DYNAMIC REPLICATION MANAGEMENT SCHEME FOR EFFECTIVE CLOUD STORAGE

MAY PHYO THU

UNIVERSITY OF COMPUTER STUDIES, YANGON

AUGUST, 2019

Dynamic Replication Management Scheme for Effective Cloud Storage

May Phyo Thu

University of Computer Studies, Yangon

A thesis submitted to the University of Computer Studies, Yangon in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

August, 2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

…..……………………………	………….......…………………………
Date	May Phyo Thu

ACKNOWLEDGEMENTS

First of all, I would like to thank the Minister, the Ministry of Education for full facilities during the Ph.D. course at the University of Computer Studies, Yangon.

Secondly, I would like to express very special thanks to Dr. Mie Mie Thet Thwin, Rector, the University of Computer Studies, Yangon, for allowing me to develop this thesis and giving me general guidance during the period of my study.

I would like to extend my special appreciation to Prof. Dr. Nwe Nwe Win, Vice-President, Myanmar Computer Federation (MCF) for the useful comments, sharing knowledge, giving advice, and insight which are invaluable to me.

I would also like to express my deepest gratitude to my supervisor, Dr. Khine Moe Nwe, Professor, Course Coordinator of the Ph.D. 9th batch, the University of Computer Studies, Yangon. She has provided not only helpful guidance but also a lot of inspiration, motivation and encouragement through all the stages of my research work. Her continuous motivation has been essential for the success of my work.

I would like to express my special gratitude to my co-supervisor, Dr. Kyar Nyo Aye, Lecturer, the University of Computer Studies, Yangon, for her patient supervision, encouragement and providing me with many ideas throughout the study of this thesis. I have learnt many things about research working together with her, and her support has been essential for the success of my work.

I would like to express my respectful gratitude to Daw Aye Aye Khine, Associate Professor, Head of English Department, the University of Computer Studies, Yangon, for her valuable supports throughout of my Ph.D course work and doing research and editing my thesis from the language point of view.

I would like to thank a lot to all my teachers for their mentoring, encouragement, and recommending the thesis during the Ph.D study. I also thank all my friends for their motivating encouragement, and many other topics, for being there when I needed them.

Last but not least, I am very grateful to my parents and hubby who specifically offered strong moral and physical support, care and kindness, during the year of my Ph.D study. Without their full support, my dissertation would not have been possible.

ABSTRACT

Replication plays an important role for storage system to improve data availability, throughput and response time for user and control storage cost. Due to different nature of data access pattern, data popularity is important in replication because of the unstable and unpredictable nature of popular files. In addition, the replica placement is important in consideration of system's performance. In data-parallel applications, data locality is a key issue and the consequence of this issue occurs the decrement of system’ performance. Therefore, this thesis proposes a dynamic replication management scheme for effective cloud storage (ECS). The system contains two portions; replica allocation and replica placement. In the first portion, replica allocation, popularity is taken into account by analyzing the changes in data access pattern. Second for replica placement, replicas are placed and performed on dedicated assigned nodes in order to enhance data locality. The proposed placement algorithm is able to avoid the overloaded problem of nodes and the more effective storage utilization by considering the load of nodes; that is, disk utilization, CPU utilization and adjustable disk bandwidth.

The proposed system is implemented as cloud storage system by using open-source CloudSim simulator. The aims of the proposed scheme are (i) to reduce unnecessary replication cost for unpopular data (ii) to achieve load balancing in data placement and (iii) to increase replica number for popular data. The analysis results demonstrated that the proposed scheme can adapt the degree of replication based on data popularity, save storage cost for unpopular files and achieve more load balancing than existing replication strategy such as Latest Access Largest Weight (LALW) algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1
INTRODUCTION

Cloud computing is an internet-based distributed computing technology where services are provided to users on demand as pay per use basics. Storage, programs and application-development platforms can be accessed by users through the Internet with the aid of offering services of cloud computing providers. There have three service models in cloud computing. They are platform-as-a-service, infrastructure-as-a-service, and software-as-a-service. The access to these services is performed by users with pay-per-usage model in that the infrastructure provider provides guarantees with customized service level agreements (SLAs).

Configures, de-provisions servers, reconfigures and provisions can be dynamically provided as needed by the cloud computing platform. The accessible applications can also be extended through the internet. These applications apply powerful servers and large data centers that host web services and applications. Hence, cloud storage systems play an important role for cloud computing infrastructures.

Data storage is a backend foundation of cloud computing where large volume of data is needed to store. Storing extremely large volumes of information in an effective manner is an essential issue in cloud computing. High availability and high-performance accessible data storage are provided by high capacity electronic data storage devices through industry standard interface. However, they are costly to purchase.

Popular cloud storage systems as like hadoop distributed file system (HDFS) [3] and google file system (GFS) [29] are employed for huge amount of data for storage, processing and management in today data center. With the increasing amount of data stored and processed in cloud storage system, storage components are expanded in scale in cloud data center. However, as cloud storage system can be installed on commodity servers where components failures are common, the efficient storage, processing and management of data on cloud storage have raised significant concerns especially for the maintenance of the certain guarantee level of data. Among the cloud storage systems, as HDFS is the most widely used and is an open source, this study is focused on HDFS.

The system is implemented with hadoop distributed file system by analyzing replication management. HDFS is suitable for the implementation and evaluation of the proposed replication management system. In this proposed replication management system, the system provides the suitable number of data replicas as well as increasing data locality and load balancing among the storage server nodes. In this thesis, therefore, the simulated cloud storage data centers are also implemented with the associated replication management system.

1.1 Motivation and Main Issues

Nowadays, the significant technology trend is cloud computing and it reforms information technology (IT) marketplace and IT processes. In the past years, Amazon, Microsoft and Google companies have been constructing huge amount of data centers. This data centers are constructed by spanning geographic and administrative domains. Networking, storage and CPU can be provisioned by these data centers at considerable low prices by leveraging economics of scale that provides the move by many institutions to host their services in the cloud.

At the same time, keen interest of this cloud provider has been ignited by the advent of new technologies and the economic situation. Economics of scale are delivered by cloud storage providers with no different storage space for satisfying the requirements of users, at the savings of cost for their storage. High performance storage servers are deployed by cloud service providers as data center that is very reliable and expensive. But, information storage and storage management with restricted financial plan is a principal factor for large enterprises and small business.

In fact, replication is an essential corner stone in data storage not only for cloud computing but also for traditional storage systems because it can relatively impact the performance of cloud storage in terms of storage cost, network usage, response time, etc. In addition, as the cloud environment has less stable and highly skewed data access pattern, different data file may have different properties of popularity. Therefore, maintaining static number of replicas in cloud storage for every data file would be inefficient for storage cost and data availability. As a consequence, the determination of the optimum number of replicas and the suitable nodes for replicas has become a key issue in the cloud computing.

Cluster computing systems, featuring fault-tolerance data storage distribution have been widely applied for data-intensive applications. Huge clusters contain tens of thousands of devices and that are designed for searching and web indexing; small and medium sized clusters are designed for corporate data warehousing and business analytics. The more closer placement of data with computing node is a common routine of storage systems, also known as the data locality issue. These systems apply static data replication for (a) improvement of data locality by placing a task and its data at the same local storage (b) achievement of load balancing with distribution of work among the replicas (c) ensuring fault tolerance and data availability in system failure.

Data locality is one of the key issues in considering the performance of Hadoop. In order to provide data locality, Hadoop performs collocation of data with assigned nodes. However, as Hadoop randomly places data to nodes, there is a condition, that is, when assigned node load data blocks from different node that stores the same data block. Therefore, data locality problem has occurred. Data locality is interval between data block and the assigned node. This minimizes network congestion and increases the overall throughput of the system. The types of data locality are node locality, rack locality and rack-off locality. The greater node locality, the more throughput of the system.

Static replication is used in implementation of Hadoop. Static replication is not good because data access pattern always changes every time. Therefore, file popularity factor is necessary to be considered in replication. File popularity can be estimated from data access pattern. The consideration of file popularity in replication results in efficient storage because it avoids replicating unnecessary replicas.

To manage replication in cloud environment, there are two main problems that impact on system performance. They are:

(i) Replica Allocation Problem: The replication degree of files should be able to adapt the changing pattern of data access.
(ii) Replica Placement Problem: Placement of various concurrent accessed data blocks into various data nodes for the contention reduction on a particular node.

To address these challenges, this thesis proposes the dynamic replication method in order to support better locality in HDFS. It consists of two major parts. First part is the changes of file popularity which are computed by analyzing data access pattern with first order differential equation. For this part, Hadoop distributed file system is used as a framework of open source storage cluster. The second part is the calculation of replicas for each data and the replicas are placed on nodes in order to improve data locality.

1.2 Objectives of the Thesis

The major objectives of the thesis are as follows:

- To overcome the problems of static replication in cloud storage
- To reduce storage cost by replicating relevant file with suitable replication factor in accordance with data access frequency
- To achieve the increased data locality and load balancing based on the storage utilization, disk bandwidth and CPU utilization of nodes
- To apply the replica allocation algorithm and replica placement algorithm in simulated cloud environment by adjusting the replica degree based on the rate of change of file popularity
- To evaluate and analyze the performance of replication management system in terms of the number of created replicas, storage cost and disk utilization.

1.3 Contributions of the Thesis

The thesis contributes to the field of cloud storage in several ways. The Major contribution is the implementation of effective storage cluster together with enhanced replication policy which has been published in [P1][P2][P3][P4][P5]. This contribution is divided as follows:

- The rate of change of file popularity in timeslots is analyzed by applying first order differential equation.
- Determination of the decrement and increment of the number of replicas for each file is computed.
- While the replicas are placed into nodes, the load of nodes such as disk utilization, CPU utilization and bandwidth utilization are considered.
- The predefined threshold is used to compute the overloaded condition of cluster.
- If the overloaded condition of that assigned nodes occurs, the proposed replica replacement algorithm will be used.
- This proposed replacement algorithm considers not only early used blocks but also the access frequencies for blocks.

The last but the most important is the contribution of replication management over distributed cloud data centers which is published in [P6]. It consists of the following:

- Simulation of prototype cloud data centers in java environment
- Application of the replication algorithms suitable for distributed cloud data centers
- Utilization of Yahoo Audit log data set to model data access pattern for the proposed algorithms
- Comparison of proposed replication algorithms, namely *Replica Allocation Algorithm* and *Replica Placement Algorithm*, with other existing replication algorithms
- Measuring the performance of the system in utilization, storage cost model and the number of created data replicas with other existing replication algorithm LALW.

1.4 System Overview

The basic idea of replication is based on the different replication degree per data file. Keeping the fixed number of replicas causes wasteful storage for unpopular data and inefficiency for popular data. Also, maintaining too much replicas than current access count for a file does not always guarantee better locality for all blocks. The objective of this system is to propose a replication strategy in order to achieve the improved data locality by more replicas for popular data while maintaining less replicas for unpopular data.

In this thesis, a replication management strategy is proposed for cloud storage. The system contains two portions; replica allocation and replica placement. In the first portion, replica allocation, popularity is taken into account by analyzing the changes in data access pattern. At this portion, first order differential equation is applied to compute the rate of change of file popularity. After that, the number of replicas for each file is defined using changes of file popularity that is the outcome of the first stage. Initially, existing replicas will be assumed as 3 like the default replica of HDFS. If the rate of change of change of file popularity (k) is less than 0.0, then existing replicas is decreased by 1. If k is greater than 0.0, then existing replicas is increased by 1. If k is equal to 0.0, then existing replicas is unvaried. Otherwise, if it is a new file, then existing replicas is determined 3 like the default replica of HDFS.

Second for replica placement, replicas are placed and performed on dedicated assigned nodes in order to enhance data locality. The proposed placement algorithm is able to avoid the overloaded problem of nodes by considering the load of nodes; i.e, disk utilization, CPU utilization and adjustable disk bandwidth while loading into assigned nodes. That replica is loaded if the load of assigned node is less than predefined threshold of its cluster. Otherwise, the replica block is needed to replace with existing block at assigned node.

1.5 Organization of the Thesis

This thesis is organized as follows:

In this chapter, introduction to cloud computing, cloud storage infrastructure, replication management, motivations and contributions are presented with the objective of the thesis.

And, the overview of cloud computing and the state-of-the-art technology of cloud storage are reviewed in chapter 2.

In chapter 3, the literature, related work with different replication policies based on data popularity and data locality which are currently applied in traditional distributed system and/or cloud computing are looked in depth into.

Chapter 4 describes a proposed replication strategy in order to achieve the improved data locality by more replicas for popular data while maintaining less replicas for unpopular data.

Chapter 5 proposes the replication algorithms together with comparisons of the existing approaches. The comparison of performance and evaluation of the proposed system are also discussed in this chapter.

The final chapter of the thesis provides a conclusion and outlook on future work.

CHAPTER 2
CLOUD STORAGE ARCHITECTURES

Cloud computing has become popular as a cost-effective solution with reliable computing resources without possessing any infrastructure. Today, benefits of cloud computing for the technological advancements are communications, storage and computing. The basic idea is to take advantage of economies of scale so that IT services could be provided on demand with a decentralized infrastructure. Cloud storage is an interface of cloud computing where the storage can be managed on demand. Cloud storage infrastructures propose new architectures that provide various services over a potentially large set of users and geographically distributed storage capacity. There are many research areas in cloud storage. Many of them are file system-based storage systems such as HDFS [3], GFS [29] and some researchers such as VBS [28], Amazon EBS used block storage. Furthermore, cluster storage architecture like [15][59] and peer to peer architecture had also been implemented for cloud storage. In this chapter, cloud storage technology, different storage systems for cloud computing and challenges of cloud storage systems are identified.

2.1 Cloud Computing

This section aims to introduce cloud computing and its essential characteristics. Cloud computing is the scalable distributed computing environment in which a large set of virtualized computing resource, different infrastructures, various development platforms and useful software are delivered to customers as a service with pay per usage usually over the Internet. Cloud computing is defined by the U.S. National Institute of Standards and Technology (NIST) as follows: Cloud computing provides a model for ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [58].

2.1.1 The Essential Characteristics of Cloud Computing

The essential characteristics of the cloud computing are:

- On-demand Self Service: Computing abilities such as emails, applications and network storage are provided to consumers without human interaction with service provider. Service providers providing these services contain Microsoft, IBM, Amazon Web Services (AWS), Google and Salesforce.com.
- Broad Network Access: Computing Capabilities are available over the network and they can be accessed through standard mechanisms that raise the use of nonhomogeneous client devices such as mobile phones, laptops, tablets and workstations.
- Resource Pooling: Cloud computing resources such as memory, storage, processing, network bandwidth and virtual machines are pooled to provide services to various consumers by a multi-tenant model with dynamic allocation and deallocation of various virtual and physical resources depending on demand of user. They provide location independence so that consumer does not need to know on exact location of computing resources but consumer may specify location with high level of abstraction like country, city or data center.
- Rapid Elasticity: Cloud computing infrastructures can provide flexible computing platform with rapidly and elastically and in some cases, which can broaden or decrease in line with business demand, the available abilities for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
- Measured Service: The resource usage of cloud computing can be controlled, measured and reported providing transparency for both the provider and consumer of the utilized service. It uses a metering capability to various service types (e.g., processing, storage and network bandwidth) which enables to control and optimize resource use. These services are charged as pay per usage metrics.
- Multi Tenacity: Users might utilize public cloud provider’s service offerings or actually be from the same organization, such as different business units rather than different organizational entities, but would still share infrastructure [61].

2.2 Open-Source Cloud Systems

In this section, some of the current features of Cloud computing technologies are described. Eucalyptus, OpenNebula, OpenStack and Nimbus are major open-source Cloud computing software platforms.

2.2.1 Eucalyptus

Eucalyptus cloud infrastructures such as Simple Storage Service (S3), Amazon EC2, and PaaS offerings, such as Google App Engine, crow higher usability and lower cost [9]. However, it is used to build private cloud infrastructure. Private cloud would be preferred over a public cloud, which might be characterized by special security requirements or the need to store critical company data. It can be imaginable to build up an internal data mirror (RAID-0) in order to increase availability of cloud infrastructure.

Eucalyptus is an open source software platform for implementing Infrastructure as a Service (IaaS) in a private or hybrid cloud computing environment [40]. It converges together existing virtualized infrastructure to create cloud resources for infrastructure as a service, network as a service and storage as a service. Eucalyptus [40] is abbreviation for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems, and it was initiated at the University of California in Santa Barbara (UCSB). Eucalyptus allows setting up and operating an independent IaaS cloud infrastructure. It is compatible with Amazon EC2, S3, and Elastic Block Storage (EBS) [63]. Unlike Amazon EC2, which exclusively uses Xen for virtualization, Eucalyptus can cooperate with Xen and KVM. A prerequisite for using KVM is a CPU that supports hardware virtualization, i.e. AMD-V (Pacifica) or Intel VT-x (Vander pool). The commercially available Enterprise Version offered by Eucalyptus Systems supports VMware vSphere/ESX/ESXi. It is not planned to integrate VMware support into the free Eucalyptus version.

The features of Eucalyptus are:

- It supports both Linux and Windows virtual machines (VMs).
- Its application program interface- (API) is compatible with Amazon EC2.
- It is compatible with Amazon Web Services (AWS) and Simple Storage Service (S3).

- It can work with multiple hypervisors including VMware, Xen and KVM.
- Installation and deployment can be performed from source code or DEB and Red-hat Package Manager (RPM).
- Internal processes communications are secured through Simple Object Access Protocol (SOAP) and WS-Security.
- Multiple clusters can be virtualized as a single cloud.
- Administrative features such as user and group management and reports can be done.

The components of Eucalyptus are as follows:

- Cloud Controller (CLC): The controller that manages virtual resources like servers, network and storage. It is at the highest level in hierarchy. It is a Java program with web interface for outside world. It can do resource scheduling as well as system accounting. There is only one cloud controller for each cloud. It can provide authentication, accounting, reporting and quota management in cloud.
- Walrus: This is another Java program in Eucalyptus that is equivalent to AWS S3 storage. It provides persistent storage. It also contains images, volumes and snapshots similar to AWS. There is only one Walrus in a cloud.
- Cluster Controller (CC): It is a C program that is the front end for a Eucalyptus cloud cluster. It can communicate with Storage controller and Node controller. It performs management of the instance operation in cloud.
- Storage Controller (SC): It is a Java program equivalent to EBS in AWS. It can interface with Cluster Controller and Node Controller to manage persistent data via Walrus.
- Node Controller (NC): It is a C program that can host a virtual machine instance. It is at the lowest level in Eucalyptus cloud. It downloads images from Walrus and creates an instance for computing requirements in cloud.
- VMWare Broker: It is an optional component and provides AWS compatible interface to VMWare environment.

A general architecture of the Eucalyptus is shown in Figure 2.1.

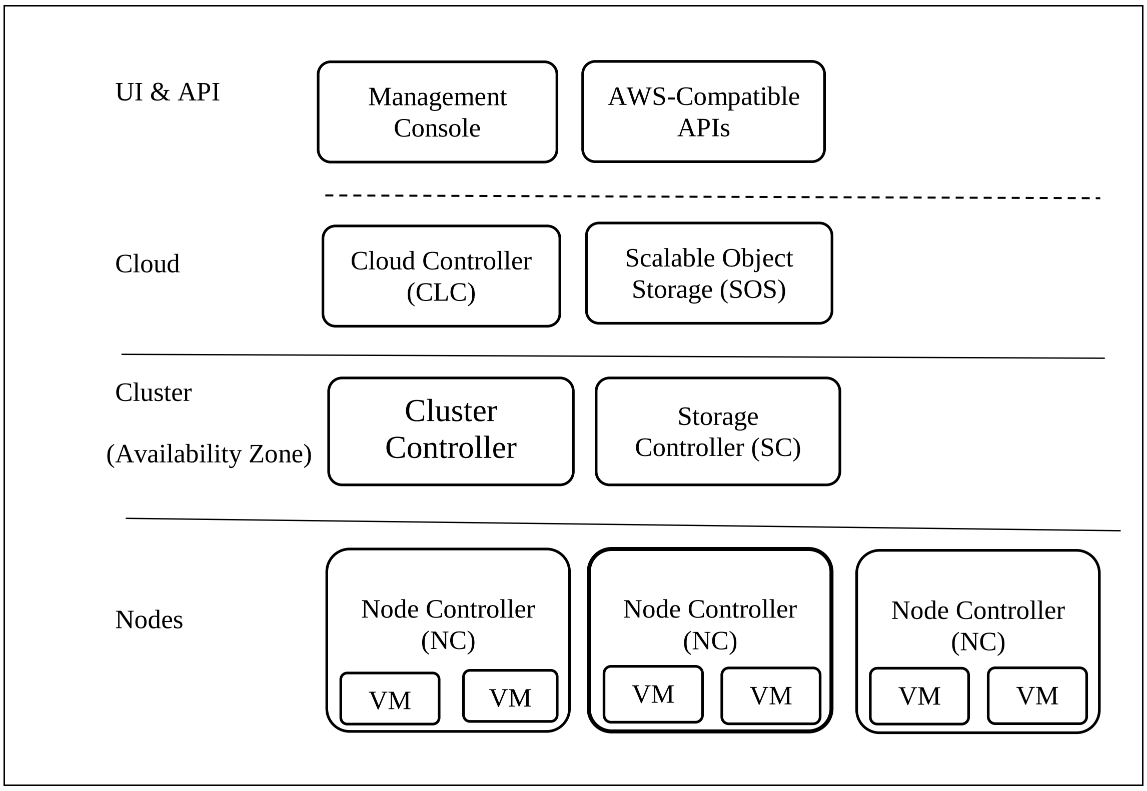

Figure 2.1 General Architecture of Eucalyptus

2.2.2 OpenNebula

OpenNebula is an open-source cloud computing platform that provides the management of virtual machines with excellent performance and scalability, and creation of virtualized data centers on various types of clouds with the most advanced functionality [9]. It is the combination of virtualization technologies with advanced features in order to provide multi-tenancy, automated provisioning and elasticity. A virtual network manager performs the mapping operation of virtual and physical networks. It is freedom of vendor, platform- and API-agnostic. OpenNebula uses the Xen Hypervisor, KVM, and VMware vSphere approaches.

Different from Eucalyptus, it provides the allowance of working instances between the connected nodes. It only supports basically on EC2 Query APIs and the EC2 Simple Object Access Protocol (SOAP). It may perform the retrieval of list of instances and images and operation of instances. Moreover, it can be used to control the resources of Amazon EC2. As node grouping capability, high performance computing is provided as a service (HPCaaS). It has no compatible storage service for EBS API and S3 like Eucalyptus and Nimbus.

2.2.3 OpenStack

Rackspace and NASA jointly developed an open source project called OpenStack [37] in summer 2010. It was managed by the OpenStack foundation in 2016 which is a non-profit corporate entity to advance OpenStack software and its community. Many renowned companies such as Dell, Intel, Cloud.com and AMD support this project. OpenStack is an open-source and free software platform for cloud computing, and is used as a infrastructure-as-a-service (IaaS), whereby other resources and virtual servers are provided available to users. It contains interrelated components to manage diverse, multi-vendor hardware pools of processing, networking resources and storage throughout a data center. Users control it through command-line tools, a web-based dashboard and RESTful web services. Basically, it provides Object Storage and the Compute components. The object storage provides scalable and redundant storage space available and the compute service allows to manage virtual servers. Microsoft announced that adaption of the OpenStack software to their hyper-V virtualization technology will be provided. The aim is to have the ability to use open source programs and windows together in cloud systems.

2.2.4 Nimbus

Nimbus [42] is an open source private cloud IaaS solution launched by the Globus Alliance. It mainly provides a configuration and deployment of virtual machines on remote resources in order to create the suitable environment for the users' requirements. It contains two main products:

- The infrastructure is an open source S3/EC2-compatible IaaS solution featured beneficially to interests of scientific community such as support for proxy credentials, auto-configuring clusters, best-effort allocations and batch schedulers, etc.
- It is an integrated set of solutions for a multi-cloud environment that enables the simplification and automation of the work with infrastructure clouds (scaling, management and deployment of cloud resources) for scientific users.

It supports the Xen hypervisor and KVM virtualization solutions. It has compatibility with S3 REST API clients, Amazon's Network Protocols via EC2 based clients, also REST API, and SOAP API which have been implemented in Nimbus. Moreover, it gives support for fast propagation, X509 credentials, compartmentalized dependencies and multiple protocols. It featured flexible group, workspaces and user management, per-client usage tracking and request authorization and authentication. Above version 2.4, it contains the Cumulus storage service which has compatible interface with the S3 REST API. Cumulus is used for storage of images. Installation and deployment of Cumulus may be done as a standalone service without Nimbus. It has no an EBS-compatible storage service [89].

2.2.5 AppScale

AppScale is an open-source cloud computing platform that supports the execution of applications developed by Google App Engine. AppScale enables multiple App Engine applications to be uploaded to a cloud. AppScale framework is an implementation of platform as a service [46]. It sits over any virtualization-supported infrastructure to host and operate applications created in the Google App Engine. It supports the deployment of multiple applications over the cloud and supports deployment for major vendors operating as infrastructure as a service. Before being commercially released, AppScale framework was developed and maintained as a university research project at the Rapid Access Computing Environment Lab at the University of Santa Barbara. It is Google App Engine (GAE)-compatible API and operates GAE applications over other cloud infrastructures or on-premise no need for modification.

Its possibility is to run and test Google App Engine-compatible applications within a public cloud (EC2) or a private cloud (Eucalyptus). Moreover, the implementation can be performed directly on the Xen hypervisor, without interposing an IaaS. AppScale is written in python, java and go for the google app engine and its execution is infrastructure-independent platforms. The operation can be implemented as a virtual machine over any virtualized infrastructure, including Eucalyptus and Amazon EC2 private clouds. It also supports the integration of applications developed for the google app Engine [47]. Moreover, it supports other APIs such as the message passing interface and mapreduce. It provides complete liberty in selection of private, public and hybrid cloud infrastructure. It also supports many different data stores, such as MongoDB, MySQL Cluster and Memcache DB. The general architecture of the AppScale is shown in Figure 2.2.

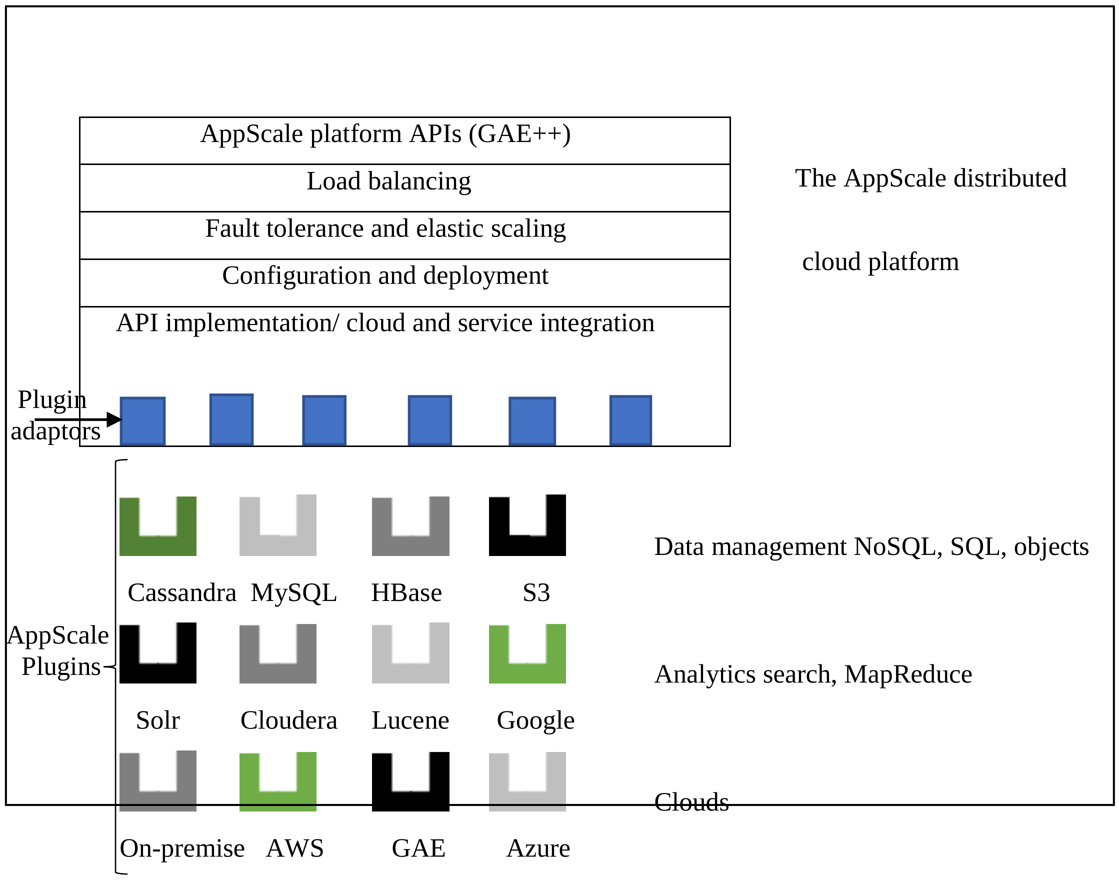
AppScale platform APIs (GAE++)
Load balancing
Fault tolerance and elastic scaling
Configuration and deployment
API implementation/ cloud and service integration
The AppScale distributed cloud platform
Plugin adaptors
AppScale Plugins
Cassandra
MySQL
HBase
S3
Data management NoSQL, SQL, objects
Solr
Cloudera
Lucene
Google
Analytics search, MapReduce
On-premise
AWS
GAE
Azure
Clouds

Figure 2.2 General Architecture of AppScale

2.2.6 Apache CloudStack

Apache CloudStack [85] is a free open source cloud computing software enabling the management of large virtual machine networks. It provides an on-premises cloud and a part of hybrid cloud to many companies and offers public cloud services to many service providers. Its features contain compute orchestration, user and account management, resource accounting, a first-class User Interface (UI), Network-as-a-Service and more. It provides the most popular hypervisors such as Citrix XenServer, Oracle VM server, KVM, VMware, Microsoft Hyper-V and Xen Cloud Platform (XCP). Management of cloud can be performed easily through web interface, a full-featured RESTful API and command line tools. Moreover, it provides an S3 and AWS EC2-compatible API for organizations wishing the deployment of hybrid clouds. It scales up and down depending on software, business, hardware and virtual machines in network.

2.3 Cloud Storage

Cloud storage is the integrated product of virtualization technologies and distributed storage. It provides as storage devices on private network or the internet and as data storage services on remote hosted servers. It will have ability to offer storage service with more security and reliability and lower cost. The advantage is it enables users at any time access data. There are many cloud storage providers, such as Microsoft, HP, IBM, Amazon, Google and iCloud, etc. Amazon S3 supports a simple web services interface for retrieving and storing any data, from anywhere on the cloud, and at any time. It supports any developer for accessing to the more reliable, fast, scalable, inexpensive and secure infrastructure that is used by Amazon for running its own global network. The service is intended in order to maximize scalability for developers.

2.4 Design Assumptions and Characteristics of Cloud Storage

When cloud storage is analyzed and reviewed, many critical points which are particularly different with traditional data storage are found. They are:

2.4.1 Scaling out the hardware

Scaling horizontally (or scale out) is the increment of nodes to a system, such as addition of a new computer is performed to a distributed software application. In case of scaling up, the machines specification is consolidated to obtain higher performance by addition of resources to a single node in a system, which contains the addition of memory or CPUs to a single computer. The cost performance in scale up usually is not so effective. To get the double speed up, the cost will be 10 times more than actual one.

Instead of using costly scaling up strategy, the cloud computing data centers usually applied scaling out technique by taking the advantages of cost saving. In case of higher number of data access to web server, one web server system can be easily scaled out by adding more web servers which is more scalable and enable to obtain better performance.

2.4.2 Highly Distributed Cluster

The cloud storage cluster may contain more than hundreds of storage devices that are built with inexpensive commodity parts. These storage nodes are geographically distributed and the accessing is made from large number of client machines. Google [71] deploys large storage cluster for the processing and generation of data. Hundreds of terabytes of storage are provided by the largest cluster to date across thousands of disks over a thousand machines, and the accessing is made concurrently by hundreds of clients.

2.4.3 Big Data Set

In accessing the collection of shared data, data sets can be used as a unit in the cloud environment. It is only single large file that has specific format or collection of files at physical disk. Most file have large sizes in cloud data storage when they are compared with traditional standards. In cloud storage, most files are multi-GB-sized files. The file includes many application objects such as web documents. In cloud computing, most data storage system are especially designed for large-sized files. As a consequence, management of approximation of billions of KB-sized files is inefficient even if the file system could only support it.

2.4.4 Immutable Objects

As a web-based system, mutation of files in the cloud are performed by appendage of new data with no need for overwriting existing data. Writing randomly within a file are impractically existent. The files are sequential and read only once they are written. These characteristics are shared by a variety of data. Those files may be intermediate results formed by computation at one device and processing at another device, either later in time or simultaneously or may be data streams continuously generation from running applications. Otherwise, they can be archival data. Because of such appending feature, it becomes the focal point of atomicity guarantees and performance optimization.

