded algorithm. In all test cases, response time of our algorithm is less than least loaded algorithm. Table 4.23displays the summary of results.

Table 4.23: Comparison of response time for testbed-4

Algorithm	Overall Response time(ms)				
	800	1000	1200	1800	2400
DDLBA Algorithm	39082	40191	61350	84055	118395
LL Algorithm	40182	49418	70219	102248	138944

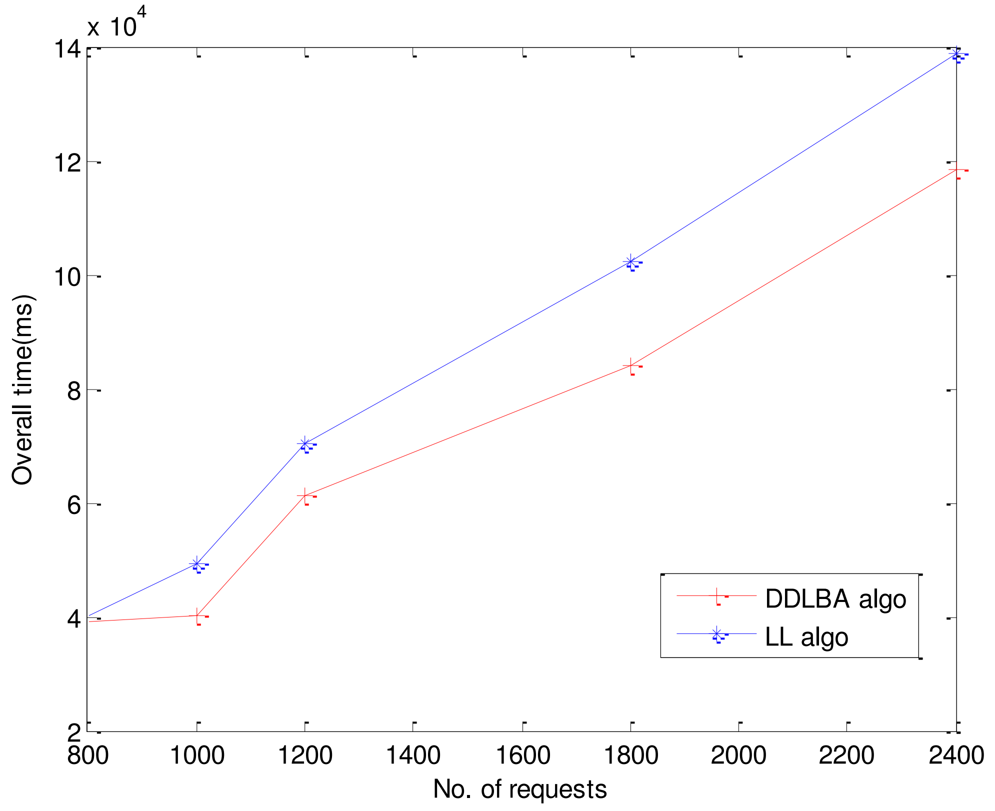


Figure 4.15: Overall response time for testbed-4

Here, Figure 4.16 shows the comparison of our proposed DDLBA algorithm with least loaded algorithm in term of server utilization. From figure 4.16, we can say that increasing the number of requests also increases the server utilization. Table 4.24 displays the results in tabular format. Server utilization can be computed using Eq. (5.1) and (5.2).

Table 4.24: Comparison of average server utilization for testbed-4

Algorithm	Average utilization (%)				
	800	1000	1200	1800	2400
DDLBA Algorithm	19.5	20.1	21.1	22.0	23.0
Least Loaded Algorithm	19.0	19.0	18.3	17.4	16.0