2.4.5 Multi-sharing

In dedication of computing facilities to a single owner or user, cloud computing is based on a business model in that sharing of resources is performed unlike previous computing models. In order to share the resources in cloud computing, multi-tenancy system is deployed that permits sharing of infrastructure to several users, without awareness of it to users and without compromising the security and privacy of each customer’s data. It means a concept in architecture of software where a single software instance (such as database, storage, server, software, etc.) works on a server with supporting multiple client organizations (tenants). As a consequence of deploying multi-tenancy feature of cloud computing, the same resource can be used by multiple customers at the application level, network level and host level [57].

2.5 Key Mechanisms of Cloud Storage

The architectural consideration of data storage in cloud computing needs more mechanisms compared with traditional data storage systems. This section presents the key mechanisms which need to be taken into account for cloud storage.

2.5.1 Distributed File System

A file system is a process that performs the management of data storage, access and operation. It is a logical disk component that controls a disk's internal operations as relation of computer and abstraction of human user. It is a set of an operating system which provides longstanding storage. It performs by operating files existed from specific creation until specific destruction and protection of temporal failures in the system. A distributed file system (DFS) is a file system which stores data on a server and its storage resources and files are shared by users. Accessing and processing of the data can be performed as data storage was at the local client machine. It is convenient for sharing of files and information to users on a network in the authorized and controlled manner. Sharing of files and storing of data as like storing of local information are allowed by the server to the client users. However, the servers give access control to the clients and have full control over the data [50].

Its structure contains server, clients and service. A server is the software that services and operates on one device. A client is a task that can request a service with a set of functions forming the interface of its clients which is organized by a set of file operations such as delete, write, create and read. A service is an entity of software that runs on one or more devices and supports a certain kind of operation to earlier unknown users.

In DFS, the dispersion of servers, storage devices and clients are performed over the devices in a distributed system. There have multiple independent storage devices substitution of a single centralized data repository and in that, the service has to be performed through the network. Popular file systems are google file system (GFS) [29] and hadoop distributed file system (HDFS) [3].

2.5.2 Service Load Distribution

In cloud computing data centers, service load balancing is the key capability in order to absolutely perform distribution of the load of service among multiple locations and servers. To do that, cloud providers should assist load balances and policies for cloud storage. Load distribution in the cloud may consist of several features such as redundancy, availability, regulatory issues, subscriber affinity, latency, capacity and security.

To meet the specific requirement of cloud data such as capacity, security, latency and availability, the following requirements should be considered;

(1) number of data objects,
(2) redundancy of data and application, and
(3) proximity of users and data objects.

2.5.3 Dynamic Data Allocation and Replication

Generally, the same information is stored by cloud service providers on multiple devices. In Yahoo and Google, splitting of cloud back-end storage into huge clusters and breaking entirely of these into blocks/chunks at 64 MB is performed. The identification of each block is unique and replication of those blocks to many servers in their data centers are performed. In this case, how many numbers of replicas are suitable and which data centers should be chosen to allocate replica efficiently is a challenging issue. This is critical in cloud storage because it can affect the performance and the cost of storage concerning latency and data availability.

2.5.4 Rapid Elasticity and Overload Control

Rapid elasticity enables the cloud service capacity to expend and contract rapidly while the service is online. It is a powerful cloud mechanism that can support rescaling and automatic scaling of hardware resources. As a consequence, the usage of resources is more efficient and the risk of overloaded conditions is eliminated. In order to increase rapid elasticity, resource monitoring, thresholds and metrics must be put in place. Therefore, the system should be designed to support rapid elasticity with the following:

(1) managing scaling and de-scaling,

(2) accurate recording and monitoring of resources and performance; and

(3) providing robust trigger mechanisms and well-defined policies to reliably accomplish the growth and de-growth of the application and automate them.

By taking advantages of rapid elasticity, the cloud storage system may automatically result the mitigation and perhaps elimination of overloading. Traditional overload control mechanisms can contain thresholds based on exceeding defined capacity which will result the rejection or shedding of traffic according to the severity of the threshold alarms.

2.6. File System for Cloud Storage

There are two kinds of file systems such as general parallel file system and distributed file system. The first one is intended for computation of high-performance applications which require more concurrent and scalable storage I/O and are implemented on large clusters [55]. In this design, the metadata server may be a cluster organizing by some servers, featuring by various metadata server supporting server for various client simultaneously. Examples are parallel virtual file system (PVFS) [15] and Sun’s LustreFS [59].

Internet services use widely the distributed file system and, the three main examples are hadoop distributed file system, amazon's simple storage service (S3), and google file system [29] at the complex cloud and internet environment. Moreover, HDFS are currently used by Facebook [32], Twitter [12] and so on.

2.6.1 Google File System (GFS)

The accommodation of the expanding data processing requirements of google are solved with the development and creation of a scalable distributed file system, google file system (GFS) by Google Inc. It supports the ability of reliability, availability, performance, fault tolerance and scalability for the connection of large networks to its nodes. It is built with various storage systems of low-cost commodity hardware components. It contains one master and many chunk servers and the access is done by many users. The general architecture of GFS is shown in Figure 2.3.

The metadata information about all file system is kept at the master. It contains the access control information, the current locations of chunks, the namespace and the mapping of files and chunks. It performs the management of system activities as like garbage collection of wasteful chunks, the migration of chunk servers and the lease management of chunk. The communication of the master with each chunk server is performed periodically with sending the heartbeat messages in order to perform the collection of its conditions and giving the instructions.

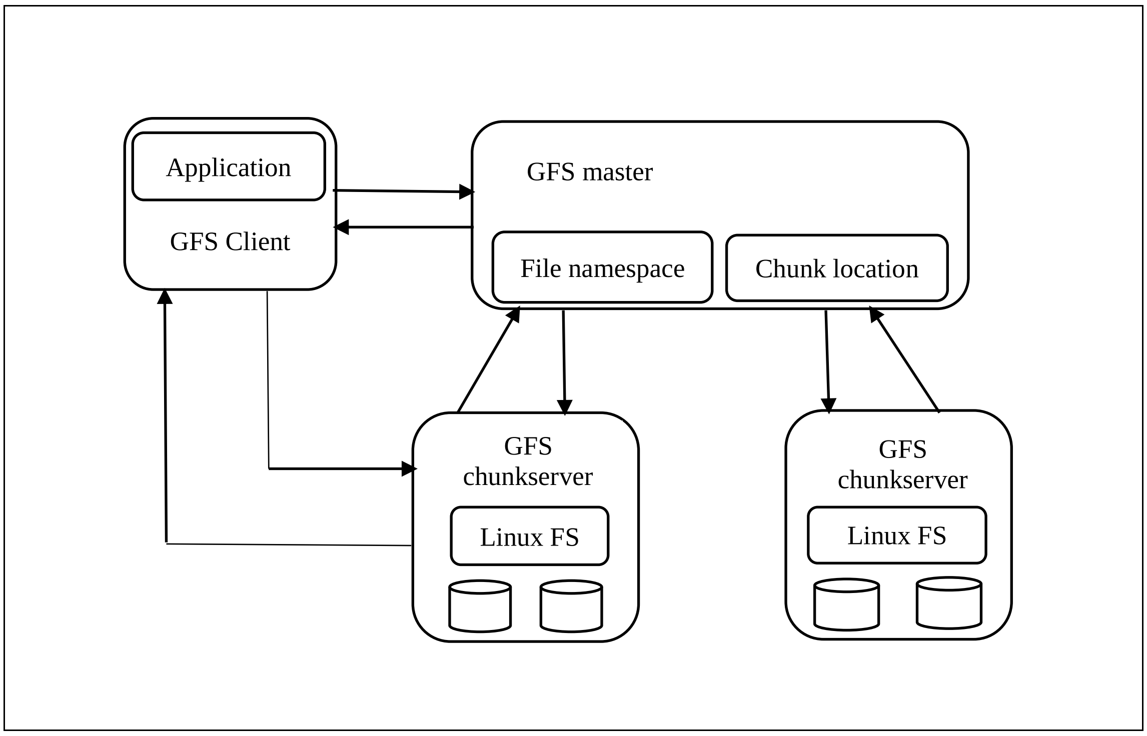

Figure 2.3 General Architecture of GFS

In GFS, input data is split into 64MB-sized chunks by the master in the creation of chunk. These chunks are stored at local disks by chunk servers as like linux files and write or read chunk data defined by a byte range and chunk handle. The replication of each chunk is performed at many chunk servers for reliability. The default replicas are three although various replica numbers can be specified by users for various regions of the file namespace.

2.6.2 Hadoop Distributed File System (HDFS)

Hadoop distributed file system (HDFS) is developed by yahoo and which is indistinguishable with GFS. However, HDFS is open-source and more light-weighted than GFS. It is scalable and is deployed with low cost hardware for providing reliable storage for huge amount of data and for streaming those data to user applications with high bandwidth [3]. In the large cluster, direct attached storage is hosted by thousands of servers and these servers performs the operation of user applications. It is built up with the interconnection of nodes with clusters. It contains a NameNode for the management of the file system namespace and the regulation of client access to files. Moreover, DataNode operates the data storage as blocks within files. HDFS architecture is shown in Figure 2.4.

It operates the separate storage of application data and file system metadata. As like other distributed file systems, metadata information is kept at a dedicated server, the NameNode and application data are kept at other servers, DataNodes. Full connection of all servers and communication with each other are operated using TCP-based protocols.

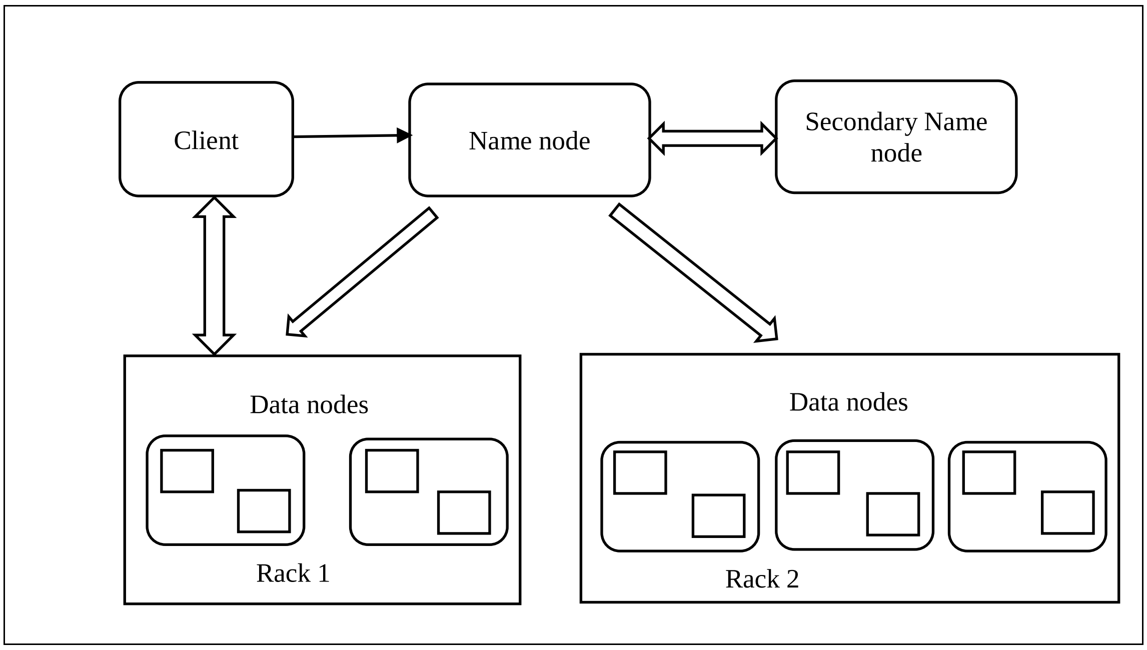

Figure 2.4 General Architecture of HDFS

However, it has some key issues. The first one is that the dependency on name node for the management of all data block operations in the file system. As a consequence, it occurs a single point of failure and a bottleneck resource. To solve these issues, a distributed file scheme was proposed in [27]. The system applies a light weight front end server for making the connection of many name nodes with all requests. This provides workload distribution of a name node to many data nodes.

Since the management of all files is done by the name node in HDFS, the system performance of name node is significantly impacted by small files. The mechanism was proposed for the improvement storage utilization of metadata and the efficient storage of small files [54]. This strategy applies hadoop's harballing compression method for better utilization of HDFS.

In order to eliminate the metadata burden on name node, a strategy was proposed for the increment of efficiency in accessing and storing small files on hadoop [26]. In this system, data access locality and file correlations characteristics remaining among small files are considered for accessing and storing them. The merging of all correlated small files of power point courseware into a larger file are operated and a two-level pre-fetching mechanism was proposed for the increment of efficiency in accessing small files [38].

2.7 Role of Data Popularity in Cloud Storage

In cloud storage systems, the increment of huge amount of processing and data storage requirements has caused to big data. The growth of the rate of change in file popularity has some key features in the architecture of cloud storage. As grow as the explosive amount of data in cloud environment day after day, the popularity of data in the system becomes an important factor in cloud computing. According to the analysis of [1][48], only a small fraction of the files typically counts for a large fraction of the access. There are only a small percentage of high popular files: less than 3% of the files count for 34% - 39% of the access. Young files count for a high fraction of accesses, but a small fraction of bytes stored.

The workloads are less skewness for popular files. Various types of storage media for various file types are combined as tiered storage systems that can be tailored to these workloads in order to increase system performance [21]. As data replication provides faster data access, improved data availability and decreased user waiting time by supporting the user with various replicas of the same service, the idea of tuning replication process based on data popularity is common. This approach is efficient when the amount of data in the system is large, especially of cloud storage.

2.8 Role of Data Locality in Cloud Storage

Cluster computing systems, featuring fault-tolerance data storage distribution, have been widely applied for data-intensive applications. Huge clusters contain tens of thousands of devices and that are designed for searching and web indexing; small and medium sized clusters are designed for corporate data warehousing and business analytics. The more closer placement of data with computing node is a common routine of storage systems, also known as the data locality issue. These systems apply static data replication for (a) improvement of data locality by placing a task and its data at the same local storage (b) achievement of load balancing with distribution of work among the replicas (c) ensuring fault tolerance and data availability in system failure.

Static data replication and placement are applied in current mapreduce applications. In order to achieve the optimization of data locality, applications depend on the scheduler. There are two ways for the improvement of data locality:

- Assignment of popular data with many data replicas are performed for the improvement of data locality with concurrent accesses
- Placement of various concurrent accessed data blocks into various data nodes for the contention reduction on a particular node

Low throughput due to the poor data locality can be eliminated with the improvement of the data locality. The number of data replicas is automatically increased with the duplication of data to fetched node in DARE [2]. However, the data replicas are decreased if the insufficiency of data storage occurs. Therefore, there is the limitation for supporting the suitable number of data replicas with the access history information.

2.9 Challenges of Cloud Data Storage

Nowadays, there are many challenges that must be solved in the combination of cloud storage in data-intensive environments.

- Data transfer rating and system performance are key problems when the increment of interval between client and data that occurs in cloud.
- The latency of the network protocols cannot be eliminated with even unlimited bandwidth and the speed of light limitations that occurs the client experience to be very poor.
- There is no well suitable data access information to cloud, if there is, at remote locations. In that conditions, network bandwidth is both an issue and a key factor of financial environment.

- There may be impossible for the replacement of the storage network in the data center by cloud storage any time, at least not for high performance, transactional applications, data-intensive, mission-critical data and low-response time.
- The cloud storage applies instances for less frequent data access natures such as backup, offsite data protection, archiving and DR. Data transfer rates and system performance become principal factors when the increment of interval between client and data that occurs in cloud.
- There will be a hybrid cloud storage with a lot of that storage living in private clouds for many years. Many large enterprises deal with their processing and data petabytes [90].

2.10 Summary

This chapter presents overview of cloud computing and storage technology including design characteristics, key components and architectures. Moreover, distributed file system such as Google File System and Hadoop Distributed File System are also discussed. In addition, how is the role of data popularity and data locality important in cloud storage are also presented. Finally, the challenges of cloud storage are presented as last section in this chapter. According to the characteristics of cloud storage system mentioned in this chapter, the replication management, data popularity and data locality are designed in the following chapters.

CHAPTER 3
THEORETICAL BACKGROUND

Data replication is a method of the duplication of an entity such as file, database and data, etc. In storage system, it is widely used in order to control cost for unnecessary storage and improve throughput, response time and availability for users. As data availability is a principal component for system performance improvement in cloud environment, replication is the key factor for performance improvement in cloud computing, that is, it provides services to users as service level agreements (SLAs).

There have been much research for various techniques of data replication. These proposed techniques solve two replication problems such as replica allocation and replica placement problems. Replica allocation problem is the determination of the number of data replicas for each file and the placement of replicas to where. The management of replicas is a critical factor for data availability and efficiency of storage. In data-intensive systems, closer placement of data to computation is a common practice. Moreover, the concurrent placement of different data blocks with different data nodes is a replica placement problem. The performance of a distributed system is largely affected by the replication strategies and methods.

3.1 Common Replication Strategies in Cloud Storage System

In cloud storage system, there are two general data allocation strategies which are currently used in today enterprises. They are static replication strategy and dynamic replication strategy.

3.1.1 Static Replication Strategy

Objects are traditionally and generally replicated in a static manner, that is, the scheme of data replication is designed by the distributed database manager and it remains constant until the execution of manual reallocation by the manager. It is a reasonable solution if the read-write patterns are known a priori and are fixed. If these patterns change dynamically and are unpredictable, a static replication scheme may occur several problems of system performance.

Not only in cloud storage systems but also in traditional database systems, static replication is well and simple technique. In static replication, the predetermination and well definition of the replication strategy is performed. The pre-configuration of the number of replicas is performed before data storage. As the pre-configuration in storage setting, the replication mechanism will duplicate the static number of data replicas. In storage system in that data access nature of file system is rarely changed and quite stable, this static technique is very useful. Today popular cloud storage systems as like HDFS and GFS use this static replication technique for their data centers. In HDFS, the fixed replicas are 3 and these replicas are assigned into data nodes with rack-awareness policy in order to achieve availability and reliability. The disadvantage of this static replication is non-adaptability of changes in dynamic user behaviors.

3.1.2 Dynamic Replication Strategy

In spite of the easy management and the usefulness of static replication, it has the rare adaptability of dynamic conditions as like the higher access of popular files or rarely used files. It may occur delay response time and network bottleneck as the very frequent and the more access of those files than others. In order solve this issue, the static replication is not effective in data availability and response time and it requires the replication of more than the default replicas for those popular files. However, the dynamic replication has the ability of determination on the dynamic nature in the number of replicas and assignment of those replicas into nodes based on the recent status of storage systems. However, the drawback of this replication is that it requires a replication decision center in order to perform the collection of the runtime information of all nodes in the system.

3.2 Dynamic Replication Strategies for Data Popularity

In order to decide replication strategies dynamically, the system requires the central decision maker and runtime information such as data access history. Access history record is a key factor in order to identify the data popularity which is usually applied for dynamic data replication. In this section, dynamic replication techniques based on data popularity are discussed.

3.2.1 History-based Proactive Approach

Replication of files are periodically performed according to prediction of popularity in proactive replication scheme. To perform the implementation of this technique, the accurate prediction of data popularity is critical. Otherwise, too few or too many replicas will be created which leads unalleviated contention or to waste both storage and network bandwidth. While minimal interfering with running jobs, Scarlett [6] performed the replication based on access information and assigned with nodes in order to avoid hotspots. A proactive approach, Tempo, is proposed by Emil Sit and et.al.,. Tempo performed effectively the adjustment of the replication level with the constraints of bandwidth budget while systems defined the number of replicas corresponding to failures by changing the consumption of bandwidth to remain unchanged that replication level [69].

The comprehensive reliability model was proposed that considered not only probability of data loss but also bandwidth allocation in the recovery process [77]. They used proposed reliability model for analyzing reliability and system repair rate for different data layout schemes, namely copyset replication, shifted de-clustering and random de-clustering layouts.

A novel cost-effective data reliability management mechanism based on proactive replica checking (PRCR) was proposed that checked the availability of replicas to maintain reliability [51]. They showed that default three-way replication strategy consumed storage space for rarely accessed files. Thus, they proposed a reliability model with the aim of reducing storage cost and demonstrated that wide range of data reliability can be assured with the maximum of two replicas stored in the cloud.

A distributed hash table is used for the allowance of defining a maximum maintenance bandwidth and to perform the proactive replication. It evens out bursts in maintenance of traffic by varying in time at which bandwidth is used. In the idle time, a proactive system performs regularly in the system background in order to increase levels of replication. Hence, this consequences in a sudden split of failures and a predictable bandwidth load: nodes will not occur to a respective split in usage of bandwidth that might occur the disturbance of the network. However, a deducible knowledge of failure condition is required to support durability. If there has no precise or true knowledge, the compromising of durability may be occurred.

3.2.2 Load-adaptive Reactive Approach

Reactive approach has adaptable and dynamic nature than proactive approach. Therefore, this approach has ability to changes in popularity in smaller time periods and can eliminate non-recurrent and recurrent hotspots. In DARE [2], data replication scheme was proposed to be efficient in cluster scheduling by using the nature of this reactive approach. In DARE, the replica allocation and replica placement algorithms are adaptable to the changes in workload. In replica allocation, it finds the most popular data and generating replicas for this data. Data popularity has the nature of the large number of access and the high intensity of access.

Although reactive approach only generates required replicas resulting in minimization of total bytes sent, crash in network usage can be occurred after a failure. These crashes may occur the disturbance of application traffic and bandwidth provision may be difficult.

3.2.3 Greedy Replication to Popular Data

The approach that makes adjustment in changes of data popularity is greedy approach. The assumption of the greedy approach is that the access of nonlocal data is worth replicating. This approach performs the unnecessary replication of unpopular data as some jobs tend the achievement of poor locality. For instance, in the map-reduce processing of DARE, the insertion of data into the fetched node at HDFS when a remote data is processed by a map task. The automatic increment of the number of replicas by one is done without explicit incursion of network traffic in this process. An eviction is needed to be performed as the limited assignment of storage space for dynamic creation of the replicas. Conventional eviction methods contain least frequently used (LFU) and least recently used (LRU). Choices between LRU and LFU should be made after profiling typical workloads.

3.2.4 Probability-based Greedy Approach

A problem that may arise from Greedy approach is thrashing. In this context, thrashing is a high rate of replica creation and eviction. To add stability and prevent thrashing, probabilistic approach is an optimal option so that blocks are not dynamically replicated immediately after a remote read, but rather they are replicated with a probability p. The algorithm iterates through the list of dynamically replicated blocks if the budget limitation occurs and the replication triggered by the remote access. By adopting a probabilistic approach, most unpopular nonlocal accesses can be ignored while replicating popular data.

3.2.5 LALW Algorithm for Data Grids

A data replication management scheme, Latest Access Largest Weight (LALW), is proposed in [18] for data grids. In this proposed system architecture, replication management is performed by the replication policymaker which is the centralized data replication management mechanism. The cluster header performs the management of each cluster and the policymaker performs the collection of the access information of all headers. The detailed information of file is kept at each cluster site and the cluster header takes the aggregation and summarization of all records in the same cluster. The summary of all records is sent to the Policymaker which selects the popular file according to the number of accesses for files and weight of records. At each time period, the algorithm searches the most popular file and performs the computation of the number of replicas for popular file and the suitable placement of the replicas at grid sites. Analysis of the access information of files was used in order to decide the popular file in this system. After the determination of most popular file, the investigation of the generation of the most accesses of popular file and placement of the new replica are operated.

3.2.6 PopStore Algorithm for Cloud Storage

As the popular growth of cloud computing, the role of storage system is important to perform the efficient storage and distribution of the massive amount of data to data centers. There is the random and dynamic nature of data popularity in cloud environment as the growth in the explosive amount of data day after day. Moreover, the maintenance of the fixed number of replicas in the system consequences in inefficiency of the most accessed data and non-effective and wasteful storage cost for unpopular data. In [34], adaptive data replication approach, PopStore is introduced by applying popularity thresholds and history of data access information in order to solve these issues. To evaluate the performance analysis of this system, Yahoo hadoop audit log file is used as a data source in order to perform the extraction of data access pattern.

A replication algorithm, PopStore is proposed using half-life concept as like LALW algorithm in grid computing. Half-life concept is the decaying of the weight of the records to half of its previous weight in an interval. LALW searches the most popular file at each time interval. However, PopStore finds out not only popular file but also non-popular file. After that, it computes the different number of replicas for various files at each time period with different popularity thresholds. In order to point out the importance of history information, time-based weight setting is used. Smaller weights are set to older history information. Moreover, PopStore specified different number of replicas based on different thresholds of popularity. After the calculation of the number of replicas for each file, the effective placement of replicas to data center is operated. The assignment of data replicas to the nearest replica site is performed for reducing the storage cost. PopStore is adaptable to the changes in data popularity of cloud storage not as like the existing methods because its consideration of data popularity.

3.3 Replication Strategies Based on Blocking and Anti-blocking Probability

Replication is the common strategy in cloud storage systems in order to increase data availability at where the failures are normally occurred. In [78], a dynamic replication technique for cloud storage called CDRM is introduced for performance improvement, data availability and load balancing. It considers the relations**h**ip between the number of replicas and data availability. In order to satisfy the data availability requirement, it calculates and keeps the minimum number of replicas. According to blocking probability and storage capacity of nodes, the placement of replicas is performed. It adjusts data replicas and assigns data replicas to nodes based on changes in node capacity and workload. It can operate the dynamic redistribution of workloads at nodes in heterogeneous cloud environment.

In cloud computing, there is a challenge for effective access to widely and huge distributed data for replication. To solve this issue, an Efficient Data Access Scheme (EDAS) is proposed for HDFS for adaptive selection of data replicas among nodes. Data distribution and replication is performed at cluster with commodity nodes in HDFS. Depending upon the load of nodes, this scheme provides the determination of the access nodes for data replicas to users for getting quick access to nodes. It supports high performance replication access and load balancing of nodes. It is implemented based on history access information of HDFS metadata and anti-blocking probability of nodes [8].

3.4 Replication Strategies for Load Balancing

As the rapid development of many storage applications such as Google Drive, Megaupload and YouTube, data storage nodes that have popular data storage have led to the bottleneck in system performance. As a consequence, high request loss rate, low resource utilization and long-latency response could be occurred by load imbalance. The architecture of an effective load balancing scheme is a key issue in order to solve this shortcoming. In general, replication is a general way to satisfy such requirement. Data replication is the technique of keeping many duplicates of same data on same or different servers. In cloud computing, data replication means the storage of many duplicates of same data on distinct places, local or remote locations. There will be very hard for handling the access requests if the existence of data is at one-sided. As a consequence, the server will encounter system performance degradation and heavy load condition. Moreover, there is failure at that site, this is also a serious concern as the loss of all that data. For keeping the level of performance, data availability, load balance and back up data storage, data replication is the essential technique. A simple and efficient load balancing scheme, namely, ARM was proposed [80]. The uniform distribution of the hotspot data access to other nodes can be performed with active replication and the effective utilization of storage resources can be operated with the execution of on-demand dereplication. The excellent load balancing can be obtained with the accomplishment of the optimal number of data duplicates for hotspot data on adequate storage nodes as the consideration of long-term and short-term data access natures.

The data placement is need to be considered as a critical factor in evaluation of the system performance such as load balancing. In HDFS, the current replica placement policy the replicas of data blocks cannot be evenly distribute across cluster nodes, so the current HDFS has to rely on load balancing utility to balance replica distributions which results in more time and resources consuming. In [4], the heuristic approach is introduced in order to handle the problems in assignment of data into nodes. It distributes replicas to cluster data nodes as evenly as possible, and also meet all replica placement requirements of HDFS, as a result, there is no need to run the balancing utility. The proposed policy in this paper is the first that addresses the load balancing problem by generating an even replica distribution to the data nodes at the beginning of distribution then during the normal operations on data nodes.

The load balancing approach is introduced with the consideration of all conditions affecting the load balancing. In the proposed algorithm, a new role named BalanceNode is introduced to help in matching heavy-loaded and light-loaded DataNodes, so light-loaded nodes can share load from heavy-loaded ones [81].

3.5 Replication Strategies Design for Heterogeneous Clusters

The architecture of hadoop distributed file system (HDFS) is intended for supporting data streaming high bandwidth to customer implementations and provides the reliable storage of big data. However, the assumption of the block placement policy of HDFS is that the homogeneity of all nodes and the random placement of data blocks no consideration of resource utilization of nodes which occurs the decrement in system self-adaptability. To solve the shortcomings in data placement of HDFS, an advanced block placement approach is proposed [84]. In this system, it considers the non-homogeneous features of nodes such as disk space utilization of nodes. The concept of this proposed approach is that it divides nodes into two groups: small network load and high network load. The network load difference between two groups is not greater than the predefined threshold, the selection of the nodes in the small load group with much disk space can be performed in preference. If not, the nodes in the high load group with much disk space is selected by this strategy. This strategy mainly focuses on load balancing with the selection of the suitable node for data placement instead of realizing balance by the default balancer procedure.

HDFS places randomly the data replicas into nodes without consideration of the heterogeneous feature of nodes. The placement policy of HDFS is not effective for heterogeneous environments, where nodes have the same disk capacity and same computing power. Practically, the holding of the assumptions of homogeneous environment is not always easy. The scheduler of Hadoop will occur the serious risks in dissipation of energy in heterogeneous environments and degradation of system performance as the placement policy of HDFS. To address these issues in large-scale non-homogeneous hadoop cluster, the novel snakelike data placement mechanism (SLDP) is introduced in [79]. Adoption of heterogeneous features, SLDP proposes an algorithm that separates nodes into several virtual storage tiers (VST) and then assigns that blocks into nodes in each VST with a circular path according to data popularity. Moreover, it applies an efficient power control function and a popularity-aware replication for decrement of disk space consumption.

3.6 Replication Strategies Design for Energy Efficiency

The huge amount of file transactions such as storage, transfer and processing are simultaneously operated in large distributed data clusters. Many file systems perform the creation of three replicas and the random placement of these replicas into nodes among various racks for the increment of data availability. However, they do not consider the heterogeneous nature of nodes and file that can support further improvement of system efficiency and data availability. As the dynamic nature of file popularity, the fixed number of replicas may not be adequate for supporting immediate response to larger number of data access to popular files and waste unnecessary storage resources of unpopular files. It is critical in the choice of nodes for low delay of data access and replication as the heterogeneous features of nodes such as system configuration, network bandwidth and the maximum capacity of simultaneous data access requests.

In order to achieve the efficiency of energy in data replication, the energy-efficient adaptive file replication system (EAFR) is introduced [52]. In order to obtain a balance between efficiency and availability, it is adaptable to changes in data popularity over time periods. Increment of data popularity occurs more data replicas and decrement of data popularity occurs less replicas and so on. In order to obtain efficiency of energy, it splits servers into cold and hot servers with various consumption of energy and performs storage of popular files at hot servers and storage of unpopular files at cold servers. It chooses the server that has the sufficient storage and network bandwidth for keeping the replica. It introduced the adjustment approach of dynamic transmission rate in order to avoid congestion potentially in replication of data to the server, the replica maintenance approach based on load for the creation of files quickly when node failure occurs and the node selection approach based on network for eliminating file access latency.

The cost of energy consumption is one of the considerable factors of the overall costs of the data center as the requirement of much energy for the execution of applications on large clusters. Therefore, in execution of each mapreduce jobs of data centers, eliminating of the energy consumption is a principal issue. In order to improve the energy efficiency in mapreduce applications with the achievement of the service level agreement (SLA), a framework was proposed [56]. In this system, a mapreduce job was designed as an integer program based on the scheduling of energy. Then, two heuristic algorithms were proposed to make the assignments of machine slots with map and reduce tasks to get the minimum energy consumption in the application execution. These experiments are performed on hadoop cluster to make the determination of the time for execution and the consumption of energy for various workloads on the benchmark suite such as PageRank, Terasort and K-means clustering and this data are used in the simulation for the analysis of the system performance.

In order to manage data processing in hadoop cluster, a hybrid, energy-saving and logical multi-regional alternative was proposed [41]. Green hdfs's data-classification placement policy scales down with the substantial assurance for long periods of idleness in a set of servers in the datacenter designated as the cold zone. These servers are transformed into energy-saving and inactive power modes with no impact on system performance of hot zone. This shows that the servers in the clusters are under-utilized and the existence of abilities for better integration of the workload on the hot zone. The analysis of the traces of Yahoo hadoop cluster showed the dynamic nature of data access patterns can be applied for the placement policies based on energy.

3.7 Elastic Replication Management Scheme

While the improvement of system performance by the replication strategy, the more than half of workload in the object-based storage system are the metadata operations. Therefore, elastic replication strategy based on the communication model of logical elements and physical nodes in storage system was proposed [53]. In this system, the formalization of the replication and the popularity of metadata was specified. After that, the capability of the metadata server-based replication of the metadata is performed in the cluster using the access history information of data. The evaluation and analysis show that this replication scheme can perform the improvement of the metadata efficiency in the storage system that can perform the further improvement of system performance.

With the replication of default three data replicas, HDFS supports reliability, availability and high performance. The pattern of data popularity is dynamic over time periods. The HDFS replication strategy should be adaptable with the nature of data popularity in order to achieve high disk utilization and improvement of system performance. Therefore, an elastic replication management scheme for HDFS was proposed [20]. This scheme proposes elastic replication for various data types and provides an active/standy model taking advantages of high-performance complex event processing in order to classify current data types. It applied Condor for the removal of unnecessary data replicas after the data becomes unpopular and the increment of data replicas for popular data in standby nodes.

As the data becomes unpopular, the erasure codes are applied for storage and network bandwidth saving. It operates the management of replica allocation and replica placement in the cluster. To classify real-time data types, it applies CEP and the system metrics was obtained from the cluster. Scheduling of replication manager tool and erasure coding tool could be done for replication management depending on the different data types.

3.8 Replication Strategies based on QoS-aware data replication

A replication management scheme based on the awareness of QoS is proposed [68]. This scheme calculates the suitable places for data replicas in order to minimize the overall replication cost. Moreover, this scheme focuses on the reduction of access latency and the improvement of data availability while achieving the maximum QoS requirement. This issue is designed by applying dynamic programming. For the demonstration of this proposed scheme, widely observed access history information are applied in order to implement the simulation experiments. For cloud computing environment, two data replication algorithms based on QoS are proposed [19]. One algorithm assumes the concept of high QoS first replication (HQRS) for replication. However, this greedy approach could not reduce the QoS-violated data replicas and the cost of replication. Another algorithm changes this issue as minimum cost maximum flow (MCMF) issue in order to obtain these two objectives. To solve this problem, the second algorithm provides the ideal key to this issue in polynomial time by using current MCMF algorithm. However, more computation time is required than the first algorithm. The approach that applies the combination of node was introduced for the reduction of much replication time in cloud computing.

The authors proposed the balanced and file reuse replication scheduling (BaRRS) approach for the optimal arrangement of scientific workflows in cloud environment [16]. To achieve the balance between parallelization and system utilization, it divides workflows into multiple sub-workflows. Replication and data reuse approaches are applied in runtime for achieving the suitable amount of data transported on jobs. The key features of workflows such as dependency patterns, task execution time and file size are analyzed for the adjustment of current replication and data reuse approaches in cloud environment. Moreover, a trade-off analysis is taken for the selection of the suitable key based on two constraints: monetary cost of running workflows and execution time.

3.9 Replication Strategies for Peer-to-Peer Architecture

In peer-to-peer system, unreliable connection, unpredictable node failure and bandwidth limitation confuse effective sharing of data. Data replication can increase response time and data availability. In order to achieve improvement of system performance in massive storage systems with random network features, dynamic behaviors of user and large files and users, determination of where and when for data replication is yet very hard. A dynamic replication model is proposed by Ranganathan et al. The optimal number of data replicas are computed with storage cost of files, system availability, accuracy of data placement and latency between nodes while satisfying the required availability [64]. This system is evaluated with the simulation environment for showing this replication approach outperforms than default replication. It also shows that this approach provides the exact prediction of the number of replicas in this system. However, there is problems such as wasteful replication for unnecessary data at sometimes and incomplete information of nodes.

The efficiency of replication is determined by many features containing data placement approach. A priori data replication approach for P2P data grid systems was proposed by Challal and Bouabana-Tebibel [17]. This approach implements dynamic replication with the suitable assignment of initial replicas to nodes before the starting of the tasks. The improved availability can be achieved by minimizing the intervals between various data replicas and maximizing the interval between the same data replicas. It ensures that each data node has various copies of various file in its surroundings. In the simulation environment, comparative analysis of this approach with no initial data placement and dynamic initial data placement. The proposed approach minimizes file transfer time and maximizes job completion time no increment of storage cost and network bandwidth.

An economic-aware replication scheme was proposed [11]. It performed the optimization of dynamic creation of data replicas at grid environment and selection of data replicas for running tasks. In this system, it applies an auction protocol for selection of the suitable copies of a file and an estimation function for determination about local replication. It placed optimization agents on grid environment. Files are bought by computing things and its objective is the minimization of buying cost. Likewise, storage elements increase the profits and provides investments depending upon the predictions of data access pattern in order to maximize revenue.

A peer to peer network supports interconnection of nodes and is one of the most usable local area networks. However, the increase in the workload on the server or some centralized nodes occurs as the increase in the communication over the network. In this condition, the distribution of network workload is performed for the setup requirement of some sub systems that are replication servers on the network. There are full or the partial copy of the actual centralized server. The key issue is the computation of the required number of these replication servers in system. In [66], the network model for distributed system is proposed in order to achieve the effective replication across the network with an easy manner. This proposed model is defined depending upon the estimation of cost. This model contains the following key features:

- The wireless connection of multiple access points to each other
- The constant and predefined location of all access points
- Only part of the access points has a physical link to the Internet, and thus act as gateways.
- The connection of the internet with multiple mobile clients through the gateways. The connection of a client to one gateway is at each time point.
- Either direct connection or connection through a series of forwarding access points to the gateways
- Dynamic switching gateways and/or routes performed by clients in the simulation. A nomadic service assignment algorithm operates the assignment of a gateway with a client.

3.10 Replication Strategies of Cloud Storage Systems with Master/Slave Architecture

Master/slave architectures are commonly used in today popular cloud enterprises such as Google and Hadoop. Depending on the master/slave architecture, the replication techniques are different on management of replicas and control strategy. This section presents replication approaches which are currently used in master/slave cloud storage architectures.

3.10.1 Master-Push Replication for Multiple Storage Clusters

The architecture pattern of Master-Push replication is originally used in HBase storage cluster which provides the same data structure of google BigTable such as row keys, column names, tables, column families, cell values and time stamps. Amongst multiple clusters of HBase cluster, one cluster acts as a master cluster and others can participate as slave clusters. Each server in master cluster has its own write-ahead log and that log provides the easier tracking of the current replication. Replication of any slave clusters can be performed by a master cluster and replication of own stream of edits will be participated by each region server. This replication is asynchronous replication as the geographical distance of the clusters and the insertion of rows on the master cluster will be unavailable at the same time on the slave clusters. The overview of HBase architecture is shown in Figure 3.1.

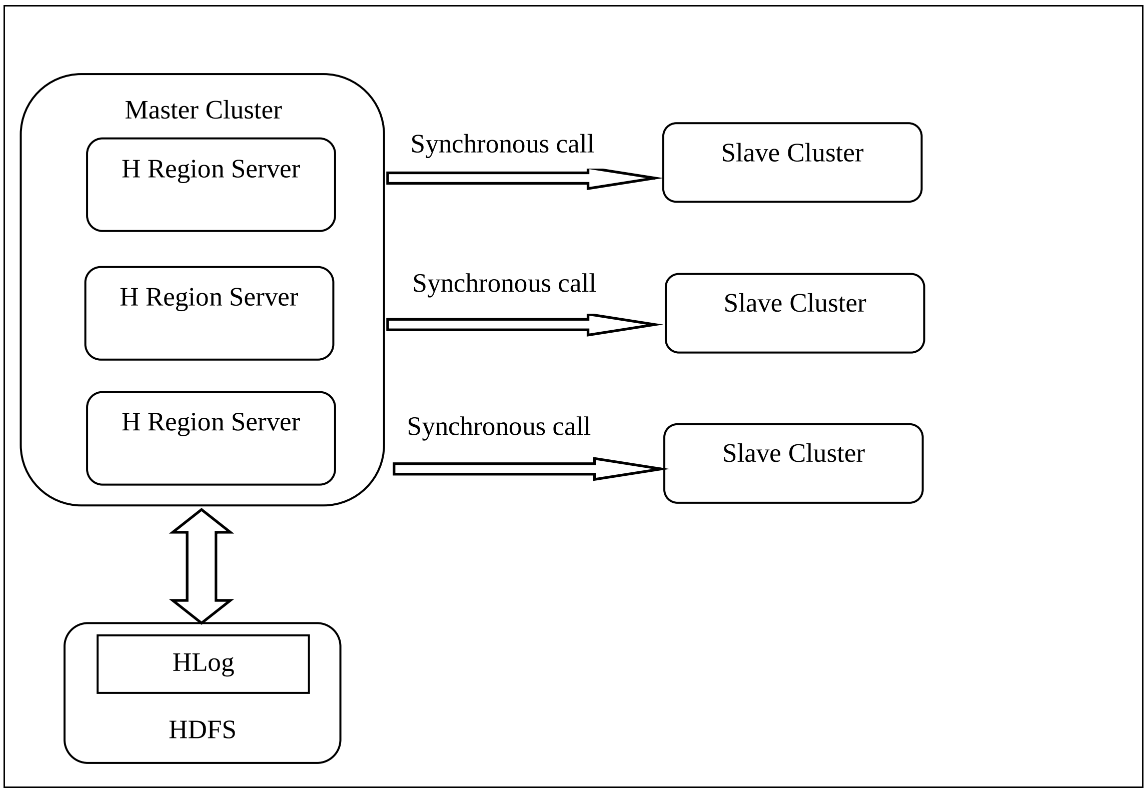

Figure 3.1 Overview of HBase Replication Architecture

3.10.2 Rack-aware Replica Placement Policy for Commodity Clusters

Rack-aware replication is designed for large scale storage cluster, the storage nodes are organized by spreading multiple racks. Each rack has a switch which are shared by multiple nodes and rack switches are connected each other. Figure 3.2 shows the example of HDFS rack organization.

The communication between nodes in various racks are performed with many switches. In replica placement, the first one is placed at the local storage by HDFS, the second and third ones are placed at two distinct data nodes in distinct racks. If there are another replicas, these rest replicas are placed with random placement. However, it does not place more than two data replicas in the same rack. By placing with that policy, data availability is increased even in the unexpected unavailability of the whole rack. To push data to the selected nodes, pipeline mechanism is used in the order of proximity. Thanks to rack-aware policy and the pipeline mechanism, the inter-node write traffic and the inter-rack are eliminated which generally improves write performance.

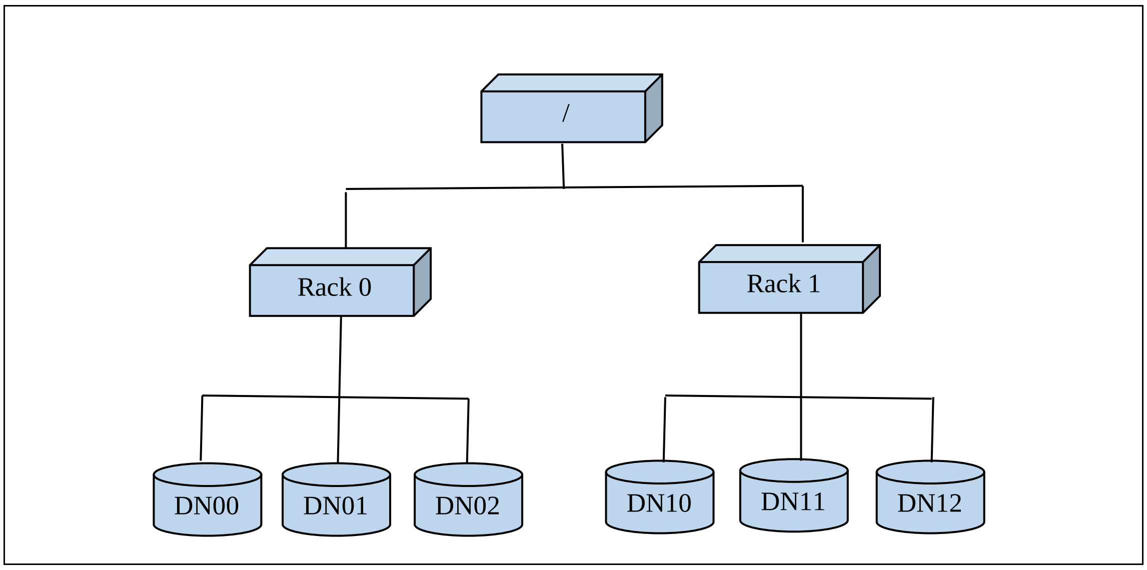

Figure 3.2 HDFS Rack Organization

3.11 Replication Models

Analysis and modeling may also be interested to support the concept of the interdependencies contained in cloud computing [62]. It is particularly suitable for evaluating the system and defining suitable values. In this section, some modeling approaches for replicated system are reviewed.

3.11.1 Full Replication Model

Most systems are modeled as full replication to consider performance evaluations. Full replication is the replication of all data objects at all places so that each place keeps a full duplicate of the distributed database. There is an extreme event of replication and it has been recognized that for many applications neither full nor no replication is the optimal configuration [22], [62], [5].

3.11.2 One-dimensional Partial Replication

Partial replication is designed in the way that the replication of some data is done to all places or the replication of each data is performed to some places. In the first case, the replication of some data to all places, the number of data replicas r is defined as $r \in [0;1]$ that represents the portion of logical fully replicated data items to all places. A data object is either not replicated at all or fully replicated. A value of r=1 describes full replication and r=0 represents no replication. These replication models have been considered by [62], [5]. However, [7] argues that full replication is only the assumption that are acceptable in the worst-case determinations.

In the latter case, the number of data replicas r is defined as $r \in \{1,2,....,n\}$, that represents each logical data item is denoted by r physical replicas, where n is the number of places. A value of r=n describes full replication, $r > 1$, every data object is replicated and r=1 represents no replication. As a consequence, either all data objects or no is replicated. The assumption is that copies are distributed evenly across the places, however it is still specified which replicas are assigned at that places, such that distinct number of data replicas for a replication scheme can be modeled.

3.11.3 Two-dimensional Partial Replication

In 2D-model, replication is designed in the way that some data objects to some places. It is represented as a set $(r1, r2) \in [0;1]x \{2, ..,n\}$ so that $r1 \in [0;1]$ that represents the partition of data objects denoted as r2 replicas, such that these data objects are duplicated to r2 replicas at the n places. The sharing of 1-r1 data objects left unduplicated, i.e. are described by only one data replica. Full replication is represented with (r1=1, r2=n). No replication is denoted with r1=0. This model does not specify the placement of data objects or the selection of data objects for replication. The exploited property of unspecified data placement and data objects selection can be used for the designing the quality of replication.

It is difficultly affordable for replication of some data objects in all places that occurs high update propagation and others into none that causes the decrement of data availability in large wide area distributed database. Therefore, the strategy for replication of some data objects in all places is not realistic. Moreover, there has read intensive data for duplication to many sites while updated intensive data for duplication to very few places in many systems. It may not be designed by the strategy for replication of all data items in some places.

3.11.4 Replication-per-Objects Models

The 2D model has limitation in the assumption that the replication degree is static for replication of all data objects although it is obviously more indicative than previous one-dimensional strategies. It can be solved if the number of data replicas is considered as a variable on a per data item basis at the cost of a considerable higher design complexity.

The number of data replicas for each item is specified individually for each item in that model. For each of the d data items are assumed as 1, 2,....,d, and the operation of replication scheme is r: $\{1,2,..d\} \rightarrow \{1,...n\}$ so that r(i) is the replication degree of object i. The distribution or assignment of the replicas over the n places lefts undefined although the replication degree is defined separately for each object.

Therefore, the extension can be made by defining not only the number of replicas but also the assignment at places separately for each object. For d objects and n places, the operation of the replication scheme is specified by r: $\{1,...n\} \times \{1,2,...,d\} \rightarrow \{0:1\}$ so that r(i, j) =0 if place i does not keep a copy of object i and r (i, j)=1 if place i keeps a copy of object i. This scheme specification may be found in [14].

True replication per object designs are of reasonable complexity as they require that distinct places keep the distinct number of copies and will hence be exposed to various workloads [60]. Therefore, the non-heterogeneous assumption applied at the analysis of system performance of distributed databases is violated such that the separate computation of system performance factor would be done for each place.

3.12 Data Locality

Data locality means the distance to which the processing and data for a operation are co-located on the local storage. Maximizing data locality is an important goal for many data intensive systems because it can have an obvious effect on the system performance in the data intensive jobs. The less data transfer across the network can be achieved by increasing data locality. Hadoop attempts the automatic collocation of the processing node with the data for achieving data locality. Map tasks are scheduled for setting the data on the same rack and the same node. Data locality is a critical key factor in consideration of the performance of hadoop. The delay scheduler is applied as the scheduler based on only data locality among the current schedulers of yarn and hadoop. There are potential performance problems concerning with data locality in YARN. First, the policy of a fixed number of replicas in Hadoop Distributed File System (HDFS) does not help to improve the data locality as the data access frequencies from different applications may vary. Second, when a YARN container requests a remote data block for processing, YARN does not keep a local copy of this data block for future containers that may require the same data block.

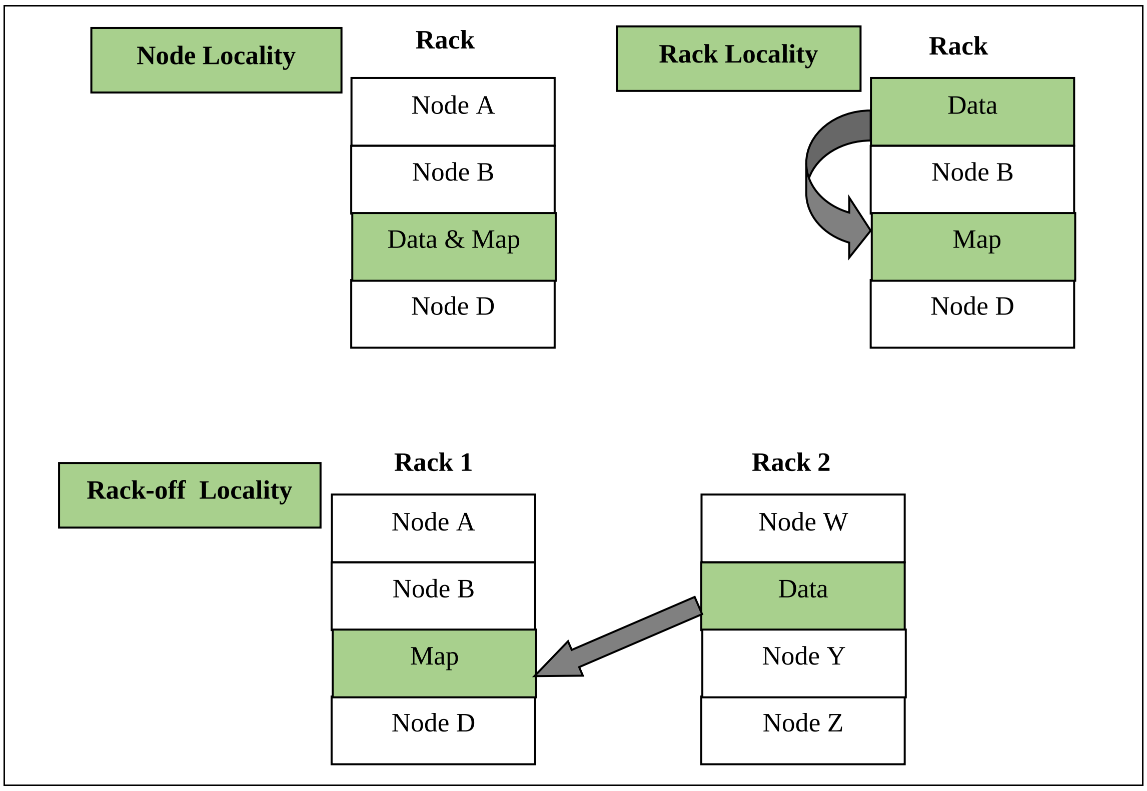

Figure 3.3 Types of data locality

Figure 3.3 describes types of data locality in hadoop. That types of data locality are recognized by the delay scheduler as follows:

- **Node locality:** In this locality, the determination for the scheduling of the input task with a node kept the needed data block for processing are made by the delay scheduler and that locality is the most effective type among locality. In this condition, the retrieval of data from remote location is needed to be done.
- **Rack locality:** If there is no node locality, as the local nodes do not have the available sufficient resources, the delay scheduler wastes a few seconds for the desired availability of one node of the local storage. If not, the delay scheduler makes the assignment of the input task with one of the local nodes on the same rack.
- **Rack-off locality:** It is the worst case of data locality, if there is no the available rack locality, the delay scheduler wastes further a few seconds, and if there is no the available local node on the same rack, the delay scheduler makes the assignment of the input task with a node on a distinct rack for the avoidance of task starvation. The delay scheduler. This locality makes the most high-cost determination for this locality.

3.13 Research Methodologies in Data Locality

Recently, a few studies attempted to improve data locality in Hadoop. This section categorizes the approaches to data locality management in the literature.

3.13.1 Locality-Aware Replication

Pegasus is a scheme for the management of workflow and it applies replica location service (RLS) for the achievement of data locality [25]. RLS is a distributed replica management system and it keeps file name mappings information of logical to physical data and the available distributed indexes. Its algorithms query RLS for the retrieval of replica places for the input tasks.

Ranganathan and Foster found that scheduling algorithms that target only processor utilization by mapping jobs to idle processors without regard the cost of retrieving the data from a remote site are inefficient [65]. A decoupled scheduling framework was introduced for data intensive operations that uses the separation of the separates the replication policy and the job scheduling policy. This framework contains three components: a local scheduler (LS) designed at each node to make the decision for the priority of the arrived jobs at this node, the dataset scheduler (DS) to detect the popularity of the data items and make the decision of which data items are to be deleted or to be replicated and the external Scheduler (ES) to make the decision for the submitting of the nodes with the jobs. In simulation experiments, the ES did the scheduling of jobs to either the place where the data is stored or the least utilized place. The DS either did random replication or no replication or replication at the least workload place among its surroundings. The system concluded that scheduling a job to a machine where the data is available results in better response time than an scheduling a job that fetches the data remotely. Also, the proposed technique causes some places kept the data to form skewness condition and, in that condition, dynamic replication should be applied.

Scarlett is an offline replication system that performed the replication of data blocks according to the observed probability from the hadoop job history logs in last time periods [6]. It uses a sample of the historical statistics from running systems and tries to predict the files' popularity. Scarlett allocates the available disk space budget to the popular files using two main approaches. The first one is called the priority approach where Scarlett sorts the popular files according to their sizes and replicates them one by one until it runs out of the disk budget. The intuition behind the priority approach is that the files with a large size are accessed more often compared to the small files. However, the priority approach distributes the disk budget over a small number of files. Alternatively, the round-robin approach increases the replication factor of each file by at most one in each iteration and continues to iterate over the file list until the budget runs out. This approach distributes the budget as many files instead of allocating all the disk space budgets to a small number of large files. Regarding the replica placement, Scarlett distributes the blocks of each popular file over as many racks as possible to ensure spreading the load uniformly across all the machines and racks. During the replication process, Scarlett tracks the load of each rack and each machine to control the placement of each block. For de-replication, Scarlett deletes the blocks of the unpopular files in a lazy manner by overwriting them when another block needs to be written on the disk.

The adaptive data replication for efficient cluster scheduling (DARE) scheme was proposed for HDFS [2]. It made the assumption that the origination of any remote data access is worth replicating with a certain probability value without additional network cost. It aids the scheduler for the achievement of better data locality with the replication of data blocks into remote nodes using the budget of disk space. It did not depend on the scheduler and could work with any scheduler of hadoop for the improvement of the data locality. It follows a greedy approach to assign the disk space budget to the replicas. The greedy approach assumes that any block that is requested remotely from a remote map task should be replicated. However, this approach leads to poor locality when unpopular blocks are replicated. Hence, DARE adopts a randomized approach such that the toss of coin is applied for making the decision if the remote data block should be replicated or not. This approach helps to decrease replicating the unpopular blocks; however, it may miss replicating popular blocks as well. The approach requires a careful adjustment of the probability threshold.

Jungha Lee, JongBeom Lim; [49] proposed a data replication scheme (ADRAP) that is adaptive to overhead, associated with the data locality problem. The algorithm works based on access count prediction to reduce the data transfer time and improves data locality thereby reducing total processing time. Maintaining the larger replication factor than the current access count for a data file does not always give the guarantee for the increment of data locality for all data blocks. To determine the number of replicas, an approach for predicting the next access information is needed. To accomplish this work, the amount of changes of access counts with time can be expressed as a mathematical formula. However, because the access for a data file can be made at random, a constant function is inappropriate. Therefore, we adopt Lagrange’s interpolation using a polynomial expression to obtain a predicted access count for a data file. Each time access is made for a data file, the algorithm determines whether the data file will be replicated or it will be used as cache, by comparing the predicted access count with the number of replicas. In addition, to effectively reduce the number of data nodes with rack-off or rack locality, the adaptive data replication scheme uses the replica placement algorithm that chooses the nodes where the replica will be placed. When replicating a data block, in turn, it traverses the circular linked list of racks to check whether the rack has the data block or not. If the rack has the data block, it traverses the next rack in the circular linked list of racks until it finds the rack that does not have the data block. If it cannot find a satisfied rack after traversing all the elements in the circular linked list of racks, it replicates the data block to the rack which it selected first. Conversely, if the rack does not possess the data block, it selects a node whose number of data blocks is minimal, and then replicates the data block to the node. With the replica placement algorithm, the data blocks to be replicated will be distributed evenly throughout the nodes. In addition to this, the algorithm will reduce the number of tasks with rack-off locality effectively.

In PHFS [43], the authors proposed a data placement scheme that balances the data load, considering the processing speed of nodes. PHFS provides the initial data placement and data redistribution algorithms to improve data locality in heterogeneous cluster environments. In PHFS, however, the performance is dependent on applications because it considered data locality on scientific applications only. As far as data locality is concerned, it is more important to consider applications that share data across the nodes in the system.

3.13.2 Locality-Aware Scheduling

To illustrate the conflict among data locality and fairness in scheduling, Zaharia.M, Borthakur.D; [82] introduced a delay scheduling approach by introducing a fair scheduler for hadoop cluster that has 600 nodes. It performs the scheduling of jobs based on the fairness and wastes a few time periods for permitting other jobs to launch the tasks. It raises throughput into to two times in preservation of the fairness and gains the optimum data locality in various workloads. The approach is useful among various scheduling policies over fair sharing as like the hadoop fair scheduler. Hadoop fair scheduler has two main objectives: data locality and fair sharing. To gain this objective, the scheduler performs the reallocation of resources among jobs as the amount of jobs varies by waiting for operating tasks to complete and eliminating operating tasks to provide space for the new tasks. It operates well in hadoop workloads and is useful over fair sharing. The generalization of delay scheduling in HFS is taken for the implementation of a hierarchical scheduling policy with the requirement of users. It split slots among users according to weighted fair sharing at top-level and permits users for scheduling of their own jobs using either fair sharing or FIFO.

Zhenhua et al. formulated the MapReduce data locality problem as a mathematical model that is used to find the optimal scheduling that maximizes the data locality [31]. It shows that scheduling multiple tasks all at once outperforms the delay scheduling approach, where the scheduling is performed task by task. Delay scheduling assigns the tasks one by one without considering the impact of this assignment on the other tasks. To reach the global minimum of data transfer over the network, a scheduling approach should calculate the cost of each assignment and the impact of the other tasks.

The authors introduced scheduling approach based on data locality for non-homogeneous environments [83]. Data transfer time and estimation of waiting time was applied for scheduling the tasks. It makes dynamic determination of whether scheduling of the task to the requesting node with transferring the data to the requesting node or reservation of the task for the stored node.

The authors proposed scheduling algorithm for map tasks designed with the policies of the maxweight and the join the shortest queue and introduced the new queueing model [75]. Firstly, an outer bound was set at the capacity portion of a mapreduce cluster based on data locality and this capacity portion contains all arrival rate vectors for the existence of scheduling algorithm which provides the stability for the system. In this new queueing model, each device has one local queue, a common queue for all devices and that devices store local tasks. According to this new queueing model, a two-stage scheduling algorithm was introduced under that routing of a new incoming task with one of the three local queues or the common queue applying the policy of the join the shortest queue; if a device has availability, a task from its local queue or the common queue applying the policy of the maxweight is selected. We proved that the maxweight and joint JSQ scheduling algorithm has throughput optimality, so that in the exact outer bound of this capacity portion, it has stability on any arrival rate vector and which also proves that the coincidence of the actual capacity portion with the outer bound. We remarked that the existing outcomes of maxweight scheduling algorithms made the assumption of geometrically distributed service time or deterministic service time with tasks preemption.

The stability of maxweight scheduling with random processing time and non-preemptive task execution has not been accomplished before. Moreover, the optimality of throughput, the number of backlogged tasks were studied, that has the direction relation with the performance delay according to Little’s law. We took the consideration of the event that the assumption of a heavy local traffic condition and the service times that have the nature of geometric distributions. Then, the maxweight and joint JSQ scheduling algorithm is showed for heavy-traffic optimality, so that the number of backlogged tasks was reduced when the boundary of the capacity portion approaches the arrival rate vector. Therefore, the proposed system performed the optimal balance between load balancing and data locality and was both delay and throughput optimality in the heavy-traffic condition.

A flexible and powerful framework is proposed for scheduling fine-grain resource sharing with concurrent distributed jobs [36]. The problem of scheduling is depicted with a graph data structure, at where capacities and edge weights compress the competitive requests of fairness, freedom of starvation and data locality and the optimum online schedule is calculated by a standard solver based on the global cost model. The implementation of this framework was evaluated at Quincy, on a cluster of a few hundred computers using various workload of CPU and data intensive jobs. We performed the evaluation of Quincy against an existing queue-based algorithm and various policies for each scheduler was implemented, with and without fairness constraints. Quincy achieved better fairness as fairness is demanded, while substantially increasing data locality.

Although there have been many methods for the improvement of data locality, most of them either ignored global optimization and were greedy, or suffered from the complexity of high computation. To solve these issues, a scheduling algorithm for heuristic task, balance-reduce (BAR), was presented [39]. Firstly, a task allocation was proposed, and then the completion time for job can be decreased gradually with tuning of the initial task allocation. Data locality can be adjusted dynamically depending upon cluster workload and network status.

A new approach was proposed for mapreduce clusters in order to improve data locality [33]. The aim of this approach was to provide a fair chance to every slave node for grabbing local tasks before assignment of slave node with non-local tasks. As it tried in order to find a matching, so that a slave node including incoming data, with every map task that has no assignment. Firstly, the matchmaking algorithm provides freedom of the strict order of job order for task assignment as like the delay scheduling algorithm. If there was no a local map task at the first job, the continued searching of succeeding jobs will be performed by the scheduler.

Second, to provide a fair chance to every slave node for grabbing its local tasks, as a local task could not be found by a node in the queue at start time in a row, there is no assignment for the node to non-local map task. So that, no map task is achieved by the node in this heartbeat period. In a heartbeat period, local task assignment was considered for all free slave nodes and their heartbeats have likely given by these free slave nodes, as a local task could not be found by a node in the queue at later time in a row in order to avoid inefficient spending of computing resources, the assignment of a non-local task with the node would be performed by this matchmaking algorithm. In this way, both higher cluster utilization and higher data locality rate were achieved by our algorithm. A locality marker was provided to each slave node for marking its conditions. If jobs at the queue did not have local map task with a slave node, according to the marked value of that slave node, whether or not the assignment of a non-local task with that slave node would be determined by this algorithm. Third, one slave node was allowed for taking at most a non-local map task by this matchmarking algorithm every heartbeat period. Finally, locality markers of all slave nodes would be deleted as the adding of a new incoming job to the job queue was taken. As new local map tasks at some slave nodes may be comprised by a new job, according to the arrival rate of new job, all nodes' conditions were reset and the matchmaking process of all to all task to node was started again.

Various studies of hadoop pointed out that, separation from the phase of shuffling, the huge amount of map task operations for remote data was the other origin of massive network traffic. These consequences occurred an unbalanced operations of map tasks and a massive amount of inefficient map tasks operations among various devices. Those factors led a noticeable degradation of system performance. Hence, Maestro, the scheduling algorithm for map tasks, was introduced for the improvement of overall system performance in the mapreduce operation [35]. The map tasks was scheduled by this algorithm in two waves: firstly, free slots of each node was filled according to the hosted amount of map tasks and the replication plan; second, the possibility of scheduling a map task at a specified device was taken into account by runtime scheduling based on the replication degree of incoming data. At the shuffling phase, the more balanced in data distribution and the improvement of locality at the map tasks operations were achieved with these two waves.