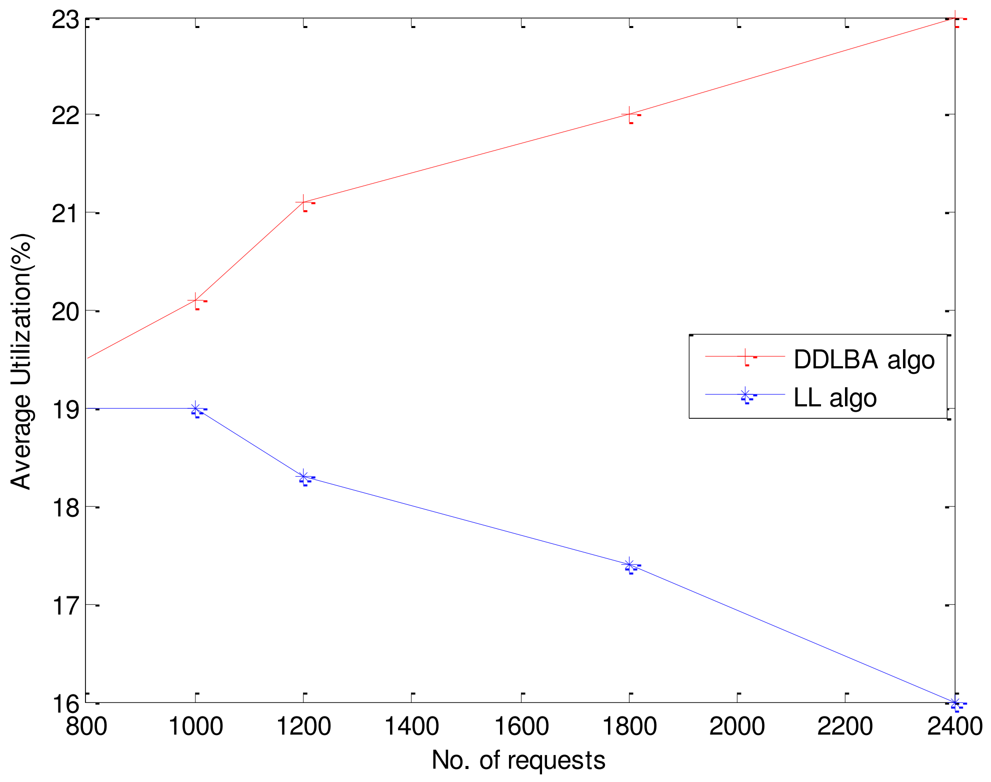


Figure 4.16: Comparison of server utilization for testbed-4

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this report, an effective distributed load balancing algorithm for cloud environment is presented. First a problem statement is formulated in cloud storage. Proposed load balancing algorithm designed and implemented to achieve global balancing in the cloud environment through adequately redistribute the extra load to the set of neighbors of the overloaded server to eliminating local server queue instability. In this report, two approaches are proposed which equally distribute the load among the storage servers to avoid load unbalancing. First proposed algorithm CDLBA balances the load among the storage server by effectively utilizing the server queue and service rate. CDLBA selects that server which has maximum free queue and maximum service rate. Second proposed algorithm DDLBA balances the load among storage server by utilizing the server queue, service rate, and service time for each client requests. DDLBA selects that server which can process the client requests within its deadline time. Through regress analysis of the performance of proposed approaches in term of number of completed client requests, number of delayed requests, overall system response time and average server utilization, we have concluded that both proposed algorithms reduce the overall response time, delayed requests and increase the completed requests as well as server utilization.

5.2 Future work

In this report, proposed work presents handling of load of storage servers in cloud environment. Our future work will be the authentic usage of our proposed methodology in a real framework, so that proposed load balancing to be utilized both as an evidence of true execution of the outcomes got through simulation and as a base for further research in the field of distributed storage. Since storage servers are interconnected through network, so the effect of network load, failure over network, storage capacity of server, etc. will be analyzed in context of proposed work.

REFERENCES

[1] Prabavathy, B., K. Priya, and Chitra Babu, "A load balancing algorithm for private cloud storage," *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*,Tiruchengode, India , 4-6 July 2013, pp. 1-6.