Many current schedulers omitted data locality for reduce tasks when the intermediate data was fetched although data locality issues have been considered for map tasks. As a consequence, it led to degradation of system performance. Therefore, recently, the issue of decreasing the fetching cost of reduce tasks has been specified. But, the introduced schemes are purely relied on the greedy strategy, depending upon the suspicion for assigning the slots with reduce tasks that slots are closest with the recent produced intermediate data. As a consequence, in the existence of job arrivals and departures, assignment of the reduce tasks of the current job to the nodes with the lowest fetching cost can preclude a subsequent job with even achieving improved data locality from being launched on the recent slots. At last, a stochastic optimization framework was formulated for achieving the improved data locality for reduce tasks, with the suitable assignment policy showing a threshold-based structure [72]. For the easier implementation, a receding horizon control policy was introduced depending upon the optimal key in restricted conditions.

The authors provided the first complete theoretical data locality analysis of the Map phase of MapReduce, and more generally, for bag-of-tasks applications that behaves like MapReduce [10]. We show that if tasks are homogeneous (in term of processing time), once the chunks have been replicated randomly on resources with a replication factor larger than 2, it is possible to find a priority mechanism for tasks that achieves a quasi-perfect number of communications using a sophisticated matching algorithm. In the more realistic case of heterogeneous processing times, we prove using an actual trace of a MapReduce server that this priority mechanism enables to complete the Map phase with significantly fewer communications, even on realistic distributions of task durations.

Although existing hadoop schedulers are quite successful, there still have issues to be solved for the optimal joint improvement for map tasks and reduce tasks, albeit there is a strong dependence between them. This can lead to unfavorable data locality and job starvation. A scheduler for hadoop based on resource was proposed and evaluated [73]. It paired the advances of map and reduce tasks, applying random peeking scheduling and wait scheduling at map and reduce tasks to achieve the optimal joint task assignment. This improved the overall data locality and eliminated the problem of starvation.

In order to achieve better load balance, a work stealing approach based on data was introduced and there has still tries for getting best data locality [76]. Basically, both shared task ready and dedicated queues are kept by each scheduler and these queues were evaluated as descending order of priority queues according to the size of data that a task requires. During the migration of tasks at the shared queue was done on schedulers to balance workloads by stealing of work, the scheduling and execution of tasks at the dedicated queue was done locally with no applying special policy. One ready task would be placed in the queue according to location and size of requested data of task. Moreover, the pushing of a task at others might be done by the scheduler if the source of requested data is remote data. It performed more than many current stealing works that applies a ready queue based on locality of task and these queues were evaluated as double-ended deque or normal queue. The distributed key-value store was utilized as the service of metadata for keeping all tasks 'efficient data locality information. This proposed approach performed well in not only heterogeneous but also homogeneous environments.

3.13.3 Locality-Aware Prefetching and Pre-shuffling

Because devices in mapreduce clusters are large-sized memories, that are often underutilized, prefetching of incoming data to memory was an efficient technique for the improvement of data locality. But it still had key issues for designers of clusters on when and what to prefetch. To achieve the efficient usage of prefetching, the high-performance scheduling optimizer (HPSO), that is also the scheduler for data prefetching, was designed for the improvement of data locality in jobs of mapreduce [70].

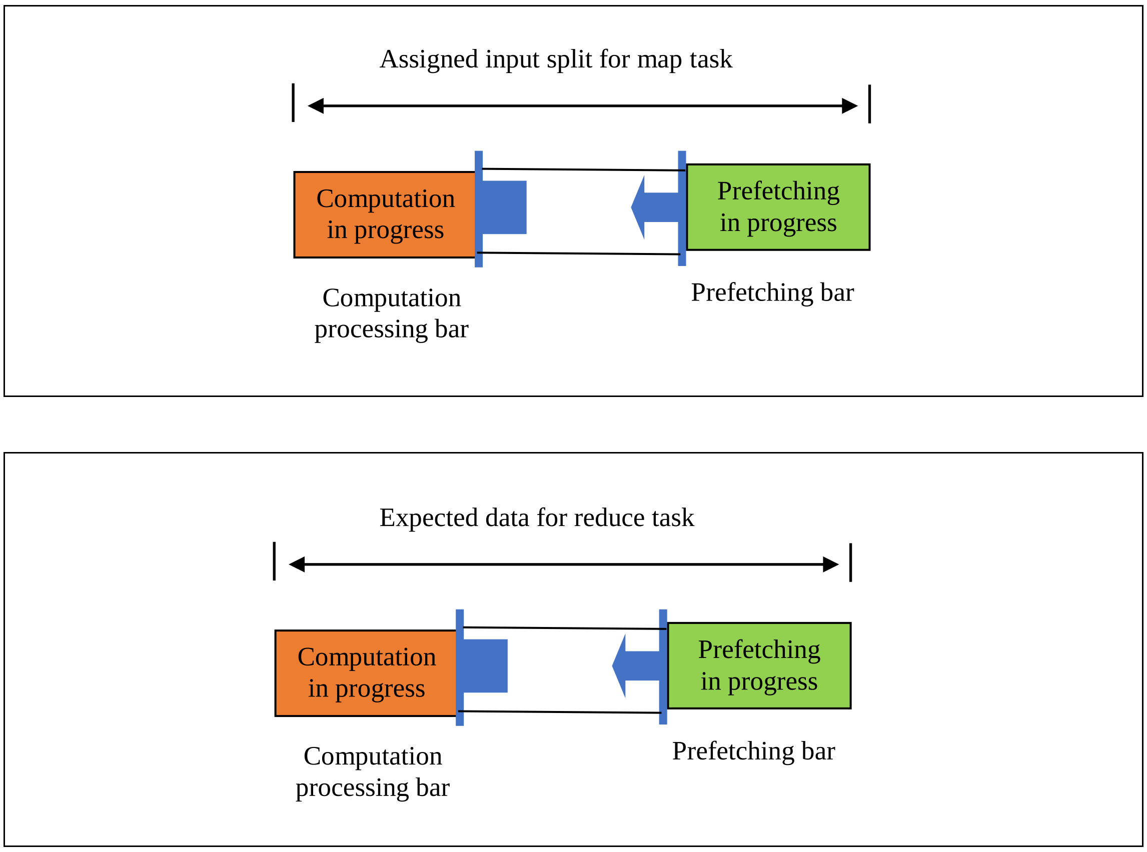

Figure 3.4 The intra-block prefetching

The principal concept was the prediction of the most suitable data nodes for the future assignment of map tasks and the prefetching of the incoming data at memory with no delay on setting new tasks. Sangwon Seo, and Ingook Jang; [67] proposed optimization schemes such as pre-shuffling and prefetching to solve the problems of sharing. The implementation of these two schemes were performed at high performance mapreduce engine (HPMR). The scheme of prefetching could be divided into two kinds: prefetching of inter-block and intra-block. The entire data block was prefetched at inter-block prefetching as only an intermediate output or an input split was prefetched at intra-block prefetching. Those prefetching schemes were utilized for all phases of map and reduce.

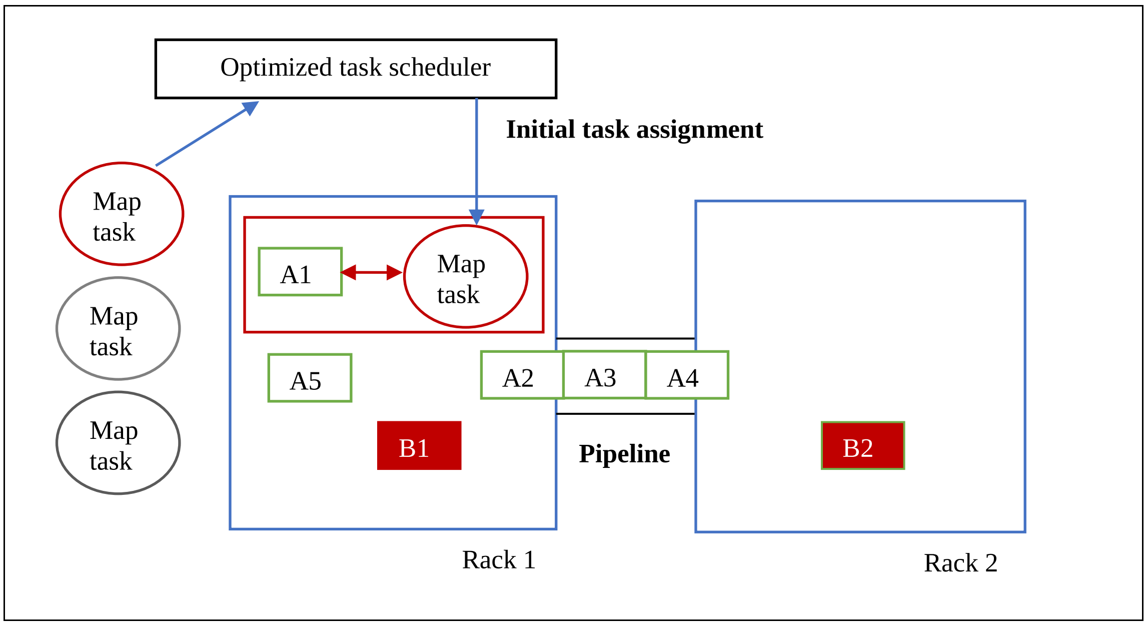

Figure 3.5 The inter-block prefetching

Figure 3.4 and Figure 3.5 describes the intra-block prefetching and the inter-block prefetching. The amount of intermediate output for shuffling was reduced with the pre-shuffling scheme. At pre-shuffling, HPMR searched the incoming block before the starting of the map phase and predicted the targeted reducer at which the partition of key-value pairs was done. If the splitting of key-value pairs of intermediate result at the local storage was performed, the amount of shuffling operations across the network could be decreased. In this system, the task scheduler only for pre-shuffling at reduce phase was proposed. Briefly, this scheme achieved better data locality, and the reduced amount of the shuffling overhead at reduce phase. The pre-shuffling scheme is presented in Figure 3.6.

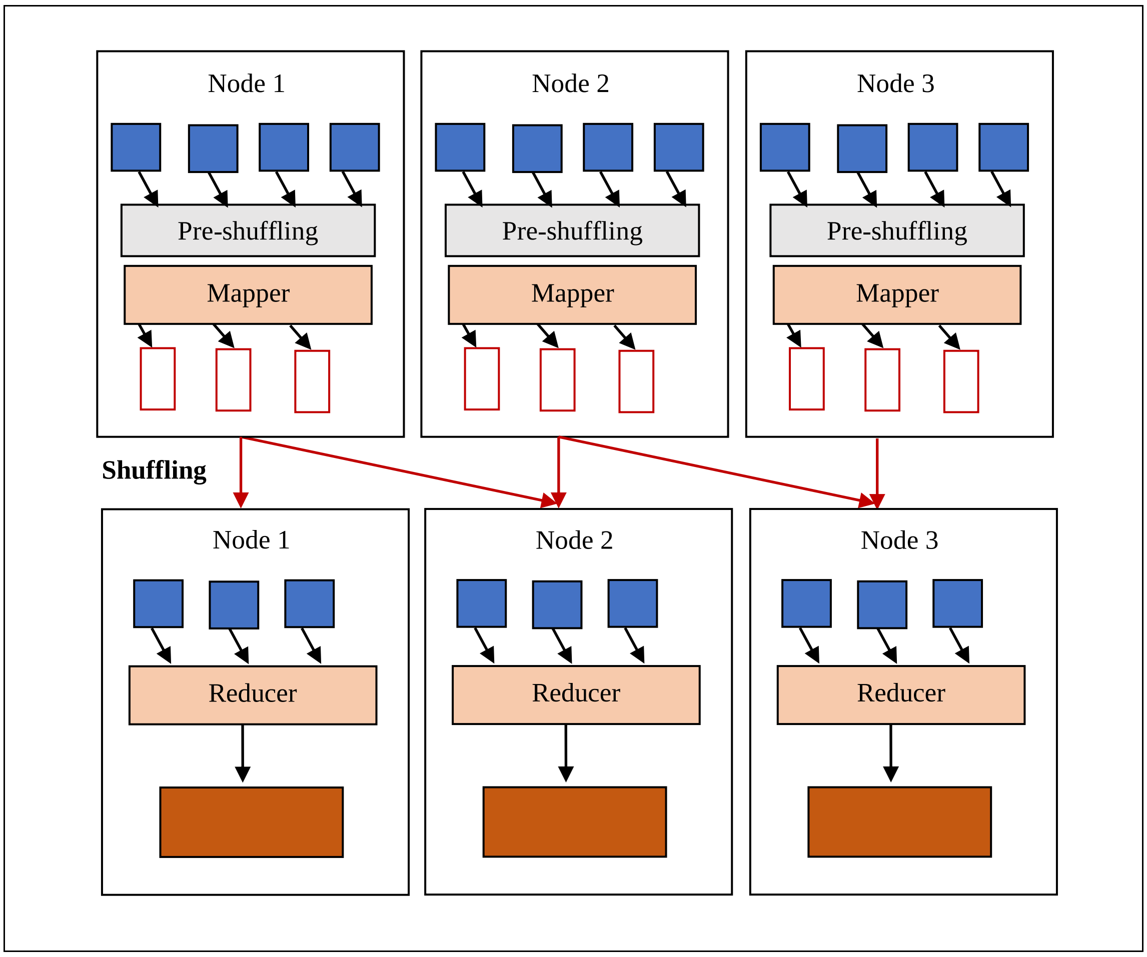

Figure 3.6 Pre-shuffling

Kousiouris et al. [45] studied multiple time steps ahead prediction of files based on fourier series analysis and used prediction results to determine replication factor that balanced the increment in availability and disk usage.

3.14. Summary

In this chapter, different strategies and models of data replication and data locality are reviewed. Among replication strategies, static method is more common in today cloud storage system because of simple and straightforward technique. However, it results more storage cost and less availability in very large storage systems as some data files may not need as many as static replication factor due to lack of usage. At the same time, some have to be replicated more than static replication factor to recover highly concurrent access. As a result, dynamic replication becomes an important strategy to cope the weakness of static method. Therefore, most parts of this chapter present different approaches of dynamic replication which are intended to play a vital role in today cloud storage systems. In the following chapters, approaches to dynamic replication are formulated and evaluated in various environments. Also, in this chapter, several research areas have been studied to improve the performance of data locality and evaluated their research outcomes in various environments such as dedicated and shared environment.

CHAPTER 4

DYNAMIC REPLICATION MANAGEMENT SCHEME FOR EFFECTIVE CLOUD STORAGE (ECS)

Replication is one of the important roles in cloud storage to improve data availability, fault tolerance and throughput for users and control storage cost. As data access pattern changes every time, the nature of popular files is unpredictable and unstable. Therefore, data popularity is taken into account as an important factor in replication. Data popularity in replication impacts an efficient storage because it is able to reduce waste storage for unpopular files. Also, data locality is a key issue in storage system and this consequence occurs performance overhead of system. This chapter presents a dynamic replication management scheme for effective cloud storage (ECS). The system contains two portions; replica allocation and replica placement.

In the first portion, replica allocation, popularity is taken into account by analyzing the changes in data access pattern. Second, for replica placement, replicas are placed and performed on dedicated assigned nodes in order to enhance data locality. The proposed placement algorithm is able to avoid the overloaded problem of nodes by considering the load of nodes; i.e, disk utilization, CPU utilization and adjustable disk bandwidth. The contributions of this proposed system are as follows:

1. The rate of change of file popularity in timeslots is analyzed by applying first order differential equation.
2. Determination of the decrement and increment of the number of replicas for each file is computed.
3. While the replicas are placed into nodes, the load of nodes such as disk utilization, CPU utilization and bandwidth utilization are considered.
4. The predefined threshold is used to compute the overloaded condition of cluster.
5. If the overloaded condition of that assigned nodes occurs, proposed replica replacement algorithm is used.
6. This proposed replacement algorithm considers not only outgoing blocks but also the access frequencies for blocks.

The basic idea of replication is based on the different replication degree per data file. Keeping the fixed number of replicas causes wasteful storage for unpopular data and inefficiency for popular data. Also, maintaining too much replicas than current access count for a file does not always guarantee better locality for all blocks. Figure 4.1 presents the proposed system flow diagram for ECS. The objective of this system is to propose a replication strategy in order to achieve the improved data locality by more replicas for popular data while maintaining less replicas for unpopular data.

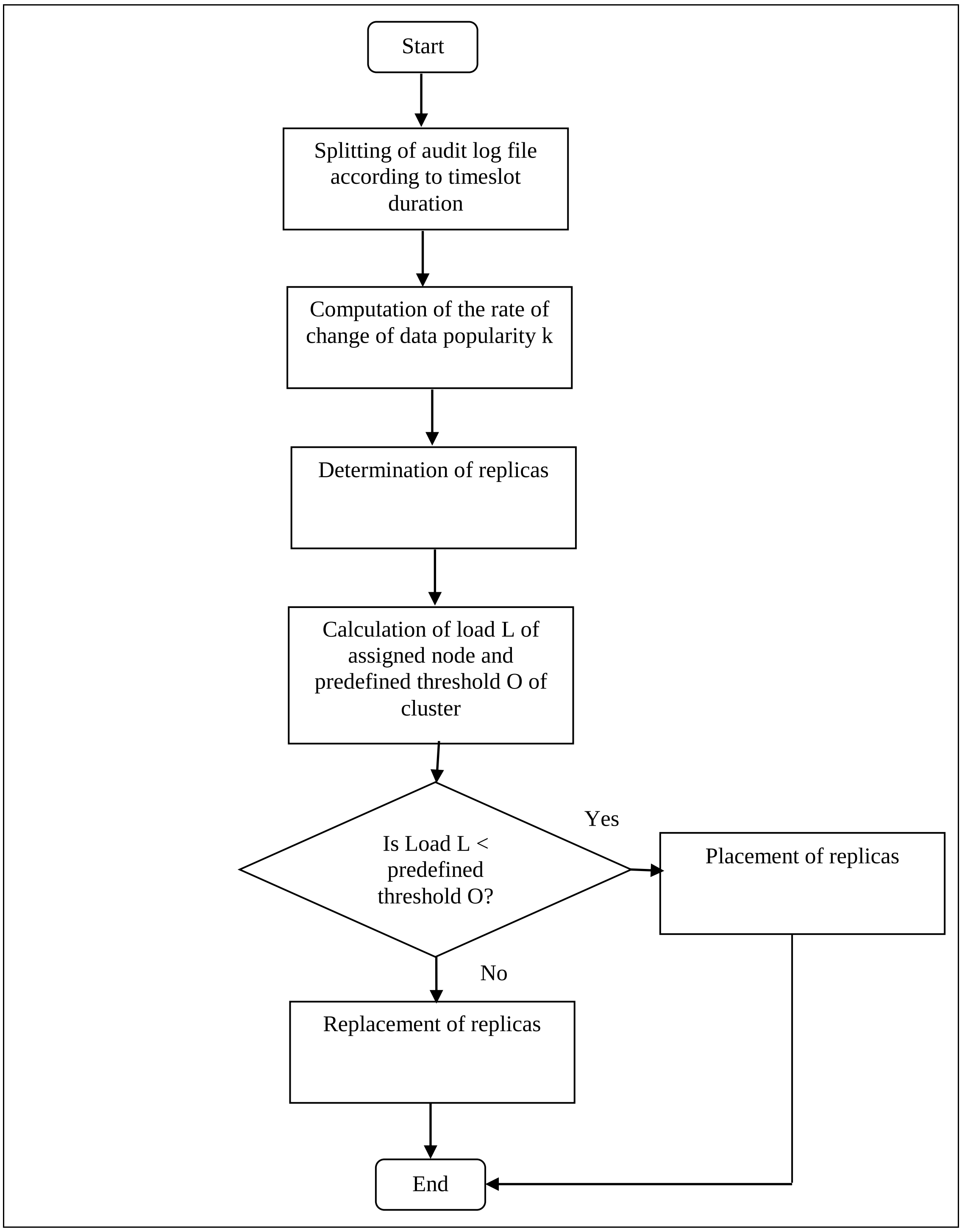
Start
Splitting of audit log file according to timeslot duration
Computation of the rate of change of data popularity k
Determination of replicas
Calculation of load L of assigned node and predefined threshold O of cluster
Is Load L < predefined threshold O?
Yes
Placement of replicas
No
Replacement of replicas
End

Figure 4.1. System Flow Diagram

4.1 Replica Allocation

At first step, first order differential equation is applied to compute the rate of change of file popularity. Pop-Store and LALW algorithms utilized the concept of half-life that denotes the weight of the access information at a period degrades to half of the weight of the last period. The assumption of popularity is that the popularity of an item which grows at a definite period has the relationship with total popularity of that item at that period. Mathematically, this assumption can be expressed as

$$\frac{dP}{dt} = kP(t)$$ Equation (4.1)

where k is the growth or the decay constant and P(t) is popularity at time t. If k is less than 0, there is decay and else if k is greater than 0, there is growth, and so on. Then, this linear differential equation is resolved into

$$P(t) = P_0 \, e^{kt}$$ Equation (4.2)

Then,

$$k = \frac{Ln(\frac{P(t)}{P_0})}{t}$$ Equation (4.3)

Where P_0 is the starting popularity, i.e. p (0) = P_0. The Yahoo HDFS user audit log format is shown in Figure 4.2.

```
2019-06-17 11:11:59,693 INFO
org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit: ugi=hadoopuser
ip=/132.42.210.34  cmd=delete src=/app/hadoop/temp/test1.txt  dst=null
perm=null
```

Figure 4.2. Yahoo HDFS User Audit Log Format

The input log file is broken into smaller files based on timeslot in order to compute the access frequency information of each file. And, the extraction of fields such as date, time and src from this log file is performed. Then, from the src link shown in Figure 4.2, access frequency is counted and kept for each file in each timeslot. The extracted grouping result can be viewed in Table 4.1.

Table 4.1. Example of Grouping Result of Frequency Counts for one Timeslot

Date	File path	Access Frequency Count
2019-06-17	src = abc/cde/a	3000
2019-06-17	src = abc/cde/b	2000
2019-06-17	src = ggh/ccd/dd	600

However, as data popularity is based on access history, it still needs to combine access counts on previous timeslots. Therefore, the resulted formats such as Table 4.1 are combined into aggregated frequency counts to process data popularity. The aggregated format is shown in Table 4.2. And, the computation of the rate of change of file popularity for individual files is done in each timeslot according to Figure 4.3 and Table 4.3.

Table 4.2. Aggregated Frequency Counts for Three Timeslots

Date	File path	Access Frequency Count		
		Timeslot 1	Timeslot 2	Timeslot 3
2019-06-17	src = abc/cde/a	3000	6088	370
2019-06-17	src = abc/cde/b	2000	1800	300
2019-06-17	src = ggh/ccd/dd	600	700	670

Table 4.3. Notations Used in File Popularity Algorithm

Notation	Description
$P(t_f)$	The values of popularity of file f
$AF(t_f)$	The total access frequency counts of file f at each timeslot
logFile	The audit log file
k	The rate of change of file popularity

```
Algorithm 4.1: File Popularity Algorithm
Input: inLog
Output: $k$
1.  Read logFile
2.  Compute the access frequency of each file by using
        $P(t_f) = AF(t_f), \forall f \in F$
3.  Compute the rate of change of file popularity $k$ of each file with the substitution of
    $P(t) = P(t_f)$ in Equation (4.3)
4.  return $k$.
```

Figure 4.3. File Popularity Algorithm

In order to verify the proposed rate of change of file popularity algorithm, three files are supposed (x1, x2 and x3) in three time slots. Each time slot duration is set as 10 seconds, therefore, (t1= t2 = t3= 10 seconds). Let P0 = 1, P (t) = AF(t_f) and calculate k by using Equation (4.3). Suppose that access frequencies of file x1, x2 and x3 in time slot 1 are 40, 1100 and 200. In time slot 1, the growth rate k in file x1, x2 and x3 is 0.3688, 0.7003 and 0.5298. Also, in time slot 2, access frequencies of file x1, x2 and x3 are 400, 100 and 900. Therefore, the growth rate k in file x1, x2 and x3 for time slot 2 is 0.5991, 0.4605 and 0.6802. Also, in time slot 3, access frequencies of file x1, x2 and x3 are 2200, 1200 and 20. Therefore, the growth rate k in file x1, x2 and x3 for time slot 3 is 0.7696, 0.7090 and 0.2996.

At second stage, the number of replicas for each file is defined using changes of file popularity that is the outcome of the first stage. Initially, existing replicas will be assumed as 3 as like the default replica of HDFS. If k is less than 0.0, then existing replicas is decreased by 1. If k is greater than 0.0, then existing replicas is increased by 1. If k is equal to 0.0, then existing replicas is unvaried. Otherwise, if it is new file, then existing replicas is determined 3 as like the default replica of HDFS. The process of the number of replicas calculation is shown in Equation 4.4 and Figure 4.4.

$$noOfReplica = \begin{cases} noOfReplica++, & k > 0 \\ noOfReplica--, & k < 0 \\ noOfReplica, & k == 0 \\ 3, & file\ is\ new \end{cases}$$

Equation (4.4)

Algorithm 4.2: Calculation of the Number of Replicas

```
Begin
        If k > 0.0 then
            noOfReplica++
        Else if k < 0.0 then
            noOfReplica--
        Else if k == 0.0 then
            noOfReplica remain unchanged
        Else file is new then
            noOfReplica = 3
        End If
        Return noOfReplica
End
```

Figure 4.4 Replica Allocation Algorithm

4.2 Replica Placement

The replica placement is one of the principal key problems for replication management in cloud storage. After the calculation of data replicas for each file in previous section, the last one is to assign the replicas into nodes effectively. The replica placement is an important issue for gaining the improvement of load balancing and data locality in cloud storage. If the replica is placed in suitable nodes, data locality and load balancing can be improved. At this step, replicas are placed into assigned nodes to achieve greater data locality. We will make the assumption that the incoming jobs must have to access these replicas at next timeslot. The entering job is split into tasks and assignment of task with nodes in the cluster is performed. Each input block has one map task. It is assumed that one data block represents one data file. We will let that maximum replicas are total nodes in the cluster and minimum replicas is 1. Node locality of task is checked and if there has node locality, then placement of task at that assigned node is performed. If the condition, that is, lack of replica at computing node for map task will occur, prefetching needed replica block into this node. In this system, the load of assigned node is considered to avoid overloaded condition while loading into assigned nodes. That replica is loaded if the load of assigned node is less than predefined threshold. Otherwise, replacement of needed replica block with existing block at assigned node is performed.

The default placement policy of Hadoop is randomness and it assumes that all nodes within cluster have equality condition. Moreover, it does not consider utilization of nodes in placement. This condition results in imbalance load to Hadoop. The proposed system considers inequality condition of nodes within the cluster. In this system, we consider disk utilization, disk bandwidth and CPU utilization as the load of nodes. Then, the disk utilization of the node is carried out as

$$U(D_i) = \frac{D_i(use)}{D_i(total)} \qquad \text{Equation (4.5)}$$

Where, $U(D_i)$ is the disk utilization of the ith DataNode, $D_i\ (use)$ is the utilized disk capacity of the ith DataNode and $D_i\ (total)$ is the total disk capacity of the ith DataNode. Then, the disk bandwidth of the node is carried out as

$$BW(D_i) = \frac{T_b}{T_s} \qquad \text{Equation (4.6)}$$

Where, $BW(D_i)$ is the disk bandwidth of the ith DataNode, T_b is the total amount of bytes transferred and T_s is the total time taken between the first request for service and the completion of the last transfer. Then, the adjustable disk bandwidth of node for load factor is considered as

$$ABW(D_i) = \frac{BW(D_i)}{Total_i\ (BW)} \qquad \text{Equation (4.7)}$$

Where, $ABW(D_i)$ is the adjustable bandwidth of the ith DataNode and $Total_i\ (BW)$ is the total bandwidth of the ith cluster. Then, the CPU utilization of the node is carried out as

$$CU(D_i) = 100\% - (\%\ of\ time\ that\ is\ spent\ in\ idle\ task) \qquad \text{Equation (4.8)}$$

Where, $CU(D_i)$ is the CPU utilization of the ith DataNode. To compute the load factor of assigned node, $\propto$, β and γ are set as the coefficients of storage utilization, disk bandwidth and CPU utilization. Then, the load factor of the node is carried out as

$$Load(D_i) = \propto U(D_i) + \beta\ ABW(D_i) + \gamma\ CU(D_i) \qquad \text{Equation (4.9)}$$

To compute the overloaded condition of cluster, the sum of maximum disk utilization, maximum disk bandwidth and maximum CPU utilization in cluster is divided by the number of nodes in the cluster is defined as the predefined threshold O_i of the ith cluster. Therefore, the predefined threshold O_i of the cluster C_i is carried out as

$$O_i = \frac{Max_i(U) + Max_i\,(ABW) + Max_i\,(CU)}{N} \qquad \text{Equation (4.10)}$$

That replica is placed at that node if the load of assigned node is less than predefined threshold. Otherwise, replacement of needed replica block with existing block at assigned node is performed. The proposed replacement algorithm is based on the concept of least recently used (LRU) [74]. It outperforms efficiently and is more reliable than LRU because it takes into account not only outgoing blocks but also access frequencies for blocks in replacement. In the proposed data replacement algorithm, the block that has minimum access frequency is considered as first for replacement. If there is one or more blocks that have minimum access frequencies, outgoing block (least recently accessed block) is determined for replacement according to the concept of LRU mechanism. The proposed replacement algorithm is described in Figure 4.5 and the proposed data placement algorithm is in Figure 4.6 and Table 4.4.

Algorithm 4.3: Proposed Replacement Algorithm

Step 1: It computes total access frequencies of all blocks at that assigned node as the replica is loaded into the assigned node.

Step 2: That replica is selected to evict from the node if only one block that has minimum access frequencies is found.

Step 3: If there have more than one block that have minimum access frequencies are found, outgoing block is chosen to remove from that assigned node as LRU.

Figure 4.5 Proposed Replacement Algorithm

Table 4.4. Notations Used in Data Placement Algorithm

Notation	Description
DN	DataNodes List
ABW	Adjustable Bandwidth
U	Disk Utilization
RP	Replica List
MT	Map Task List
CU	CPU Utilization
C	Cluster List
LF	Load Factor List
O	Predefined Threshold of the Cluster

Algorithm 4.4: Proposed Placement Algorithm

```
Input: DataNodes List DN= {$DN_1$, $DN_2$,.., $DN_n$ }, Replica List RP ={ $RP_1$, $RP_2$, $RP_3$,…., $RP_n$ }, Map Task List MT = {$MT_1$,$MT_2$,$MT_3$,…,$MT_n$}, Load Factor List LF = {$LF_1$,$LF_2$,$LF_3$,…., $LF_n$}, Predefined Threshold $O_i$, Cluster List C = {$C_1$, $C_2$, $C_3$,…., $C_n$}
Output: DataNodes List DN
for each incoming map task MT do
    for each DataNode DN do
        Check node locality of task $MT_i$
        if there is node locality then assign task $MT_i$ to that DataNode $DN_i$
        else
            Perform remote data replica retrieval for task $MT_i$
            Calculate storage utilization U of this assigned DataNode $DN_i$ using Equation (4.5)
            Calculate adjustable disk bandwidth ABW of this assigned DataNode $DN_i$ using Equation (4.6) and (4.7)
```

```
                Calculate CPU utilization **CU** of this assigned DataNode $DN_i$
                using Equation (4.8)
                Calculate load factor $LF_i$ for this assigned DataNode $DN_i$ using
                Equation (4.9)
                Calculate predefined threshold $O_i$ for the cluster $C_i$ using
                Equation (4.10)
                if $LF_i$ > threshold $O_i$ then
                    Perform replacement by using Algorithm 4.3
                    Place replica $RP_i$ for this task on that DataNode $DN_i$
                    break
                else
                    Place replica $RP_i$ for this task on that DataNode $DN_i$
                    break
                end if
            end if
        end for
end for
```

Figure 4.6 Proposed Placement Algorithm

4.3 Summary

In this chapter, replication algorithms are proposed for replica allocation and replica placement. To test actual situations, Yahoo web log data set is used to apply as data access pattern, which is the critical input for the proposed algorithm. In the first portion, replica allocation, popularity is taken into account by analyzing the changes in data access pattern. Second, for replica placement, replicas are placed and performed on dedicated assigned nodes in order to enhance data locality. The proposed placement algorithm is able to avoid the overloaded problem of nodes by considering the load of nodes; that is, disk utilization, CPU utilization and adjustable disk bandwidth.

CHAPTER 5
IMPLEMENTATION OF ECS

This chapter describes the proposed dynamic replication management scheme for cloud storage (ECS). The proposed replication scheme is based on individual file and replication factor is considered upon the popularity of each file. CloudSim [13][86]is applied as the simulation environment to perform the evaluation of the proposed replication scheme. This simulator has the discrete event for providing the simulation and modeling of the components of cloud like hosts, RAM, VMs, internal network topology, data centers, CPU components, power aware provisioning policies and storage. Since there is the ability of extensible, the modification and customization can be easily made by the extension of the class, making a little adaptation to its core component. There have restrictions for disk I/O processing although fundamental components like SAN storage, files and hard drives are provided for the simulation of cloud data storage.

The additional modeling of disk I/O processing and CPU processing for operation of jobs is extended by CloudSimEx [30][87]. CloudSimEx combines the disk I/O processing modules in the CloudSim simulator. The extensions of some native class of CloudSimEx have been performed for the simulation of the proposed replication scheme.

A replication module has extended in CloudSim simulator for the simulation of HDFS environment such as heartbeat mechanism for monitoring the utilization of the datanode and metadata management on the namenode. The simulator for HDFS environment designed in [24] is more closer to the original system. This simulator has the discrete event and is implemented with java language like CloudSim. Hence, some functions are extracted from that simulator and combined to CloudSim with a few changes. The main classes for the implementation of replica management are:

a. Replication Scheduler: This entity performs the replication by accessing the replica catalog. The management of replication scheduling and maintenance of metadata is done by the namenode in the actual system of HDFS. This is not deployed within a node as it is a separate entity.
b. Heartbeat: This entity describes the heartbeat mechanism in the HDFS that periodically sends the signal to the namenode to inform the resource

utilization information of the nodes. Before starting the simulation, the initialization of this class needs to be done as like any other entities of CloudSim.

c. Replica Catalog: This entity is responsible for keeping the current location of blocks stored in various datanodes. It also maintains access information of the data blocks for making the determination of data popularity.

This simulation is run for 24-hour period. In the simulation environment, 8 cluster having 50 heterogeneous datanodes are created and the placement of these nodes at these cluster is performed by using CloudSim [13][86] and CloudSimEx [30][87]. The reason behind creating simulation environment that consists of 400 nodes was that the work in [23] showed that there is almost guarantee to occur data loss event if the cluster scales up beyond 300 nodes. At the start of the simulation, the equal distribution of blocks at nodes in the cluster is performed for ease of evaluation and simplicity. The replication element is assumed as one data block and this block represents one file.

5.1 Replication Algorithms in Cloud Storage

Among different replication strategies presented in Chapter 3, static strategy and *LALW* (Latest Access Largest Weight) algorithm are commonly used in cloud data replication for centralized system. In this section, the proposed replication strategy is presented and compared with *LALW* algorithm.

5.1.1 Static Replication

Static replication is the simplest commonly used replication approach in Cloud computing. The number of replicas is preconfigured before the system starts and the system replicates the static number of data whenever data is stored. The popular file systems in today cloud environment such as HDFS and GFS apply this static replication strategy and the default is tri-replication.

5.1.2 LALW Algorithm

LALW (Latest Access Largest Weight) algorithm is widely used in Grid system for dynamic replication. Detailed description of *LALW* algorithm is presented in chapter 3. In *LALW*, only the most popular file is selected and considered to replicate more numbers. Every time interval, it finds out one popular file but not others. Actually, there can be unpopular files which are rarely accessed in the cluster. In the proposed strategy, therefore, different numbers of replicas are considered for different data in every time interval.

5.2 Proposed Dynamic Replication Management Scheme (ECS)

Based on the consideration of cloud data popularity, data locality and the problems of existing approaches, a scheme (ECS) is proposed which is able to adapt the data popularity changes in cloud storage. To evaluate the proposed replication algorithms, Yahoo Audit log dataset [48] is used and the description of dataset is mentioned in Table 5.1. In order to count data access frequency in each timeslot, the dataset is divided depending on date and time. To simplify the analysis, each time slot is defined as 3 minutes period.

Table 5.1 Description of Tested Dataset

Test data description		Yahoo Webscope user audit logs (2010-01-12 00:00:00 to 2010-01-12 00:29:59)
Number of timeslots tested in the algorithm		10
Each Timeslot duration		3 minutes
Total records	**Timeslot 1**	496,845
	Timeslot 2	492,357
	Timeslot 3	536,221
	Timeslot 4	425,188
	Timeslot 5	542,627
	Timeslot 6	580,447
	Timeslot 7	538,569
	Timeslot 8	358,455
	Timeslot 9	107,786
	Timeslot 10	92,255

From these 10 timeslots, timeslot 1, 2 and 3 are set as timeslot 1, timeslot 2, 3 and 4 are set as timeslot 2, and timeslot 3, 4 and 5 are set as timeslot 3 and so on for the computation of the rate of change of data popularity using differential equation. From this dataset, 1000 files are extracted to perform the evaluation of proposed system. At the beginning of simulation, the number of existing replicas is set as 3 for all 1000 files. From Figure 5.1 to 5.10 show the access frequencies of 1000 files in 8 timeslots. According to the analysis output, data access pattern fluctuates in different timeslots. Therefore, we propose differential equation to find out the rate of change of file popularity.

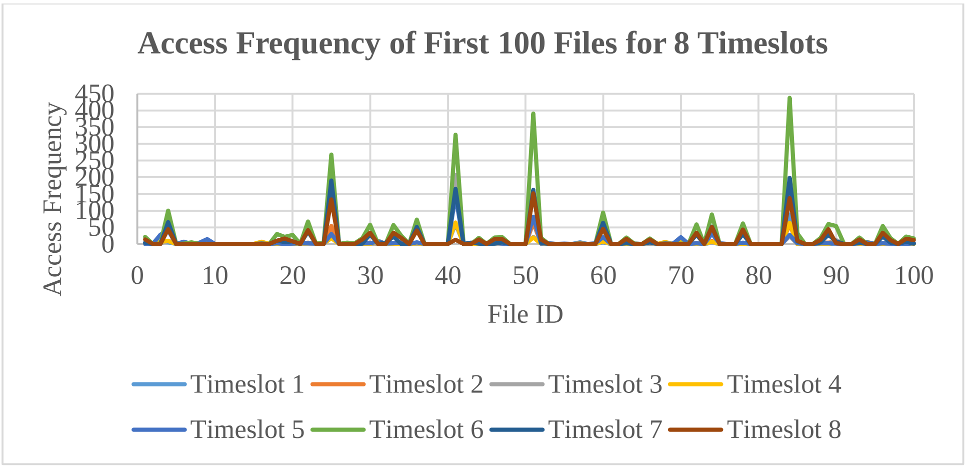

Figure 5.1 Access Frequency of First 100 Files for 8 Timeslots

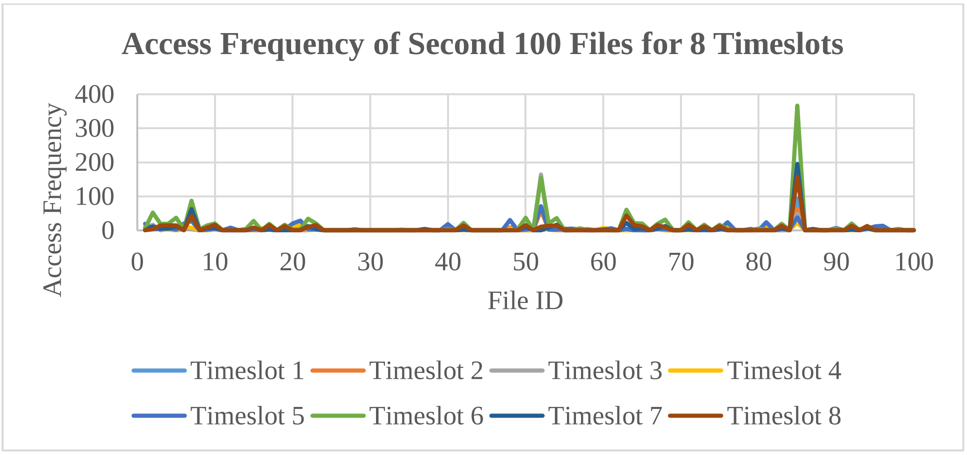

Figure 5.2 Access Frequency of Second 100 Files for 8 Timeslots

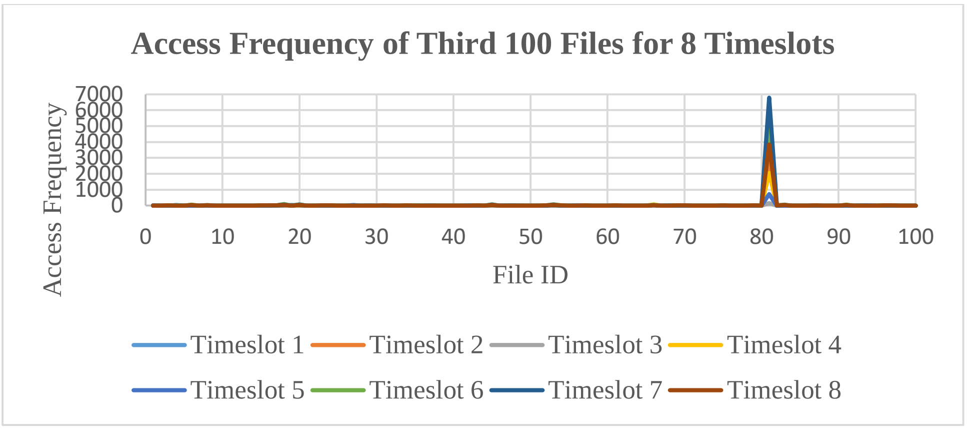

Figure 5.3 Access Frequency of Third 100 Files for 8 Timeslots

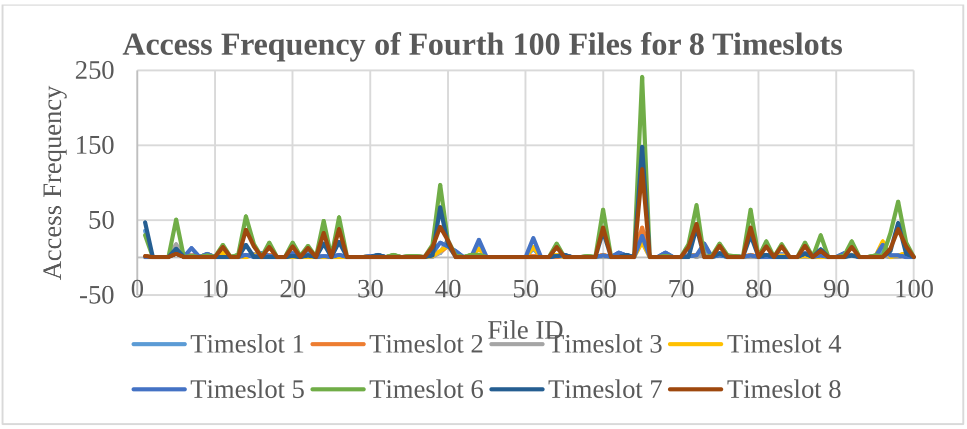

Figure 5.4 Access Frequency of Fourth 100 Files for 8 Timeslots

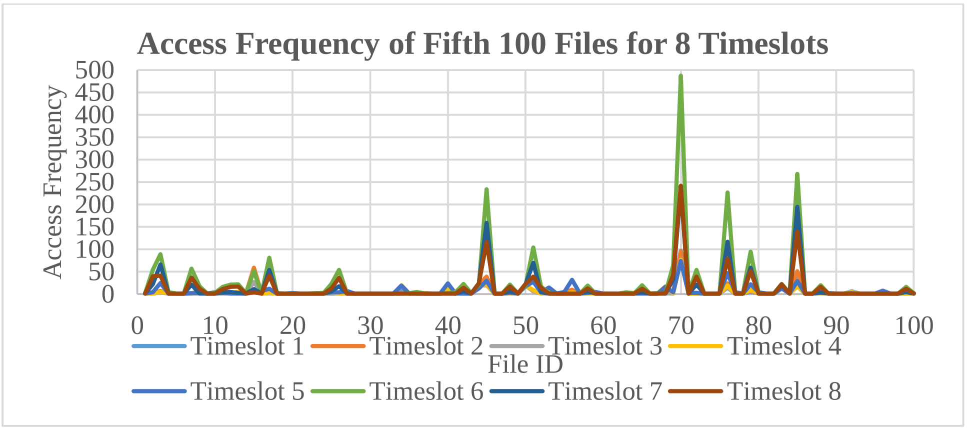

Figure 5.5 Access Frequency of Fifth 100 Files for 8 Timeslots

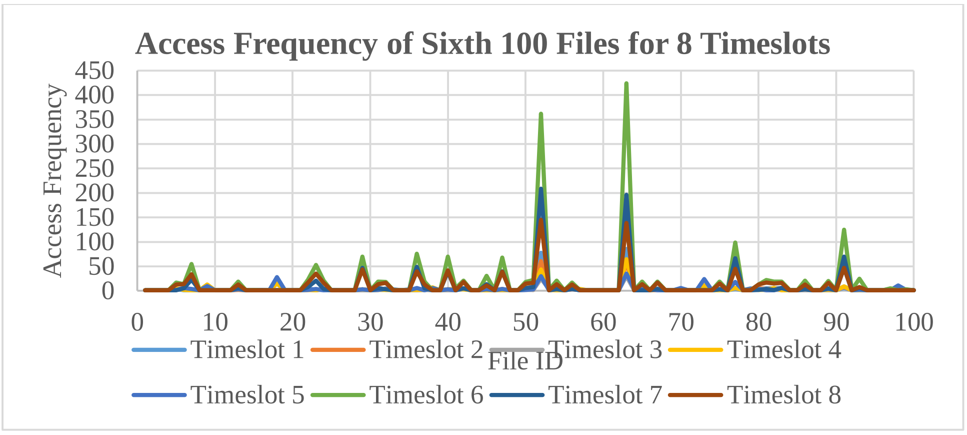

Figure 5.6 Access Frequency of Sixth 100 Files for 8 Timeslots

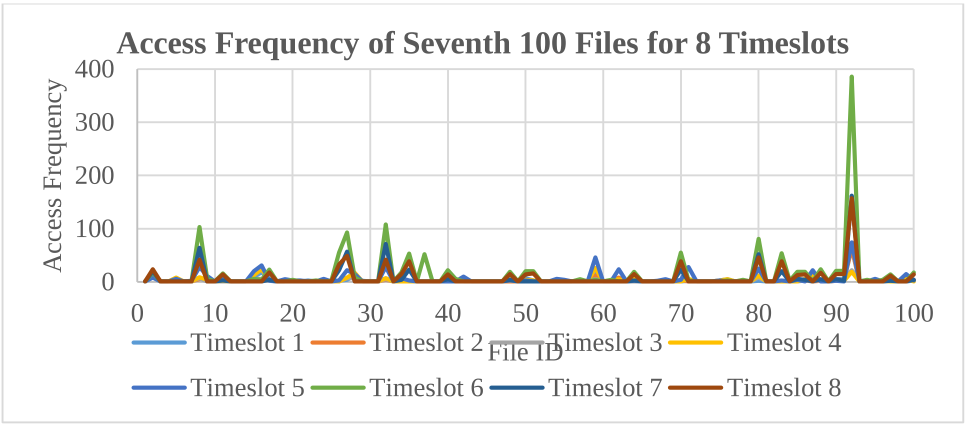

Figure 5.7 Access Frequency of Seventh 100 Files for 8 Timeslots

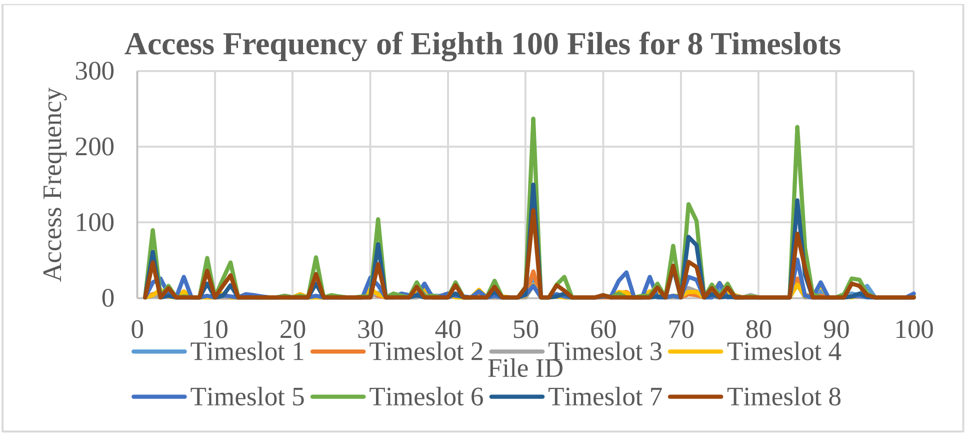

Figure 5.8 Access Frequency of Eighth 100 Files for 8 Timeslots

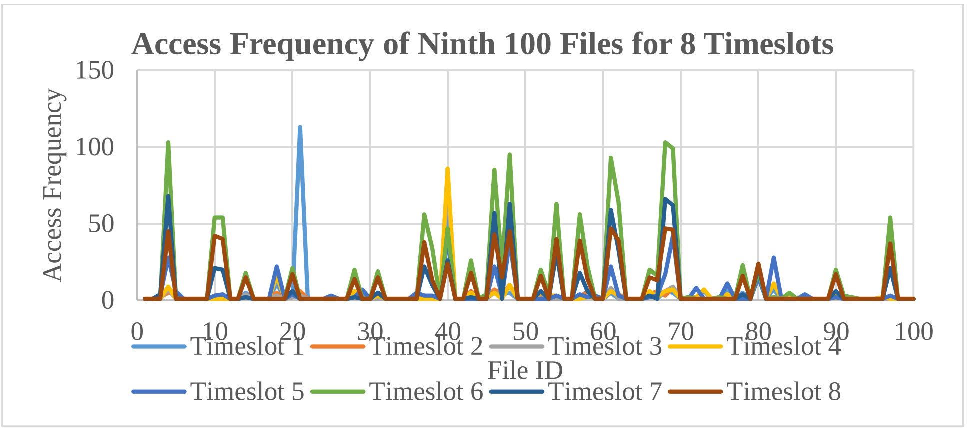

Figure 5.9 Access Frequency of Ninth 100 Files for 8 Timeslots

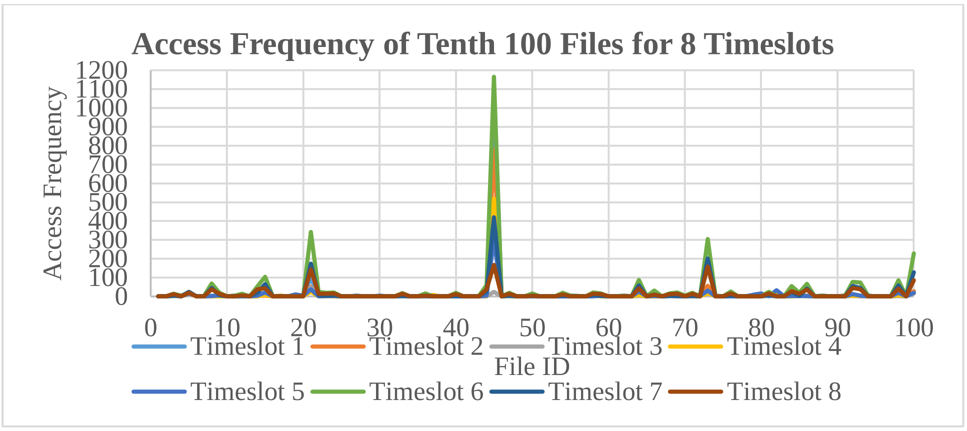

Figure 5.10 Access Frequency of Tenth 100 Files for 8 Timeslots

From Figure 5.11 to 5.20 show the popularity index of 1000 files in 8 timeslots according to their access frequencies.

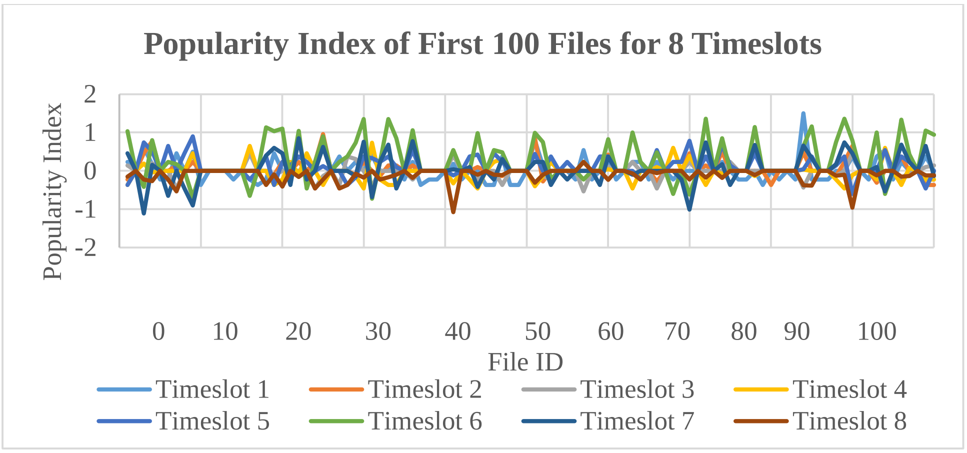

Figure 5.11 Popularity Index of First 100 Files for 8 Timeslots

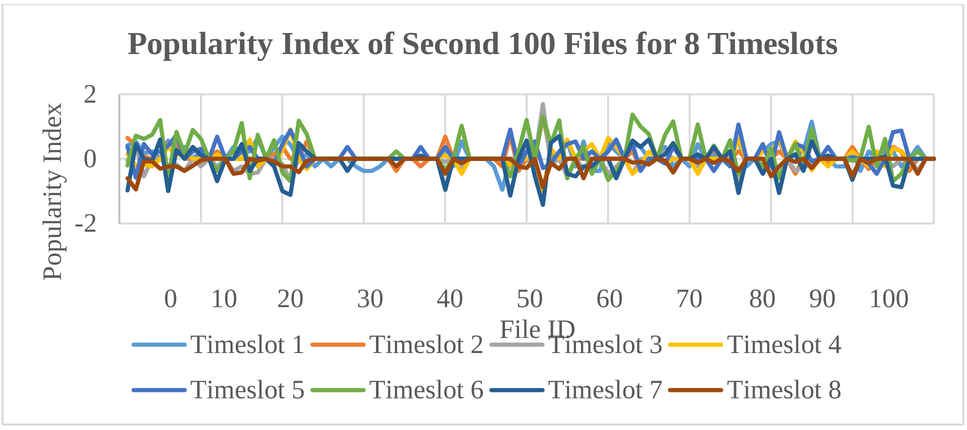

Figure 5.12 Popularity Index of Second 100 Files for 8 Timeslots

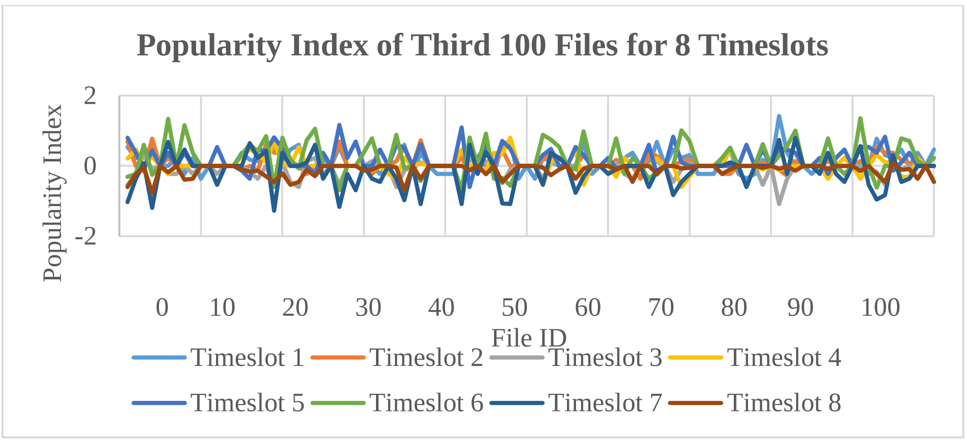

Figure 5.13 Popularity Index of Third 100 Files for 8 Timeslots

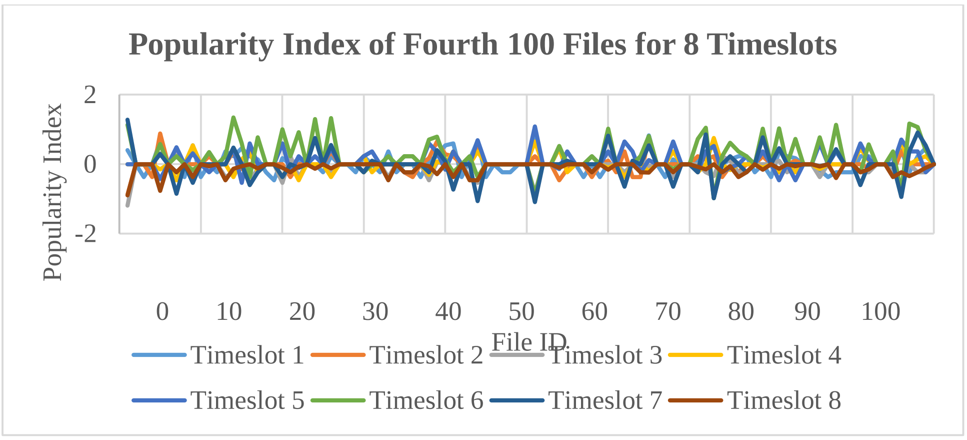

Figure 5.14 Popularity Index of Fourth 100 Files for 8 Timeslots

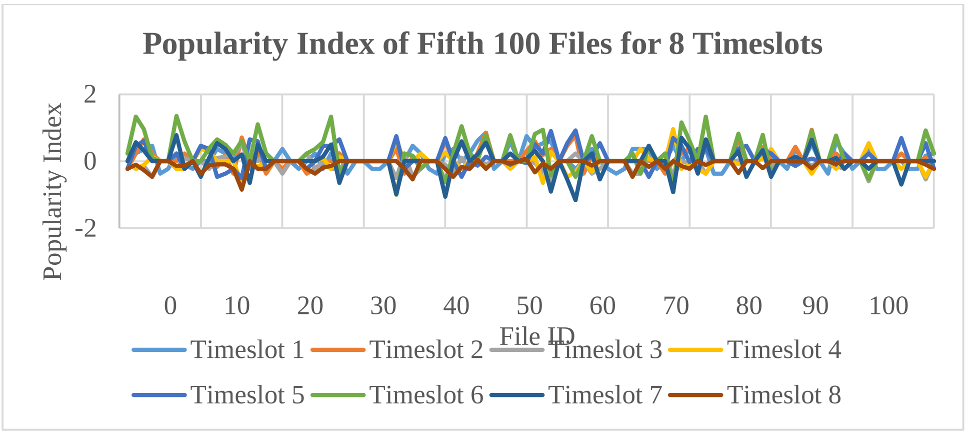

Figure 5.15 Popularity Index of Fifth 100 Files for 8 Timeslots

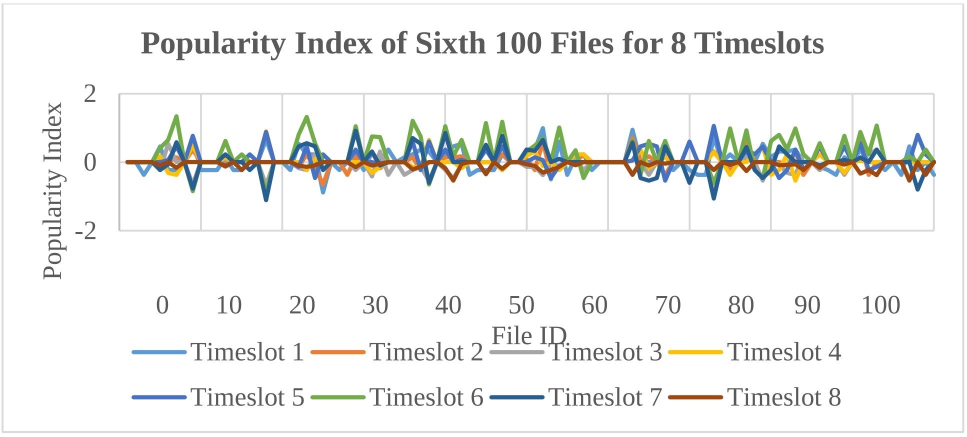

Figure 5.16 Popularity Index of Sixth 100 Files for 8 Timeslots

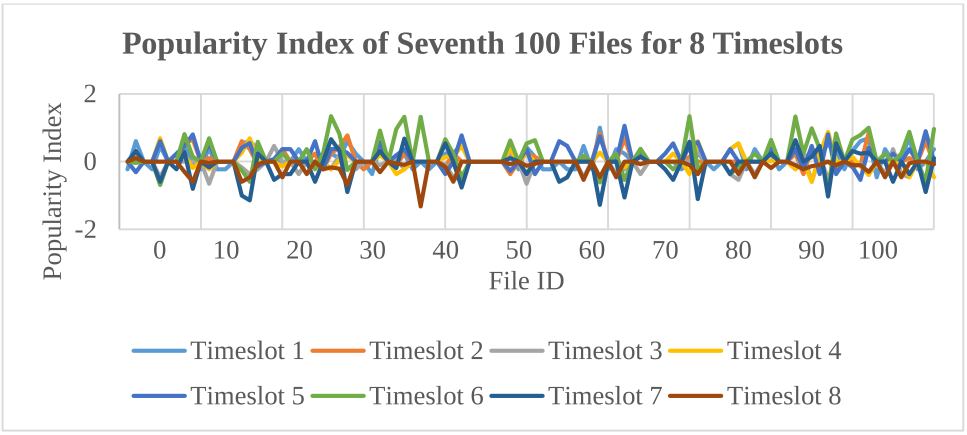

Figure 5.17 Popularity Index of Seventh 100 Files for 8 Timeslots

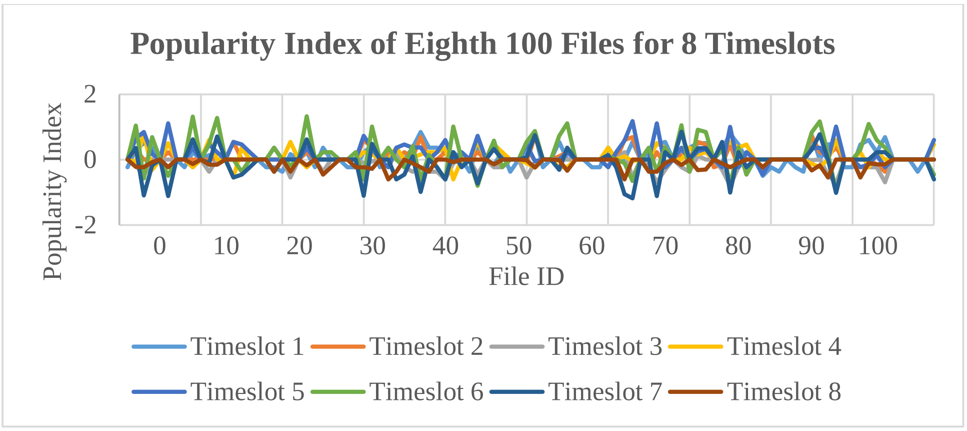

Figure 5.18 Popularity Index of Eighth 100 Files for 8 Timeslots

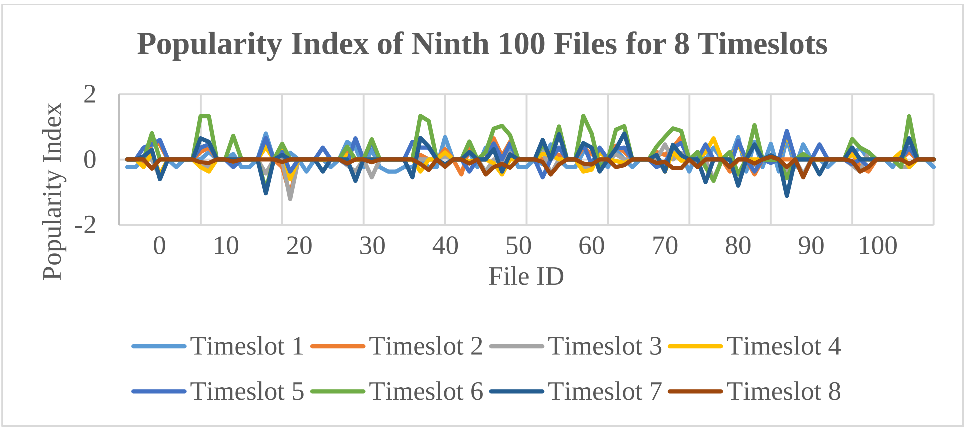

Figure 5.19 Popularity Index of Ninth 100 Files for 8 Timeslots

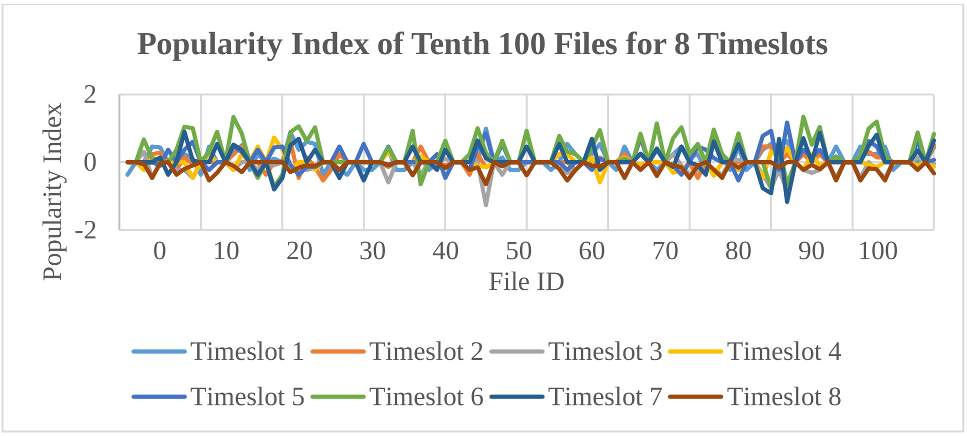

Figure 5.20 Popularity Index of Tenth 100 Files for 8 Timeslots

5.3 Evaluation Metrics

The replication algorithms are implemented and tested. The experiments are set up by using three evaluation parameters: number of replicas, storage cost, and disk utilization. Detailed explanations are presented in the following:

5.3.1 Number of Replicas

To get the effective availability level and to reduce delay time, a reasonable number of replicas of data files are needed. Instead of maintaining static replica number, numbers of replicas should be adaptable to the data popularity in every time. After the calculation of the rate of change of data popularity, the number of replicas for each file is defined using changes of data popularity, which is the outcome of the first stage. Initially, existing replicas will be assumed as 3 like the default replica of HDFS. If k is less than 0.0, then existing replicas is decreased by 1. If k is greater than 0.0, then existing replicas is increased by 1. If k is equal to 0.0, then existing replicas is unvaried. Otherwise, if it is a new file, then existing replicas is determined 3 like the default replica of HDFS. According to the evaluation results, the number of replicas is changeable with access counts changing in proposed system and LALW algorithm and however, LALW algorithm creates more replicas than the proposed system ECS.