[2] You, Gae-won, Seung-won Hwang, and Navendu Jain, "Scalable load balancing in cluster storage systems," *Proceedings of the 12th International Middleware Conference*, 12 Dec. 2011, pp. 100-119.

[3] Mell, Peter, and Tim Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, 2011.

[4] Staten, James, Simon Yates, Frank E. Gillett, WalidSaleh, and Rachel A. Dines, "Is cloud computing ready for the enterprise," *Forrester Research,* March, 2008.

[5] Zhang, Qi, Lu Cheng, and Raouf Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, May 2010, vol. 1, Issue 1, pp 7-18.

[6] Rimal, Bhaskar Prasad, Eunmi Choi, and Ian Lumb, "A taxonomy, survey, and issues of cloud computing ecosystems," *Cloud Computing*, Springer London, chapter 2, 2010, pp 21-46.

[7] Kulkarni, Gurudatt, Rani Waghmare, RajnikantPalwe, VidyaWaykule, HemantBankar, and KundlikKoli, "Cloud storage architecture," In *7th International Conference on Telecommunication Systems, Services, and Applications (TSSA),* Bali, Oct 2012, pp 76-81.

[8] Wu, Jiyi, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu, "Cloud storage as the infrastructure of cloud computing," In *International Conference on Intelligent Computing and Cognitive Informatics*, Kuala Lumpur, June 2010, pp 380-383.

[9] R. Arokia Paul Rajan and S. Shanmugapriyaa, "Evolution of Cloud Storage as Cloud Computing Infrastructure Service," *IOSR Journal of Computer Engineering (IOSRJCE*), vol. 1, No.1, pp 38-45, 2012.

[10] Mishra, Ratan, and AnantJaiswal, "Ant colony optimization: A solution of load balancing in cloud," *International Journal of Web & Semantic Technology (IJWesT),* vol.3, No.2, pp 33-50, 2012.

[11] Kansal, Nidhi Jain, and Inderveer Chana, "Existing load balancing techniques in cloud computing: a systematic review," *Journal of Information Systems and Communication*, vol. 3, No.1, pp. 87-91, 2012.

[12] Alakeel, Ali M, "A guide to dynamic load balancing in distributed computer systems," *International Journal of Computer Science and Information Security,* vol. 10 No. 6, pp. 153-160, 2010.

[13] Rathore, Neeraj, and Inderveer Chana, "Load Balancing and Job Migration Techniques in Grid: A Survey of Recent Trends," *International Journal on Wireless Personal Communications*, vol. 79, Issue 3, pp. 2089-2125, 2014.

[14] Zhu, Yingwu, and Yiming Hu. "Efficient, proximity-aware load balancing for DHT-based P2P systems." *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 4, pp. 349-361, 2005.

[15] Zeng, Wenqiu, Ying Li, Jian Wu, Qingqing Zhong, and Qi Zhang, "Load Rebalancing in Large-Scale Distributed File System," In *1st International Conference on Information Science and Engineering (ICISE),* Nanjing, Dec. 2009, pp. 265 – 269.

[16] Hu, Jinhua, JianhuaGu, Guofei Sun, and Tianhai Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," In *3rd International Symposium on Parallel Architectures, Algorithms and Programming*, Dalian, Dec. 2010, pp. 89 – 96.

[17] Wang, Shu-Ching, Kuo-Qin Yan, Wen-Pin Liao, and Shun-Sheng Wang, "Towards a load balancing in a three-level cloud computing network," In *3rd International conference on Computer and Information Technology (ICCSIT),* Chengdu, July 2010, pp. 108-113.

[18] Tian, Wenhong, Yong Zhao, Yuanliang Zhong, Minxian Xu, and Chen Jing, "A dynamic and integrated load-balancing scheduling algorithm for Cloud datacenters," In *Proceeding of IEEE Cloud Computing and Intelligence Systems (CCIS)* , Chengdu, Sept. 2011, pp. 311-315.

[19] Lee, Rich, and Bingchiang Jeng, "Load-balancing tactics in cloud," In *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Beijing, Oct. 2011, pp. 447-454.