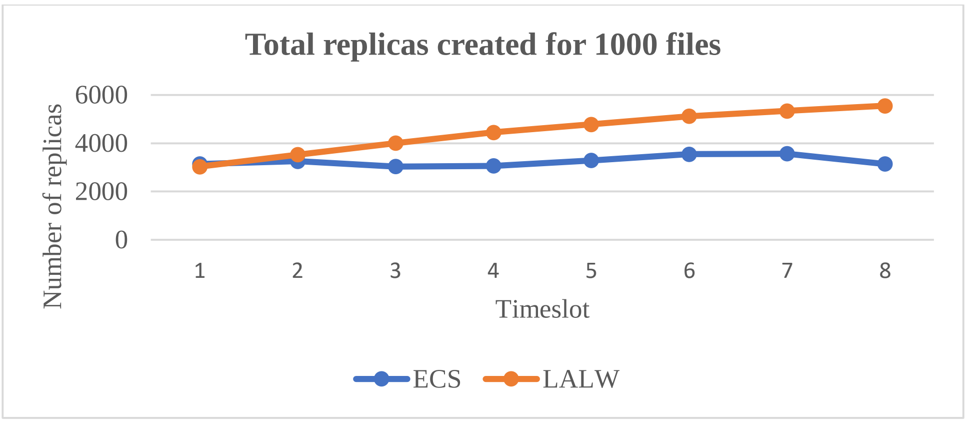

Figure 5.21 Total Number of Created Replicas for 8 Timeslots

5.3.2 Storage Cost

The performance of the proposed replication algorithm is measured in terms of storage cost. In the proposed system, a cost model is applied which takes not only physical storage cost but also maintenance cost and data access cost into account. Equation 5.1 presents the general formula of the cost model which has been used in (Kim & Fox n.d.) [44].

$$Cost_s(r) = Cost_{phy}(r) + Cost_{access}(r) + Cost_{main}(r) \qquad \text{Equation (5.1)}$$

$$Cost_{phy}(r) = r * Size_n * C_{phy} \qquad \text{Equation (5.2)}$$

$$Cost_{access}(r) = \left(\frac{Afi_n}{r}\right) * C_{access} \qquad \text{Equation (5.3)}$$

$$Cost_{main}(r) = (\lambda T)^r * C_{main} \qquad \text{Equation (5.4)}$$

In Equation 5.1 to 5.4, $Cost_s(r)$, $Cost_{phy}(r)$, $Cost_{access}(r)$ and $Cost_{main}(r)$ are functions of total storage cost, physical storage cost, data access cost and maintenance cost for replication factor r. The size of file n is defined as $Size_n$ and access frequency of file n in time interval i is Afi_n. For maintenance cost $Cost_{main}(r)$, λ and T are failure rate and transaction time. Finally, C_{phy} , C_{access} and C_{main} are constant parameters for physical cost, access cost and maintenance cost for all equations. To compare the cost of replication by using Equation 5.1, the system parameters are tuning according to Table 5.2. However, constant value C_{access} for data access overhead is set to 2 for each Data Center deployment in each comparison. The failure rate in the system is 0.001, 0.002 and 0.003. The file size in the system is 64 MB, 128 MB and 512 MB. The transaction time is 10. The constant parameter for physical cost, is 1, 2, 3, 4, and 5 and the constant parameter for maintenance cost is the thrice of physical cost so that 3, 6, 9, 12, and 15 because maintenance effort includes the correction effort, evolution effort and management effort [88].

Table 5.2 Parameters for Storage Cost

C_{phy}	C_{main}	λ	T	$Size_n$
1	3	0.001	10	64
2	6	0.001	10	64
3	9	0.001	10	64
4	12	0.001	10	64

5	15	0.001	10	64
1	3	0.002	10	128
2	6	0.002	10	128
3	9	0.002	10	128
4	12	0.002	10	128
5	15	0.002	10	128
1	3	0.003	10	512
2	6	0.003	10	512
3	9	0.003	10	512
4	12	0.003	10	512
5	15	0.003	10	512

Figure 5.22 to 5.26 shows the comparison of storage cost ECS and LALW for 8 timeslots with various physical cost and maintenance cost when failure rate in the system is 0.001. From the evaluation results, ECS costs few than LALW at timeslot 1, however, it does not cost more than LALW at other 7 timeslots because ECS considers the number of replicas for both popular and unpopular data.

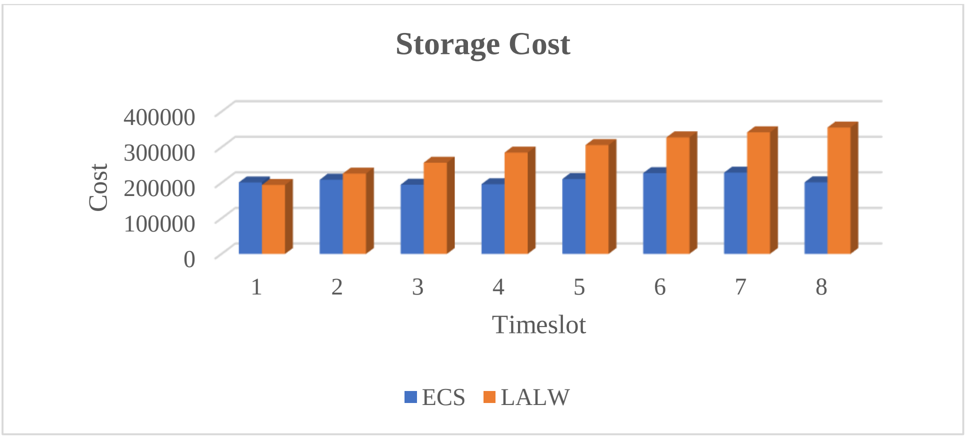

Figure 5.22 Storage Cost of ECS and LALW for 8 Timeslots

$(\boldsymbol{C_{phy} = 1, C_{main} = 3, \lambda = 0.001, T = 10, Size_n = 64})$

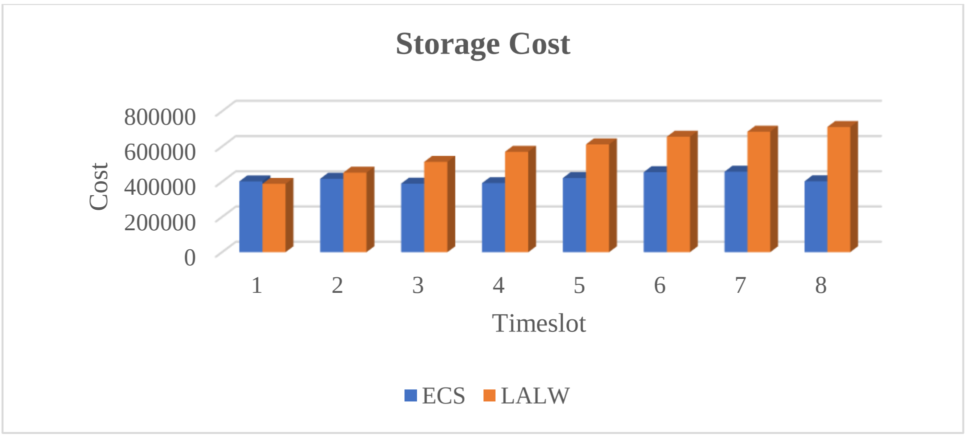

Figure 5.23 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 2, C_{main} = 6, \lambda = 0.001, T = 10, Size_n = 64)$$

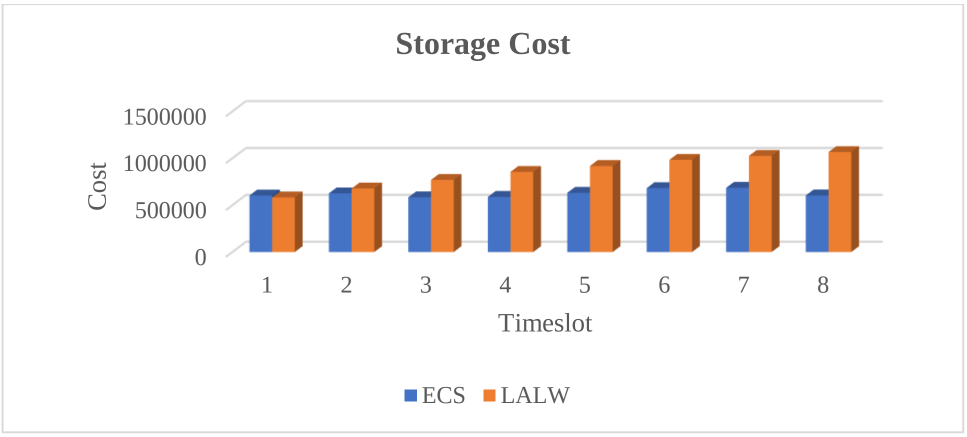

Figure 5.24 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 3, C_{main} = 9, \lambda = 0.001, T = 10, Size_n = 64)$$

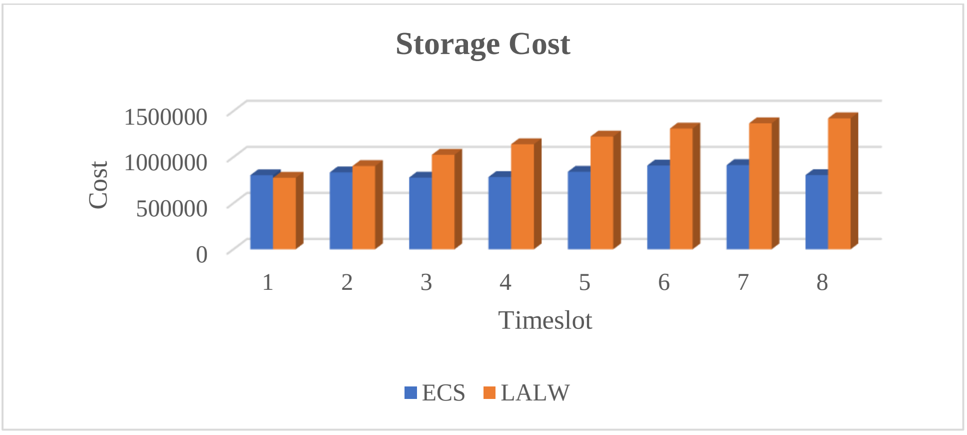

Figure 5.25 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 4, C_{main} = 12, \lambda = 0.001, T = 10, Size_n = 64)$$

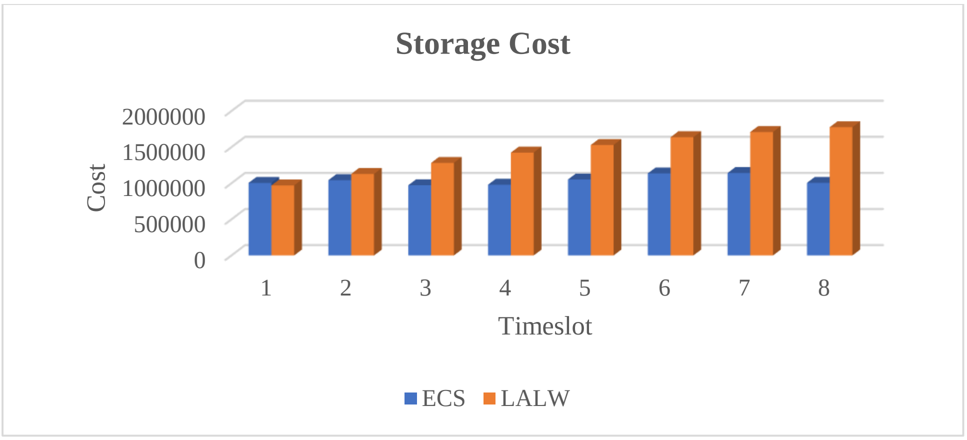

Figure 5.26 Storage Cost of ECS and LALW for 8 Timeslots
$$(C_{phy} = 5, C_{main} = 15, \lambda = 0.001, T = 10, Size_n = 64)$$

Figure 5.27 to 5.31 shows the comparison of storage cost of ECS and LALW for 8 timeslots with various physical cost and maintenance cost when failure rate in the system is 0.002. From the evaluation results, ECS costs few than LALW at timeslot 1, however, it does not cost more than LALW at other 7 timeslots.

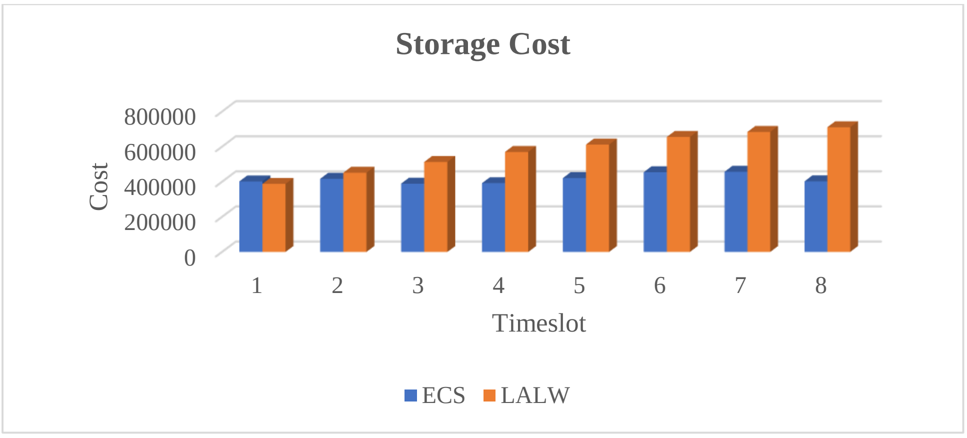

Figure 5.27 Storage Cost of ECS and LALW for 8 Timeslots
$$(C_{phy} = 1, C_{main} = 3, \lambda = 0.002, T = 10, Size_n = 128)$$

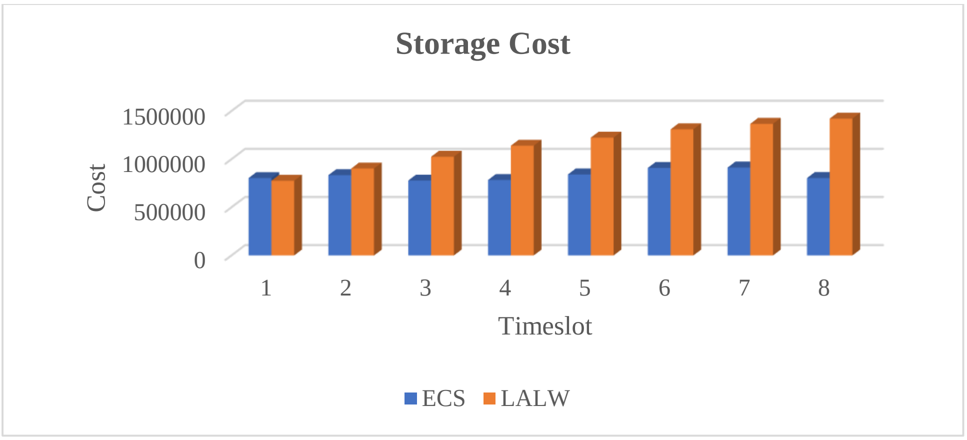

Figure 5.28 Storage Cost of ECS and LALW for 8 Timeslots

$(\boldsymbol{C_{phy}} = \mathbf{2}, \boldsymbol{C_{main}} = \mathbf{6}, \boldsymbol{\lambda} = \mathbf{0.002}, \boldsymbol{T} = \mathbf{10}, \boldsymbol{Size_n} = \mathbf{128})$

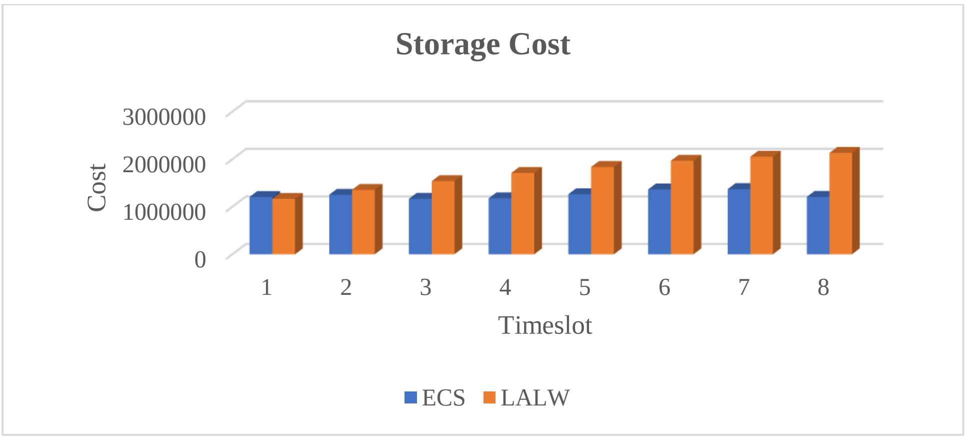

Figure 5.29 Storage Cost of ECS and LALW for 8 Timeslots

$(\boldsymbol{C_{phy}} = \mathbf{3}, \boldsymbol{C_{main}} = \mathbf{9}, \boldsymbol{\lambda} = \mathbf{0.002}, \boldsymbol{T} = \mathbf{10}, \boldsymbol{Size_n} = \mathbf{128})$

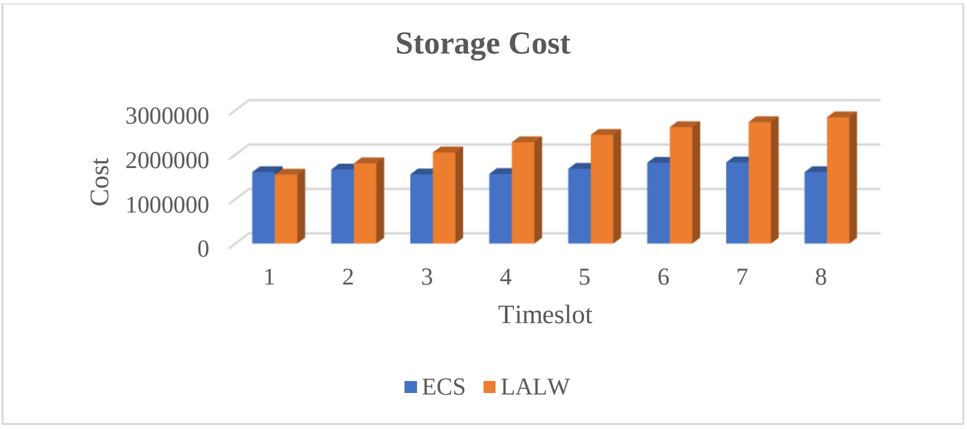

Figure 5.30 Storage Cost of ECS and LALW for 8 Timeslots

$(\boldsymbol{C_{phy}} = \mathbf{4}, \boldsymbol{C_{main}} = \mathbf{12}, \boldsymbol{\lambda} = \mathbf{0.002}, \boldsymbol{T} = \mathbf{10}, \boldsymbol{Size_n} = \mathbf{128})$

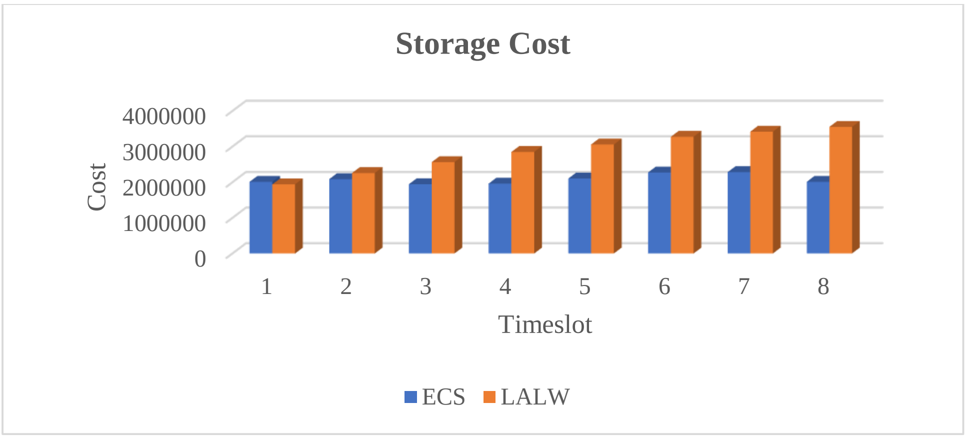

Figure 5.31 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 5, C_{main} = 15, \lambda = 0.002, T = 10, Size_n = 128)$$

Figure 5.32 to 5.36 shows the comparison of storage cost of ECS and LALW for 8 timeslots with various physical cost and maintenance cost when failure rate in the system is 0.003. From the evaluation results, ECS costs few than LALW at timeslot 1, however, it does not cost more than LALW at other 7 timeslots.

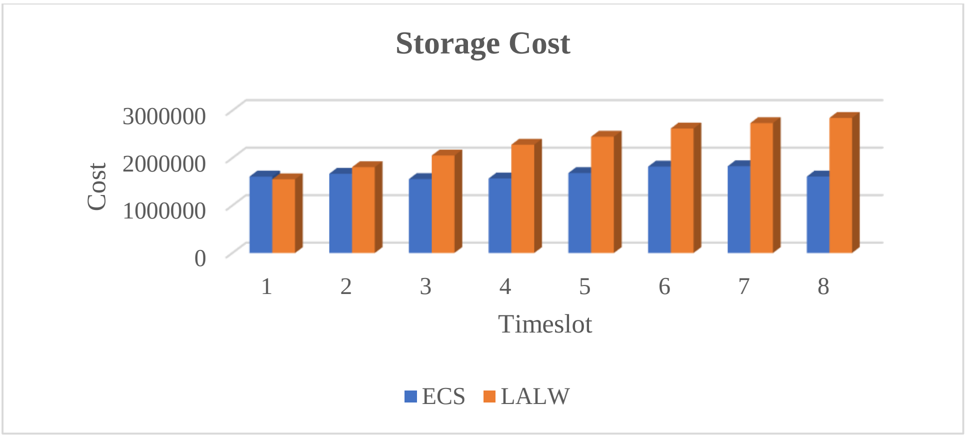

Figure 5.32 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 1, C_{main} = 3, \lambda = 0.003, T = 10, Size_n = 512)$$

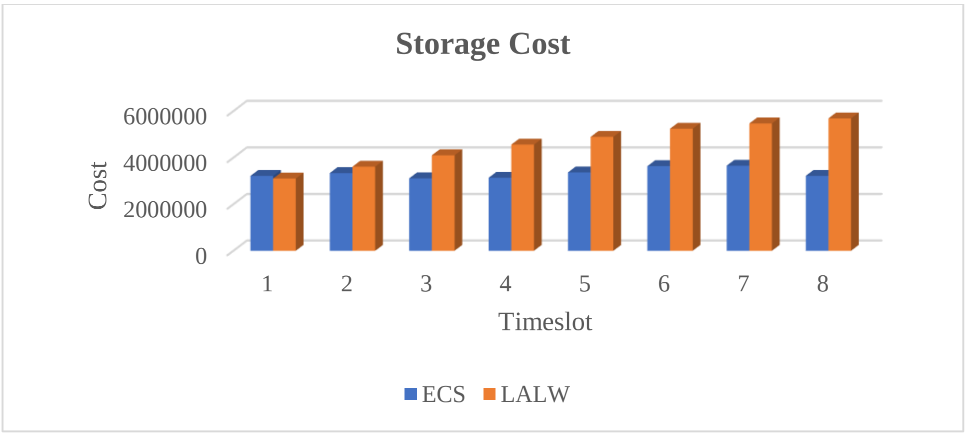

Figure 5.33 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 2, C_{main} = 6, \lambda = 0.003, T = 10, Size_n = 512)$$

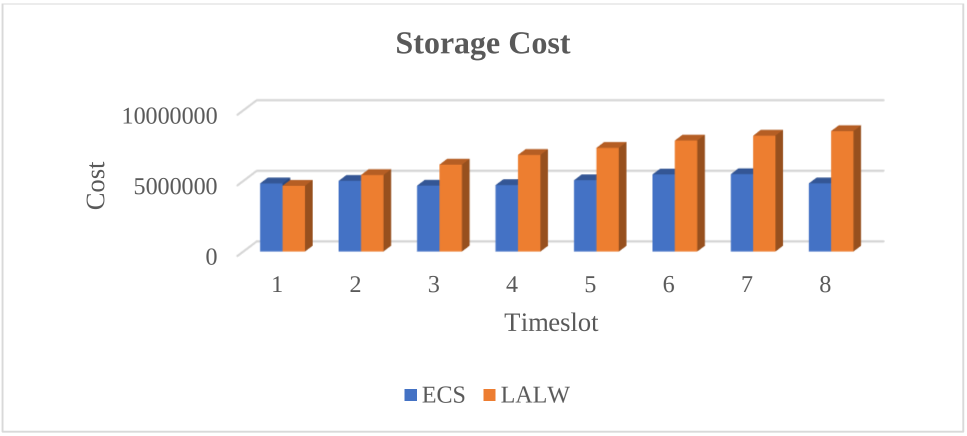

Figure 5.34 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 3, C_{main} = 9, \lambda = 0.003, T = 10, Size_n = 512)$$

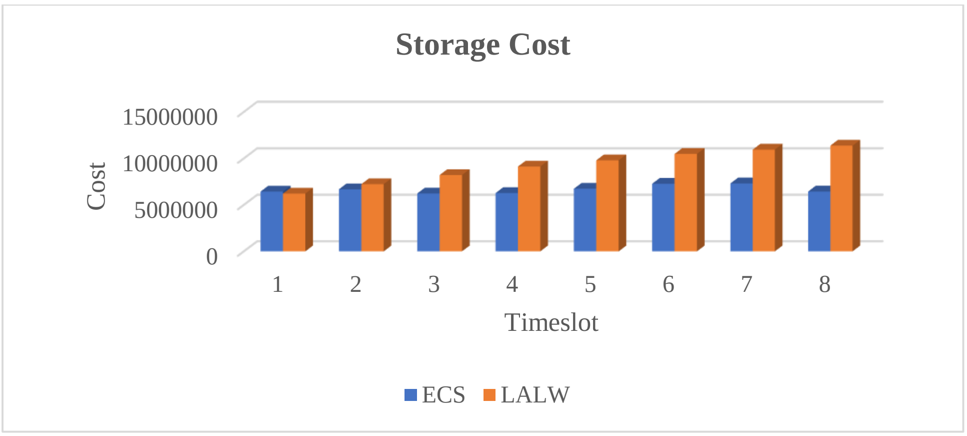

Figure 5.35 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 4, C_{main} = 12, \lambda = 0.003, T = 10, Size_n = 512)$$

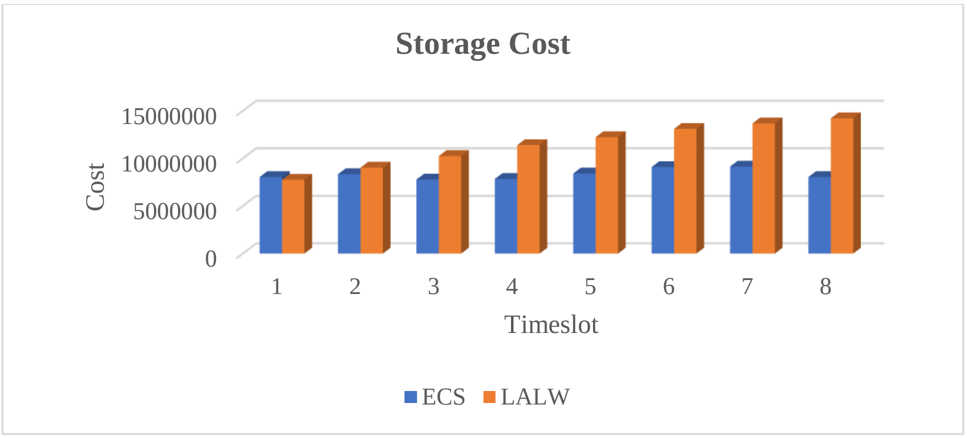

Figure 5.36 Storage Cost of ECS and LALW for 8 Timeslots

$$(\boldsymbol{C_{phy} = 5, C_{main} = 15, \lambda = 0.003, T = 10, Size_n = 512})$$

5.3.3 Disk Utilization

In this proposed system, the replicas are almost uniformly distributed for achieving the load balancing in nodes in the cluster. Disk utilization of the proposed system are compared with LALW algorithm in order to avoid overload condition. LALW does not obey the placement policy of hadoop because it places the same data replicas at one host. Therefore, LALW does not achieve the load balancing like the proposed system. Figure 5.37 shows the disk utilization comparison of the proposed system and LALW at timeslot 1. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 1 because ECS considers the overload condition of the nodes in the cluster.

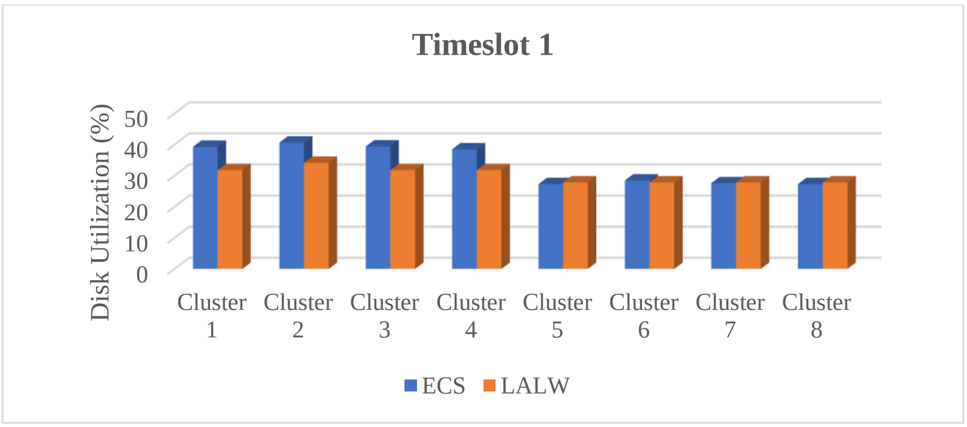

Figure 5.37 Disk Utilization of ECS and LALW at Timeslot 1

Figure 5.38 shows the disk utilization comparison of ECS and LALW at timeslot 2. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 2.

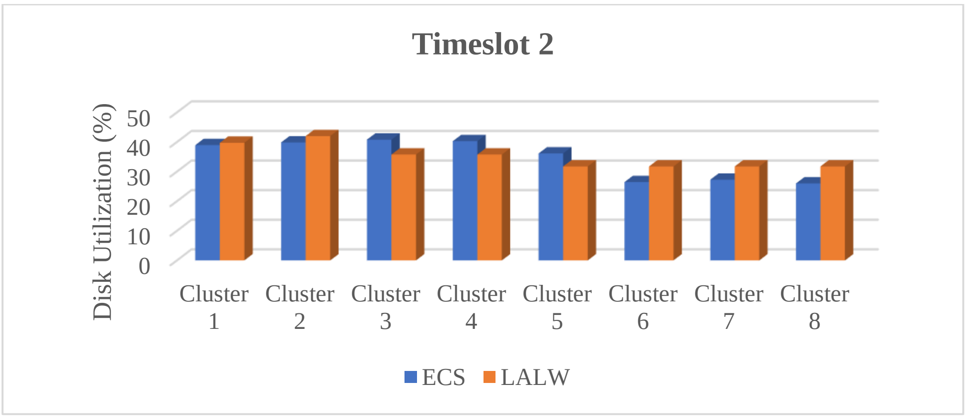

Figure 5.38 Disk Utilization of ECS and LALW at Timeslot 2

Figure 5.39 shows the disk utilization comparison of ECS and LALW at timeslot 3. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 3.

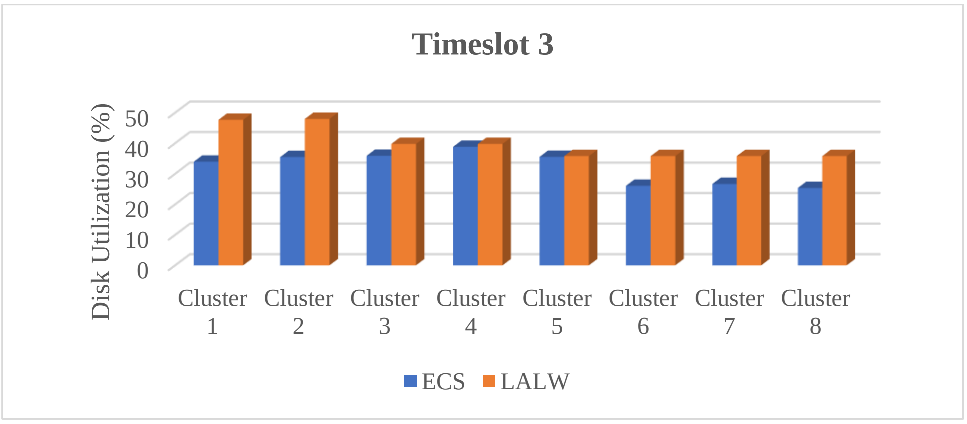

Figure 5.39 Disk Utilization of ECS and LALW at Timeslot 3

Figure 5.40 shows the disk utilization comparison of ECS and LALW at timeslot 4. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 4.

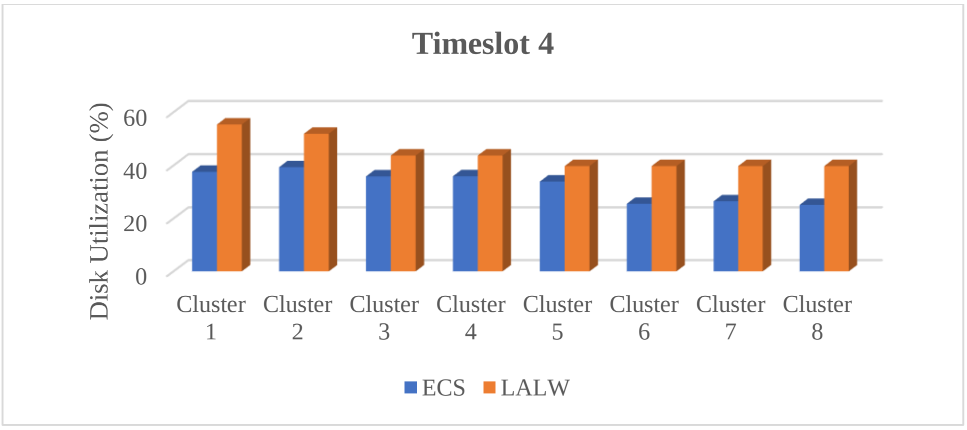

Figure 5.40 Disk Utilization of ECS and LALW at Timeslot 4

Figure 5.41 shows the disk utilization comparison of ECS and LALW at timeslot 5. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 5.

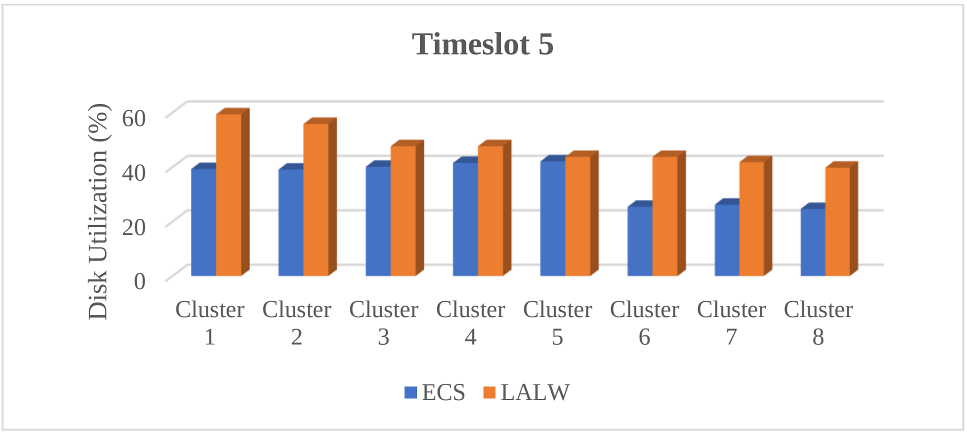

Figure 5.41 Disk Utilization of ECS and LALW at Timeslot 5

Figure 5.42 shows the disk utilization comparison of ECS and LALW at timeslot 6. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 6.

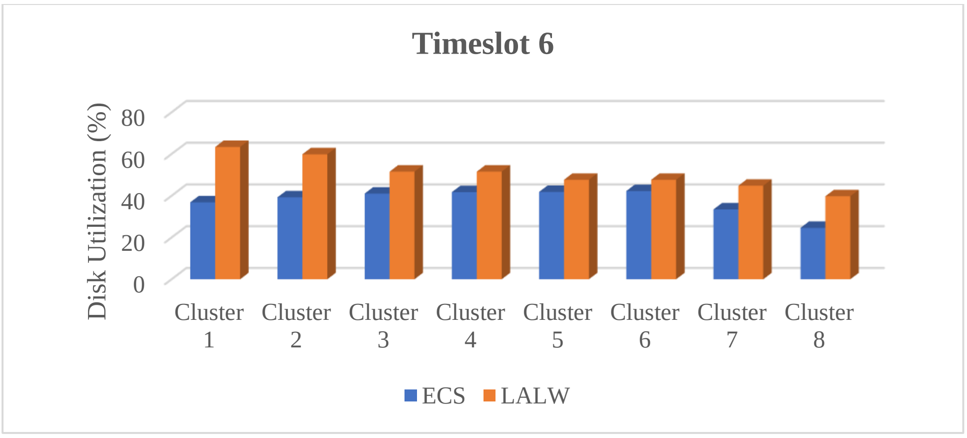

Figure 5.42 Disk Utilization of ECS and LALW at Timeslot 6

Figure 5.43 shows the disk utilization comparison of ECS and LALW at timeslot 7. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 7.

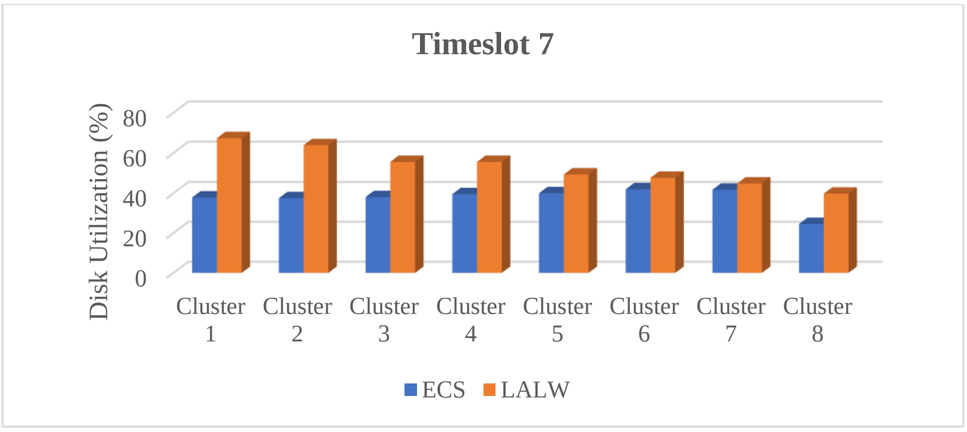

Figure 5.43 Disk Utilization of ECS and LALW at Timeslot 7

Figure 5.44 shows the disk utilization comparison of ECS and LALW at timeslot 8. From the evaluation results, ECS achieves more load balancing than LALW at this timeslot 8.

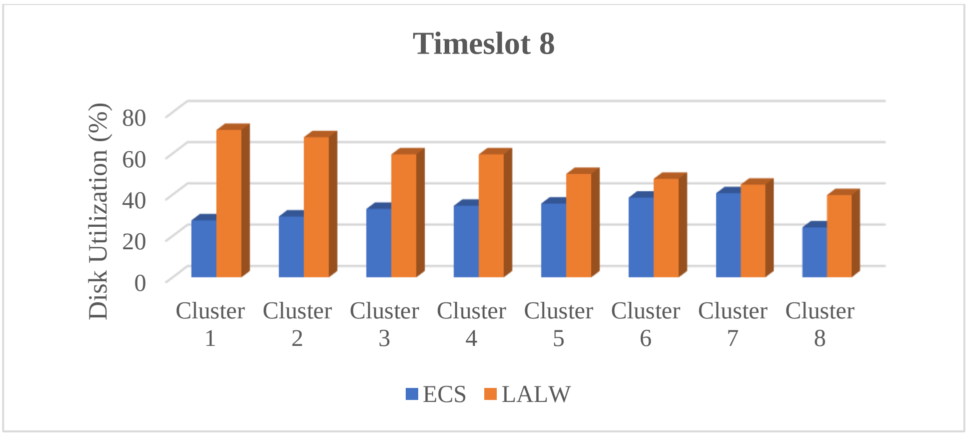

Figure 5.44 Disk Utilization of ECS and LALW at Timeslot 8

Figure 5.45 shows the average disk utilization comparison of ECS and LALW for 8 timeslots. From the evaluation results, ECS achieves more load balancing than LALW for 8 timeslots.

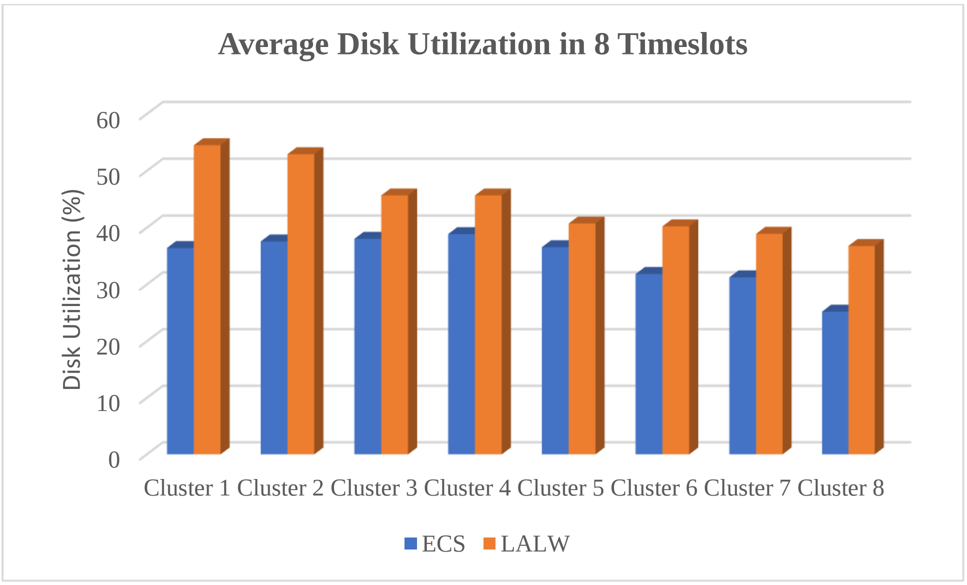

Figure 5.45 Average Disk Utilization of ECS and LALW for 8 Timeslots

5.4 Analysis of Load Factor in Data Placement

In this data placement, load factor of host or node is considered. The experiment is tested with varying the coefficient values of disk utilization $\boldsymbol{\alpha}$, disk bandwidth $\boldsymbol{\beta}$ and CPU utilization $\boldsymbol{\gamma}$. If the value of load factor at host is less than the predefined value of cluster, data is placed into node and if not, it is performed by replacing the replica having minimum access frequency with the new replica. To achieve the optimal coefficient values of disk utilization $\boldsymbol{\alpha}$, disk bandwidth $\boldsymbol{\beta}$ and CPU utilization $\boldsymbol{\gamma}$, the system parameters are varied according to Table 5.3.

Table 5.3 Parameters for Load Factor

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$
0.35	0.33	0.32
0.35	0.34	0.31
0.35	0.35	0.3
0.4	0.35	0.25
0.4	0.4	0.2
0.45	0.35	0.2
0.45	0.3	0.25
0.45	0.4	0.15
0.45	0.45	0.1
0.5	0.45	0.05
0.5	0.3	0.2
0.5	0.35	0.15
0.5	0.4	0.1
0.5	0.25	0.25
0.6	0.3	0.1
0.6	0.25	0.15
0.6	0.35	0.05
0.6	0.2	0.2
0.7	0.2	0.1
0.7	0.25	0.05
0.7	0.15	0.15
0.8	0.1	0.1
0.8	0.15	0.05
0.85	0.1	0.05

In the analysis of load factor, I varied the coefficient values of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ in order to get the optimal parameter of load factor. Firstly, I varied the coefficient value of $\boldsymbol{\alpha}$ while the coefficient value of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are set with fixed value. In that condition, I found that 0.35 is the best optimal value for disk utilization $\boldsymbol{\alpha}$. And then, I varied the coefficient value of $\boldsymbol{\beta}$ while the coefficient value of $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are set with fixed value in second evaluation. In that condition, I found that 0.33 is the best optimal value for disk bandwidth $\boldsymbol{\beta}$. And then, I varied the coefficient value of $\boldsymbol{\gamma}$ while the coefficient value of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are set with fixed value in third evaluation. In that case, I found that 0.32 is the best optimal value for CPU utilization $\boldsymbol{\gamma}$.

From the evaluation results, disk utilization $\boldsymbol{\alpha} = 0.35$, disk bandwidth $\boldsymbol{\beta} = 0.33$ and CPU utilization $\boldsymbol{\gamma} = 0.32$ is the optimum parameter for the calculation of load factor in data placement. From Figure 5.46 to 5.53 shows the comparison of the optimal parameter ($\boldsymbol{\alpha} = 0.35$, $\boldsymbol{\beta} = 0.33$, $\boldsymbol{\gamma} = 0.32$) and the worst non-optimal parameter ($\boldsymbol{\alpha} = 0.85$, $\boldsymbol{\beta} = 0.1$, $\boldsymbol{\gamma} = 0.05$) for 8 clusters in the system. Figure 5.46 shows the load factor condition of fifty nodes in the cluster 1. According to first parameter, there is only one overload condition in this cluster, however, there is four overload condition in this cluster according to second parameter. Actually, there are enough storages for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

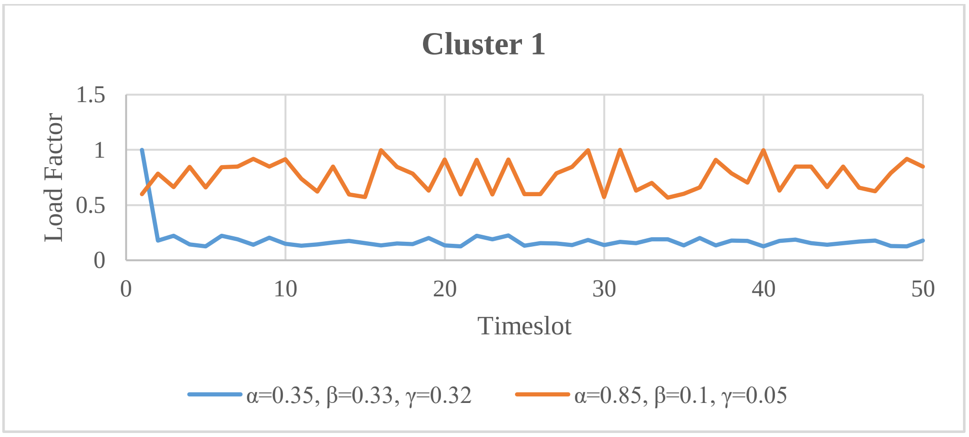

Figure 5.46 Load Factor of Cluster 1

Figure 5.47 shows the load factor condition of fifty nodes in the cluster 2. According to first parameter, there is four overload condition in this cluster, however, there is almost seven overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

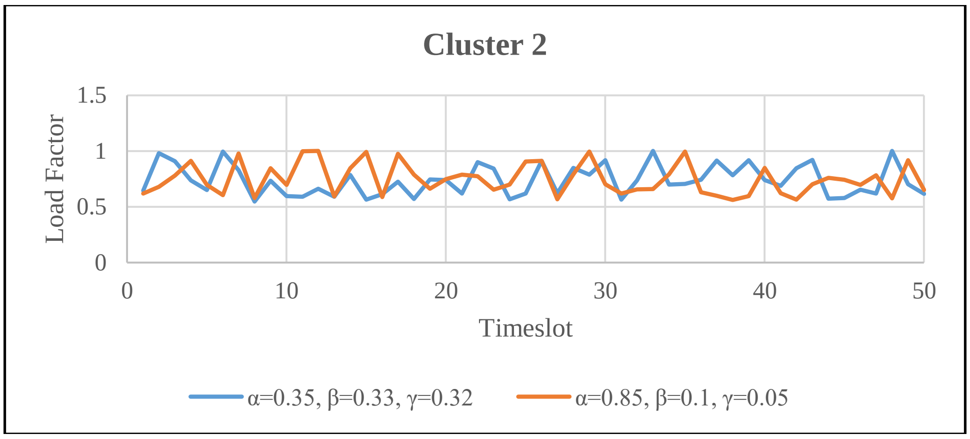

Figure 5.47 Load Factor of Cluster 2

Figure 5.48 shows the load factor condition of fifty nodes in the cluster 3. According to first parameter, there is almost five overload condition in this cluster, however, there is almost six overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

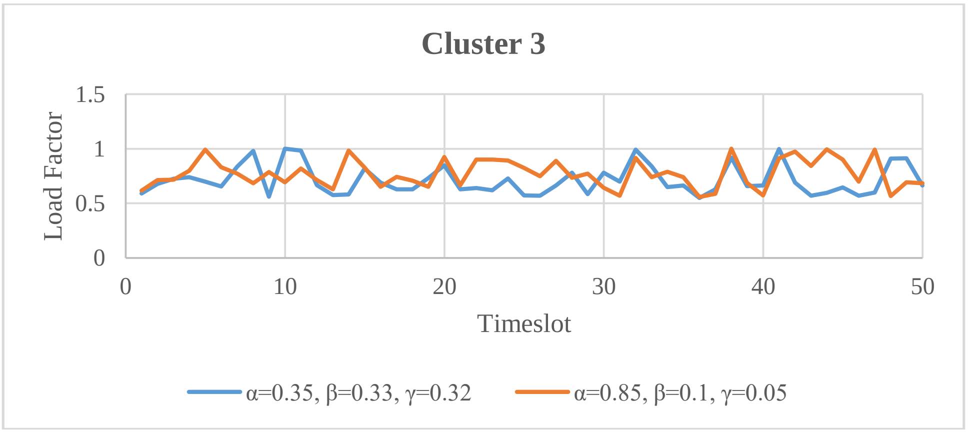

Figure 5.48 Load Factor of Cluster 3

Figure 5.49 shows the load factor condition of fifty nodes in the cluster 4. According to first parameter, there is only two overload condition in this cluster, however, there is almost five overload condition in this cluster according to second parameter. Actually, there are many enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

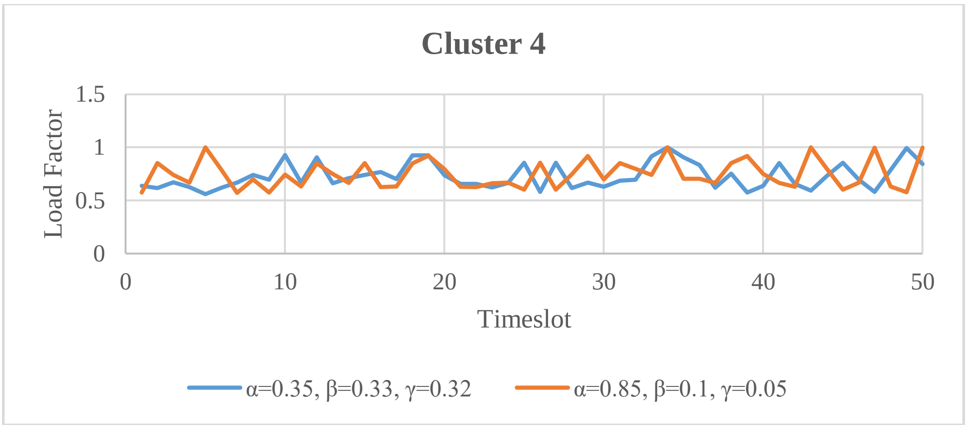

Figure 5.49 Load Factor of Cluster 4

Figure 5.50 shows the load factor condition of fifty nodes in the cluster 5. According to first parameter, there is only two overload condition in this cluster, and there is only three overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not much different as the first parameter.

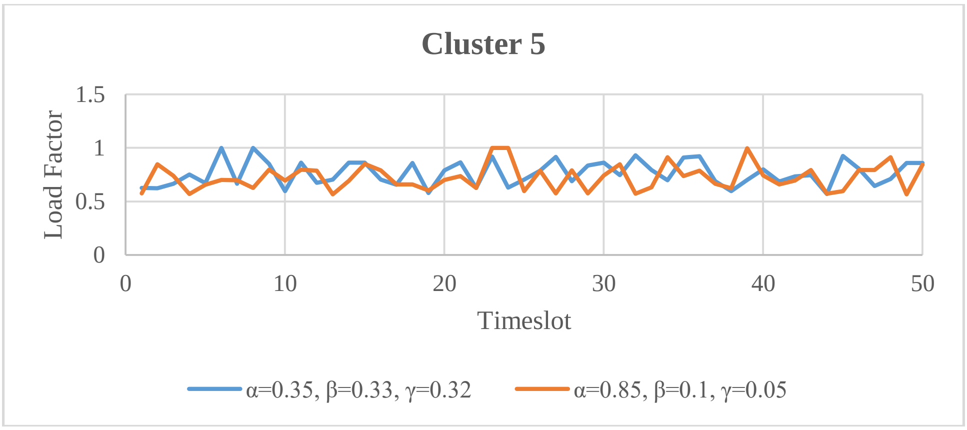

Figure 5.50 Load Factor of Cluster 5

Figure 5.51 shows the load factor condition of fifty nodes in the cluster 6. According to first parameter, there is only one overload condition in this cluster, however, there is seven overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

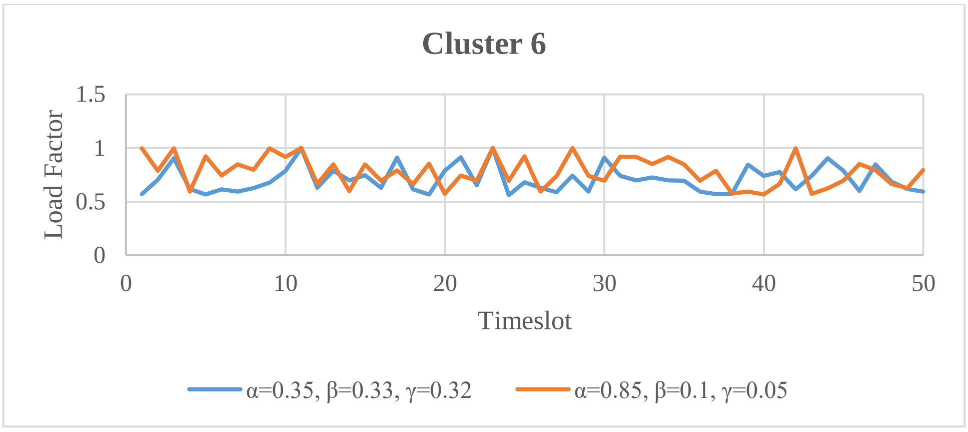

Figure 5.51 Load Factor of Cluster 6

Figure 5.52 shows the load factor condition of fifty nodes in the cluster 7. According to first parameter, there is three overload condition in this cluster, and there is almost five overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not much different as the first parameter.

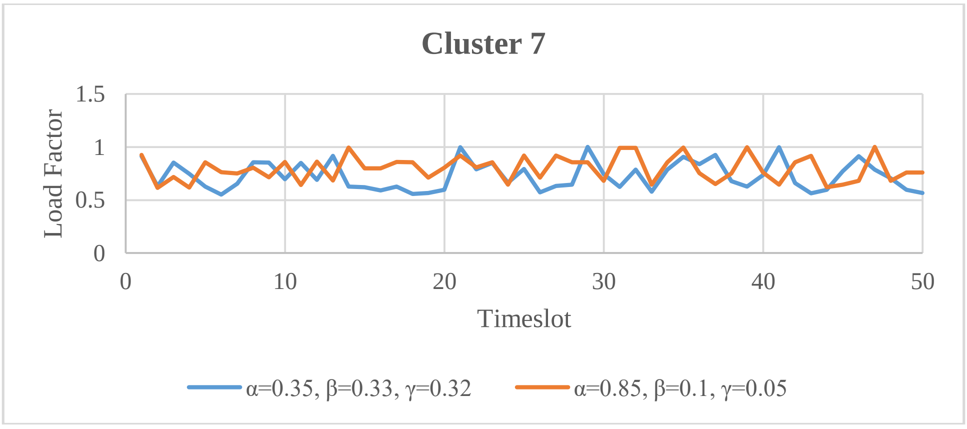

Figure 5.52 Load Factor of Cluster 7

Figure 5.53 shows the load factor condition of fifty nodes in the cluster 8. According to first parameter, there is four overload condition in this cluster, however, there is almost nine overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

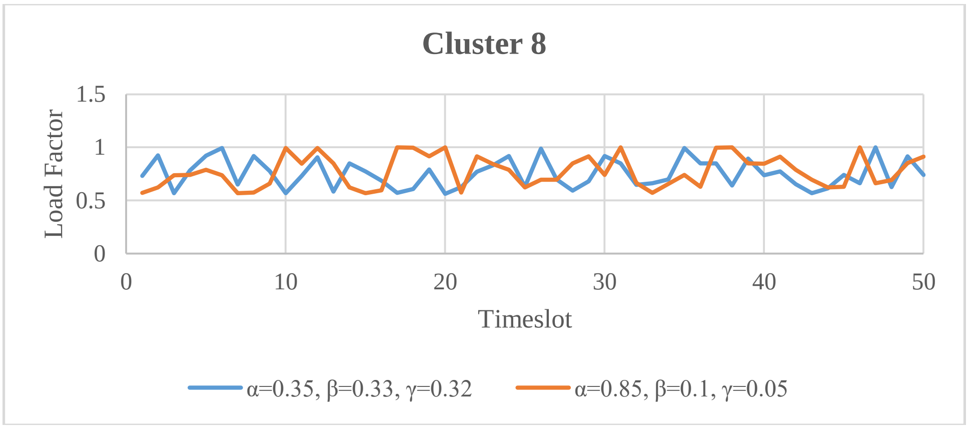

Figure 5.53 Load Factor of Cluster 8

From the evaluation results, the less the coefficient values of disk utilization, disk bandwidth and CPU utilization, the more getting the optimal parameter for load factor in data placement. The most optimum parameter can give the accurate detection of the overload condition in the cluster.

To perform the next evaluation of the proposed replication algorithms, different timeslots of Yahoo Audit log dataset [48] is used and the description of dataset is mentioned in Table 5.4. In order to count data access frequency in each timeslot, the dataset is divided depending on date and time. To simplify the analysis, each time slot is defined as 3 minutes period.

Table 5.4 Description of Tested Dataset

Test data description		Yahoo Webscope user audit logs (2010-01-12 00:30:00 to 2010-01-12 00:59:59)
Number of timeslots tested in the algorithm		10
Each Timeslot duration		3 minutes
Total records	**Timeslot 1**	97,834
	Timeslot 2	96,735
	Timeslot 3	74,733
	Timeslot 4	100,393
	Timeslot 5	87,786

	Timeslot 6	78,902
	Timeslot 7	87,880
	Timeslot 8	91,403
	Timeslot 9	52,452
	Timeslot 10	42,534

From these 10 timeslots, timeslot 1, 2 and 3 are set as timeslot 1, timeslot 2, 3 and 4 are set as timeslot 2, and timeslot 3, 4 and 5 are set as timeslot 3 and so on for the computation of the rate of change of data popularity using differential equation. From this dataset, 1000 files are extracted to perform the evaluation of proposed system. At the beginning of simulation, the number of existing replicas is set as 3 for all 1000 files.

5.5 Evaluation Metrics

The replication algorithms are implemented and tested. The experiments are set up by using three evaluation parameters: number of replicas, storage cost, and disk utilization. Detailed explanations are presented in the following:

5.5.1 Number of Replicas

To get the effective availability level and to reduce delay time, a reasonable number of replicas of data files are needed. Instead of maintaining static replica number, numbers of replicas should be adaptable to the data popularity in every time. After the calculation of the rate of change of data popularity, the number of replicas for each file is defined using changes of data popularity, which is the outcome of the first stage. According to the evaluation results, the number of replicas is changeable with access counts changing in ECS and LALW algorithm and however, LALW algorithm creates more replicas than ECS.

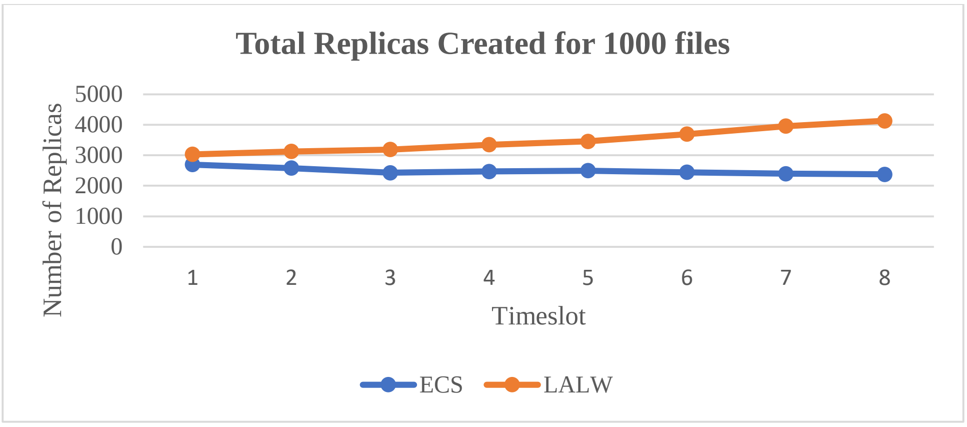

Figure 5.54 Total Number of Created Replicas for 8 Timeslots

5.5.2 Storage Cost

The performance of the proposed replication algorithm is measured in terms of storage cost. In the proposed system, a cost model is applied which takes not only physical storage cost but also maintenance cost and data access cost into account. To compare the cost of replication by using Equation 5.1, the system parameters are varied according to Table 5.4. However, constant value C_{access} for data access overhead is set to 2 for each Data Center deployment in each comparison. The failure rate in the system is 0.001, and 0.002. The file size in the system is 64 MB, and 128 MB. The transaction time is 10. The constant parameter for physical cost, is 1, 2, 3, 4, and 5 and the constant parameter for maintenance cost is the thrice of physical cost so that 3, 6, 9, 12, and 15 because maintenance effort includes the correction effort, evolution effort and management effort.

Table 5.5 Parameters for Storage Cost

C_{phy}	C_{main}	λ	T	$Size_n$
1	3	0.001	10	64
2	6	0.001	10	64
3	9	0.001	10	64
4	12	0.001	10	64
5	15	0.001	10	64
1	3	0.002	10	128

2	6	0.002	10	128
3	9	0.002	10	128
4	12	0.002	10	128
5	15	0.002	10	128

Figure 5.55 to 5.59 shows the comparison of storage cost of ECS and LALW for 8 timeslots with various physical cost and maintenance cost when failure rate in the system is 0.001. From the evaluation results, ECS does not cost more than LALW at 8 timeslots.