[20] Radojevic, Branko, and Mario Zagar, "Analysis of issues with load balancing algorithms in hosted (cloud) environments," *Proceeding of the 34th International Convention on MIPRO*, Opatija, Croatia, May 2011, pp. 416-420.

[21] Yamamoto, Hiroshi, and Daisuke Maruta, "Replication methods for load balancing on distributed storages in P2P networks," In *Proceedings of the 2005 Symposium on Applications and the Internet (SAINT'05)*, Feb 2005, pp.264 – 271.

[22] Randles, Martin, David Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," In *IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, Perth, WA, April 2010, pp. 551-556.

[23] Yao, Jing, and Ju-hou He, "Load balancing strategy of cloud computing based on artificial bee algorithm," In *8th International Conference on Computing Technology and Information Management (ICCTIM),* Seoul, April 2012, pp 185-189.

[24] Kwannetr, Umapom, Uthen Leeton, and Thanatchai Kulworawanichpong, "Optimal power flow using artificial bees algorithm," In *International Conference on Advances in Energy Engineering (ICAEE),* Beijing, June 2010, pp. 215-218.

[25] Liu, Hongzhi, LiqunGao, Xiangyong Kong, and Shuyan Zheng, "An improved artificial bee colony algorithm," In *3rd International Conference on Computer Research and Development (ICCRD)*, Shanghai, March 2011, pp. 174-177.

[26] Li, Kun, GaochaoXu, Guangyu Zhao, Yushuang Dong, and Dan Wang, "Cloud task scheduling based on load balancing ant colony optimization," In *Sixth Annual ChinaGrid Conference,* Liaoning, Aug. 2011, pp. 3-9.

[27] Dorigo, Marco, and Christian Blum, "Ant colony optimization theory: A survey," *Journal in Theoretical Computer Science*, vol. 344, Issues 2–3, pp. 243–278, 2005.

[28] Dorigo, Marco and Mauro Birattari, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, pp.28-39, 2006.

[29] Buyya, Rajkumar, Rajiv Ranjan, and Rodrigo N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit:

Challenges and opportunities," In *Proceedings of the 7th High Performance Computing and Simulation Conference*, Leipzig, Germany, 2009, pp. 1-11.

[30] Wickremasinghe, Bhathiya, Rodrigo N. Calheiros, and RajkumarBuyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," In *24th IEEE International Conference on Advanced Information Networking and Applications*, Melbourne, 2010, pp.446-452.

[31] Lu, Yilin, Jian Zhang, Shaochun Wu, and Shujuan Zhang, "A Hybrid Dynamic Load Balancing Approach for Cloud Storage," In *International Conference on Industrial Control and Electronics Engineering,* Xi'an, Aug. 2012, pp. 1332 – 1335.

[32] Hsiao, Hung-Chang, Hsueh-Yi Chung, Haiying Shen, and Yu-Chang Chao, "Load rebalancing for distributed file systems in clouds," *IEEE transactions on Parallel and Distributed Systems*, vol. 24, no. 5, pp. 951-962, 2013.

[33] Chung, Hsueh-Yi, Che-Wei Chang, Hung-Chang Hsiao, and Yu-Chang Chao, "The load rebalancing problem in distributed file systems," In *IEEE International Conference on Cluster Computing (CLUSTER),* Beijing , 24-28 Sept 2012, pp. 117 – 125.

[34] Manfredi, Sabato, Francesco Oliviero, and Simon Pietro Romano, "A distributed control law for load balancing in content delivery networks." *IEEE/ACM transaction on networking,*vol. 21, no. 1,pp. 55-68, 2013.

[35] Zeng, Zeng, and Bharadwaj Veeravalli, "Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks." *IEEE transaction on Computer*, vol. 55, no. 11, pp.1410–1422, 2006.

List of Publication

1. **Ravideep et al.**, "Load Balancing of Distributed Servers in Distributed File System", *7th ICT Innovations Conference*, Ohrid R. Macedonia, Oct. 2015.(*Communicated*).