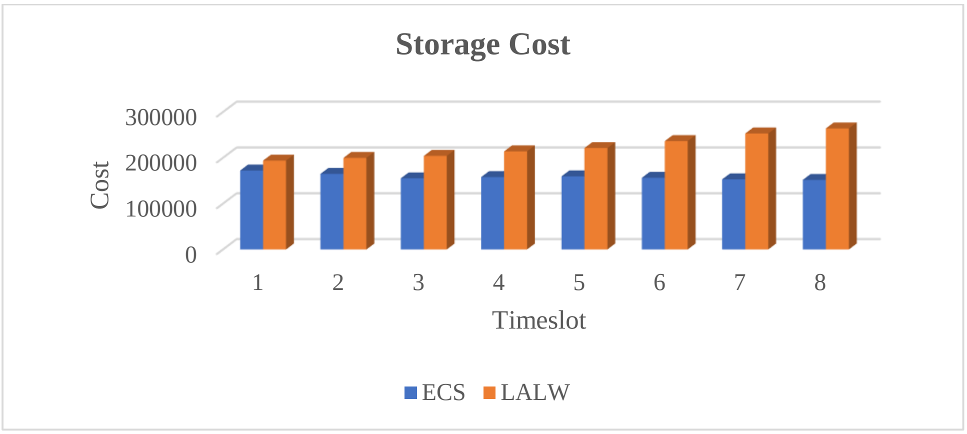

Figure 5.55 Storage Cost of ECS and LALW for 8 Timeslots
$$\left(C_{phy} = 1, C_{main} = 3, \lambda = 0.001, T = 10, Size_n = 64\right)$$

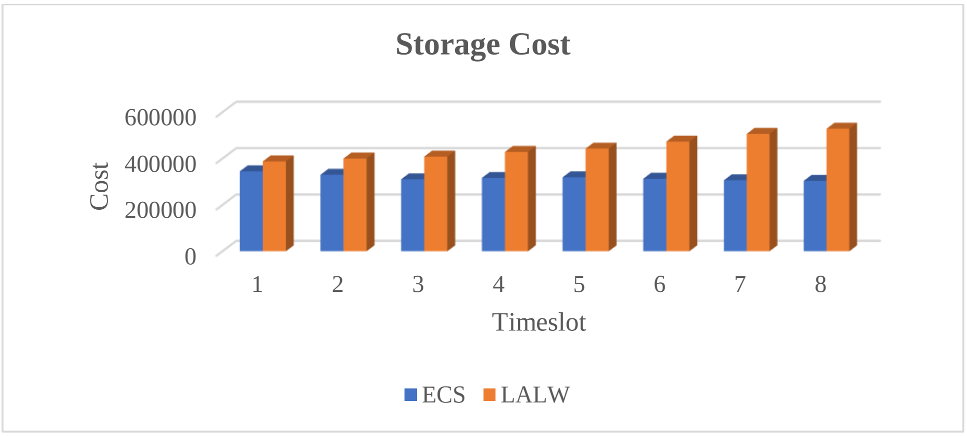

Figure 5.56 Storage Cost of ECS and LALW for 8 Timeslots
$$\left(C_{phy} = 2, C_{main} = 6, \lambda = 0.001, T = 10, Size_n = 64\right)$$

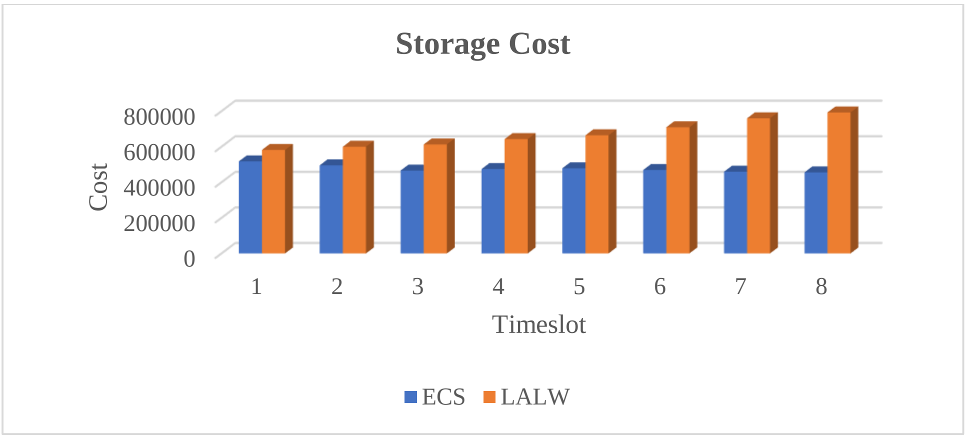

Figure 5.57 Storage Cost of ECS and LALW for 8 Timeslots

$$(\boldsymbol{C_{phy}} = \mathbf{3}, \boldsymbol{C_{main}} = \mathbf{9}, \boldsymbol{\lambda} = \mathbf{0.001}, \boldsymbol{T} = \mathbf{10}, \boldsymbol{Size_n} = \mathbf{64})$$

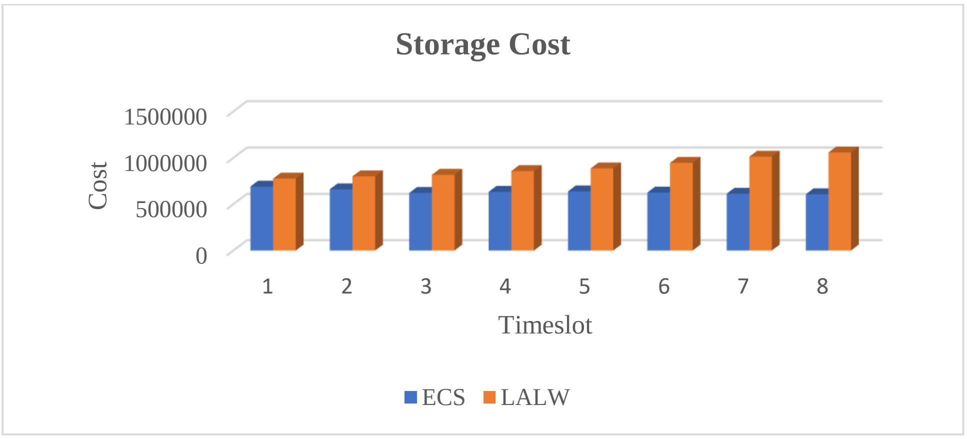

Figure 5.58 Storage Cost of ECS and LALW for 8 Timeslots

$$(\boldsymbol{C_{phy}} = \mathbf{4}, \boldsymbol{C_{main}} = \mathbf{12}, \boldsymbol{\lambda} = \mathbf{0.001}, \boldsymbol{T} = \mathbf{10}, \boldsymbol{Size_n} = \mathbf{64})$$

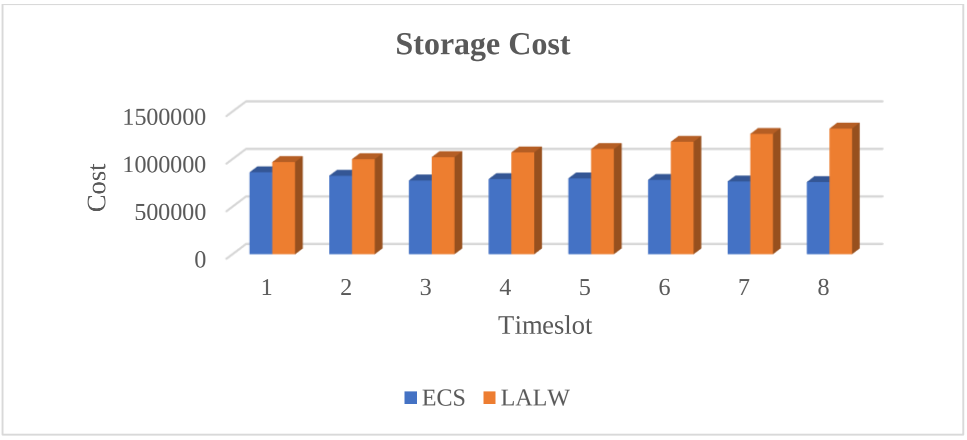

Figure 5.59 Storage Cost of ECS and LALW for 8 Timeslots

$$(\boldsymbol{C_{phy}} = \mathbf{5}, \boldsymbol{C_{main}} = \mathbf{15}, \boldsymbol{\lambda} = \mathbf{0.001}, \boldsymbol{T} = \mathbf{10}, \boldsymbol{Size_n} = \mathbf{64})$$

Figure 5.60 to 5.64 shows the comparison of storage cost of ECS and LALW for 8 timeslots with various physical cost and maintenance cost when failure rate in the system is 0.002. From the evaluation results, ECS does not cost more than LALW at 8 timeslots.

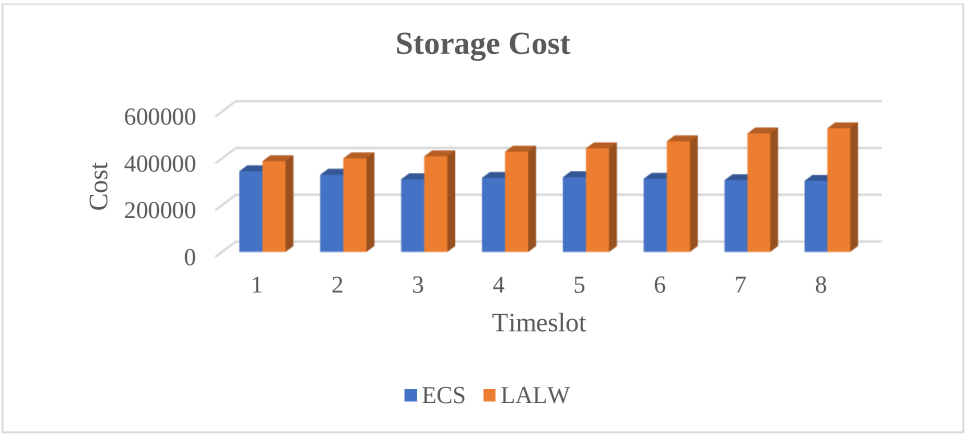

Figure 5.60 Storage Cost of ECS and LALW for 8 Timeslots
$(C_{phy} = 1, C_{main} = 3, \lambda = 0.002, T = 10, Size_n = 128)$

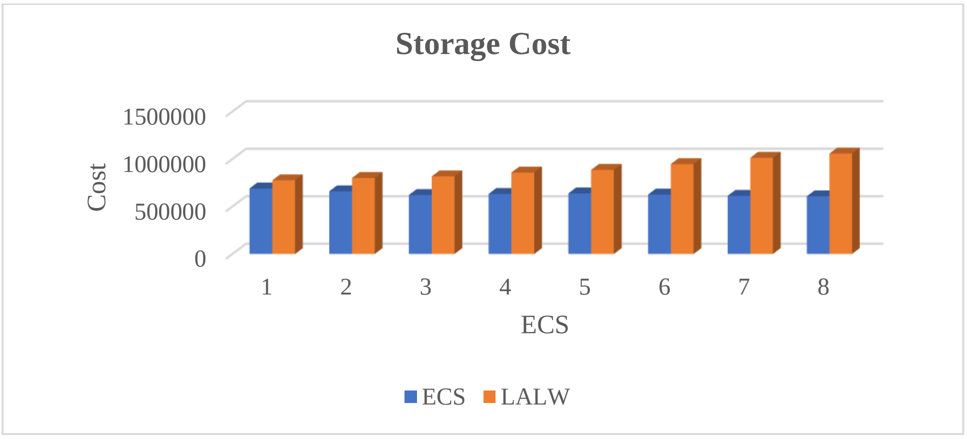

Figure 5.61 Storage Cost of ECS and LALW for 8 Timeslots
$(C_{phy} = 2, C_{main} = 6, \lambda = 0.002, T = 10, Size_n = 128)$

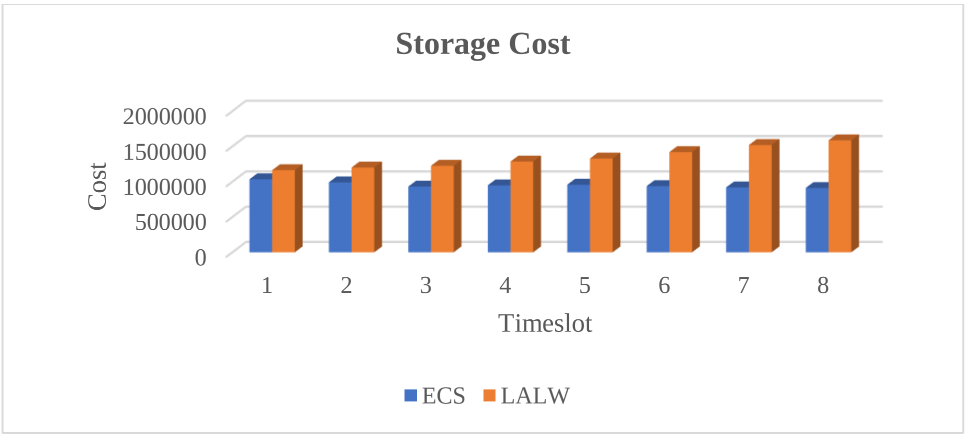

Figure 5.62 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 3, C_{main} = 9, \lambda = 0.002, T = 10, Size_n = 128)$$

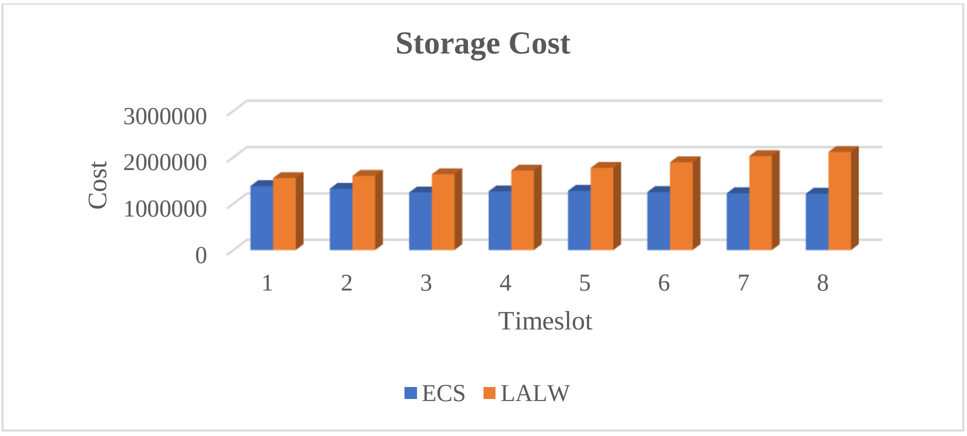

Figure 5.63 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 4, C_{main} = 12, \lambda = 0.002, T = 10, Size_n = 128)$$

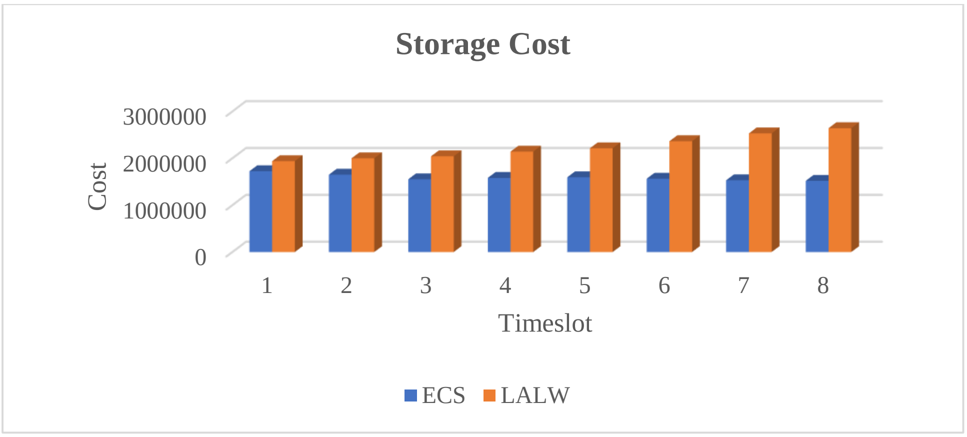

Figure 5.64 Storage Cost of ECS and LALW for 8 Timeslots

$$(C_{phy} = 5, C_{main} = 15, \lambda = 0.002, T = 10, Size_n = 128)$$

5.5.3 Disk Utilization

In this proposed system, the replicas are almost uniformly distributed for achieving the load balancing in nodes in the cluster. Disk utilization of ECS is compared with LALW algorithm in order to avoid overload condition. LALW does not obey the placement policy of hadoop because it places the same data replicas at one host. Therefore, LALW does not achieve the load balancing like the proposed system ECS. Figure 5.65 shows the average disk utilization comparison of ECS and LALW for 8 timeslots. From the evaluation results, ECS achieves more load balancing than LALW for 8 timeslots except cluster 1.

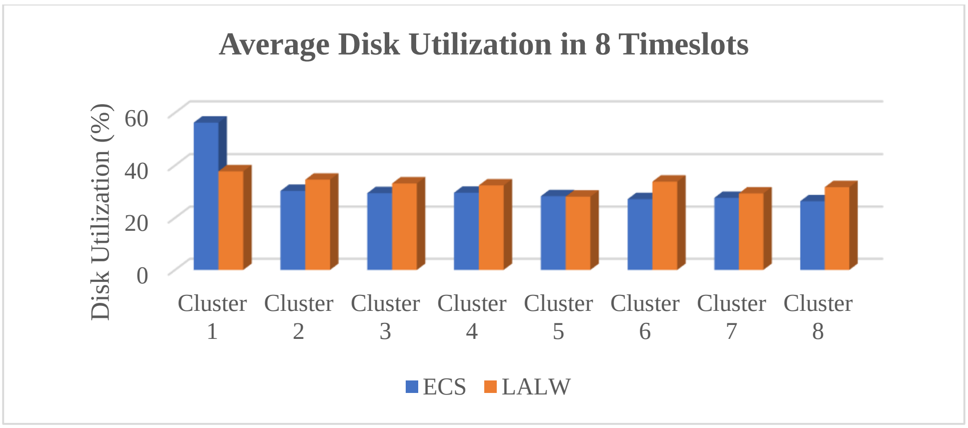

Figure 5.65 Average Disk Utilization of ECS and LALW for 8 Timeslots

5.6 Analysis of Load Factor in Data Placement

In this data placement, load factor of host or node is considered. The experiment is tested with varying the coefficient values of disk utilization $\boldsymbol{\alpha}$, disk bandwidth $\boldsymbol{\beta}$ and CPU utilization $\boldsymbol{\gamma}$. If the value of load factor at host is less than the predefined value of cluster, data is placed into node and if not, it is performed by replacing the replica having minimum access frequency with the new replica. To achieve the optimal coefficient values of disk utilization $\boldsymbol{\alpha}$, disk bandwidth $\boldsymbol{\beta}$ and CPU utilization $\boldsymbol{\gamma}$, the system parameters are varied according to Table 5.6.

Table 5.6 Parameters for Load Factor

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$
0.35	0.33	0.32

0.35	0.34	0.31
0.35	0.35	0.3
0.4	0.35	0.25
0.4	0.4	0.2
0.45	0.35	0.2
0.45	0.3	0.25
0.45	0.4	0.15
0.45	0.45	0.1
0.5	0.45	0.05
0.5	0.3	0.2
0.5	0.35	0.15
0.5	0.4	0.1
0.5	0.25	0.25
0.6	0.3	0.1
0.6	0.25	0.15
0.6	0.35	0.05
0.6	0.2	0.2
0.7	0.2	0.1
0.7	0.25	0.05
0.7	0.15	0.15
0.8	0.1	0.1
0.8	0.15	0.05
0.85	0.1	0.05

In the analysis of load factor, I varied the coefficient values of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ in order to get the optimal parameter of load factor. Firstly, I varied the coefficient value of $\boldsymbol{\alpha}$ while the coefficient value of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are set with fixed value. In that condition, I found that 0.35 is the best optimal value for disk utilization $\boldsymbol{\alpha}$. And then, I varied the coefficient value of $\boldsymbol{\beta}$ while the coefficient value of $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are set with fixed value in second evaluation. In that condition, I found that 0.33 is the best optimal value for disk bandwidth $\boldsymbol{\beta}$. And then, I varied the coefficient value of $\boldsymbol{\gamma}$ while the coefficient value of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are set with fixed value in third evaluation. In that case, I found that 0.32 is the best optimal value for CPU utilization $\boldsymbol{\gamma}$.

From the evaluation results, disk utilization $\boldsymbol{\alpha}$ = 0.35, disk bandwidth $\boldsymbol{\beta}$ = 0.33 and CPU utilization $\boldsymbol{\gamma}$ = 0.32 is the optimum parameter for the calculation of load factor in data placement. From Figure 5.46 to 5.53 shows the comparison of ($\boldsymbol{\alpha}$ = 0.35, $\boldsymbol{\beta}$ = 0.33, $\boldsymbol{\gamma}$ = 0.32) and ($\boldsymbol{\alpha}$ = 0.85, $\boldsymbol{\beta}$ = 0.1, $\boldsymbol{\gamma}$ = 0.05) for 8 clusters in the system. Figure 5.66 shows the load factor condition of fifty nodes in the cluster 1. According to first parameter, there is only one overload condition in this cluster, however, there is five overload condition in this cluster according to second parameter. Actually, there are enough storages for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

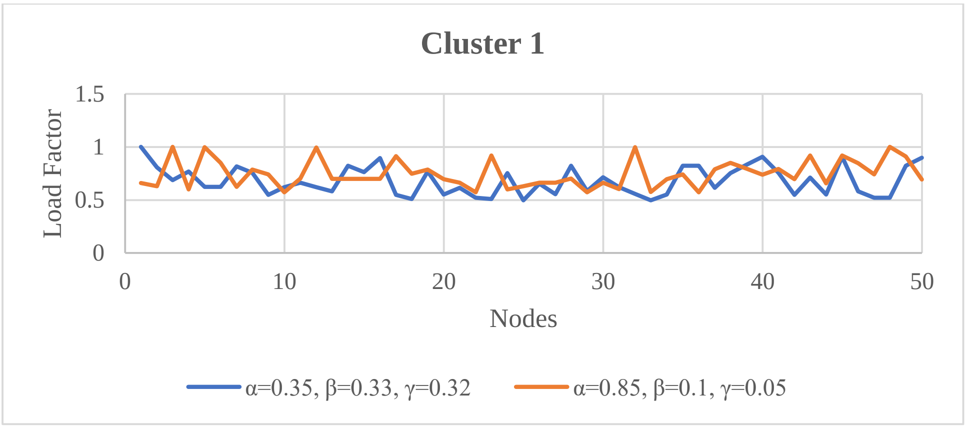

Figure 5.66 Load Factor of Cluster 1

Figure 5.67 shows the load factor condition of fifty nodes in the cluster 2. According to first parameter and second parameter, there is one overload condition in this cluster.

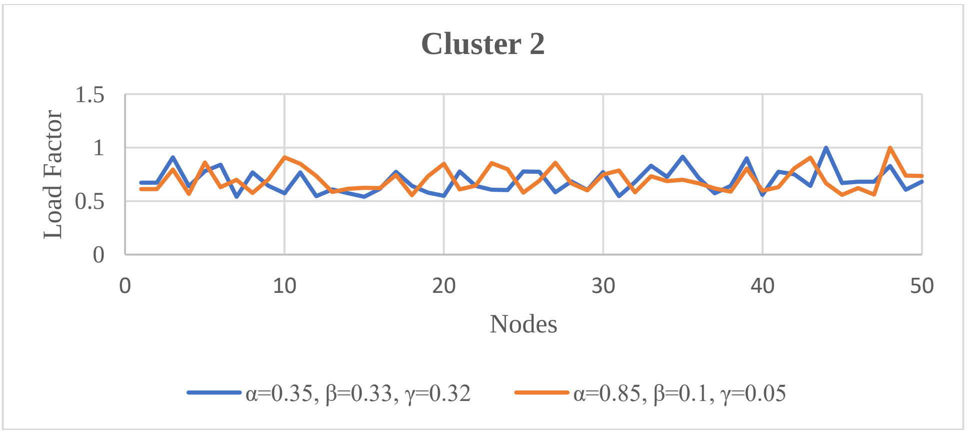

Figure 5.67 Load Factor of Cluster 2

Figure 5.68 shows the load factor condition of fifty nodes in the cluster 3. According to first parameter, there is almost four overload condition in this cluster, however, there is almost seven overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

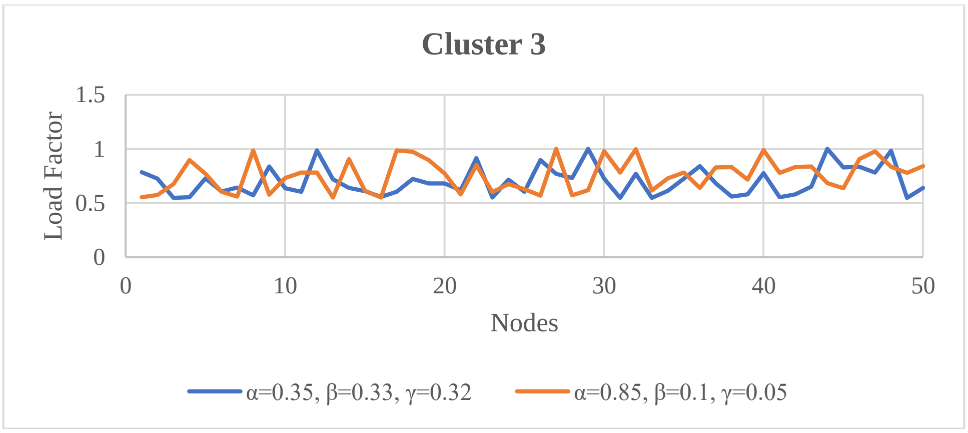

Figure 5.68 Load Factor of Cluster 3

Figure 5.69 shows the load factor condition of fifty nodes in the cluster 4. According to first parameter, there is only four overload condition in this cluster, however, there is almost five overload condition in this cluster according to second parameter. Actually, there are many enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

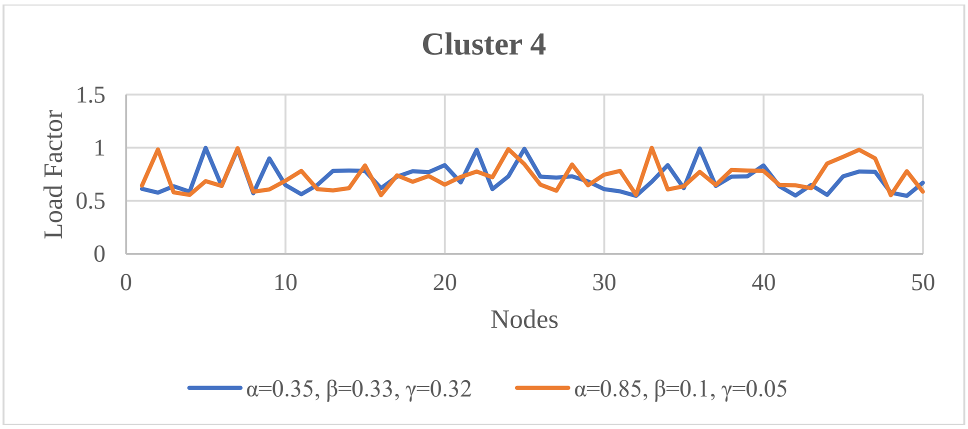

Figure 5.69 Load Factor of Cluster 4

Figure 5.70 shows the load factor condition of fifty nodes in the cluster 5. According to first parameter, there is only three overload condition in this cluster, and there is almost four overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not much different as the first parameter.

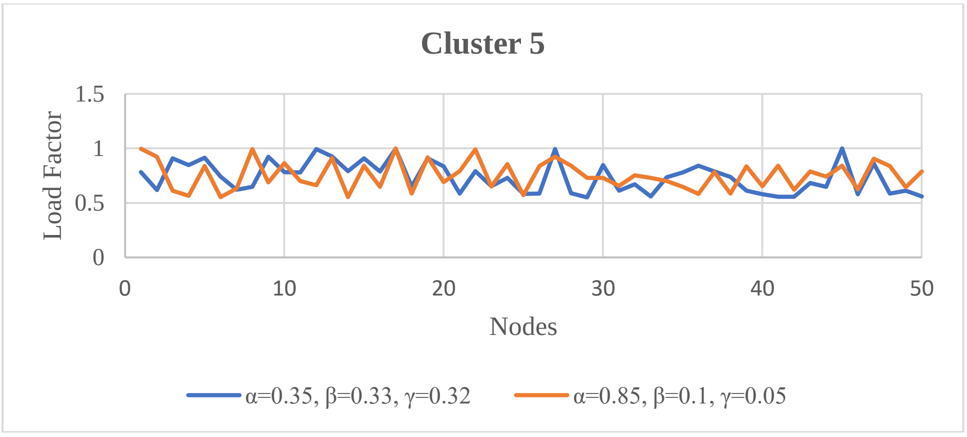

Figure 5.70 Load Factor of Cluster 5

Figure 5.71 shows the load factor condition of fifty nodes in the cluster 6. According to first parameter, there is only four overload condition in this cluster, however, there is seven overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

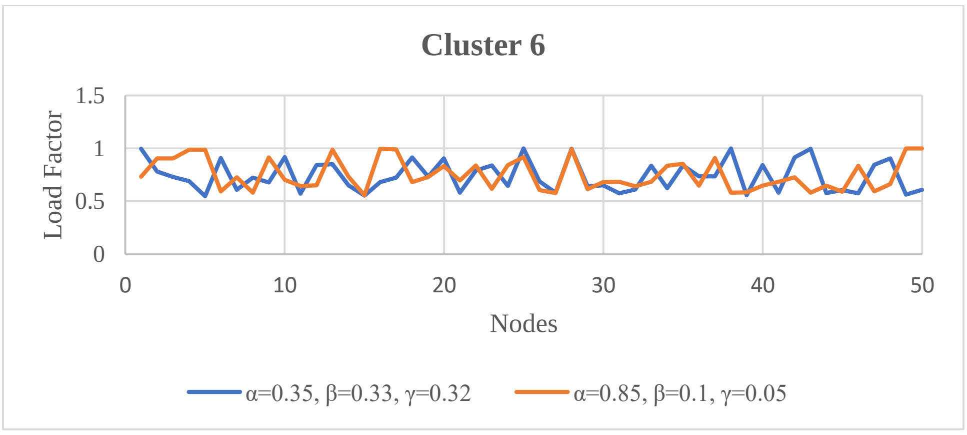

Figure 5.71 Load Factor of Cluster 6

Figure 5.72 shows the load factor condition of fifty nodes in the cluster 7. According to first parameter, there is three overload condition in this cluster, and there is almost six overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not much different as the first parameter.

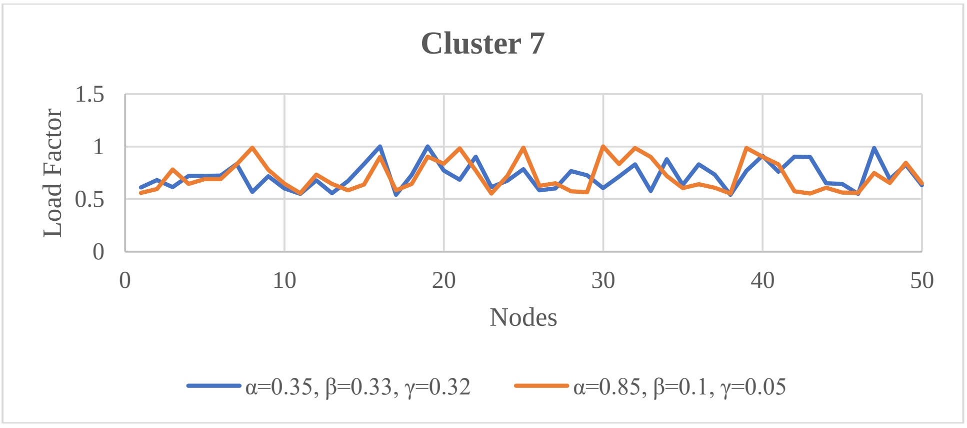

Figure 5.72 Load Factor of Cluster 7

Figure 5.73 shows the load factor condition of fifty nodes in the cluster 8. According to first parameter, there is four overload condition in this cluster, however, there is almost seven overload condition in this cluster according to second parameter. Actually, there are some enough storage for new replica for placement in nodes in this cluster. Therefore, the second parameter is not optimal as the first parameter.

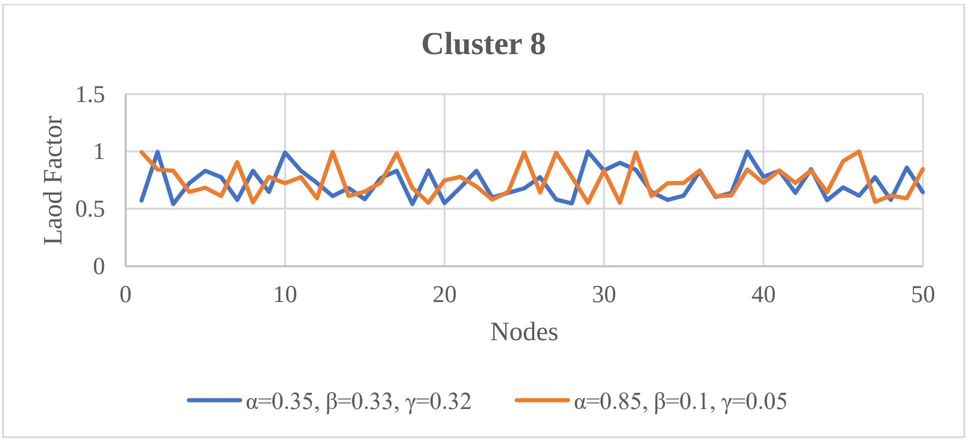

Figure 5.73 Load Factor of Cluster 8

From the evaluation results, the less the coefficient values of disk utilization, disk bandwidth and CPU utilization, the more getting the optimal parameter for load factor in data placement. The most optimum parameter can give the accurate detection of the overload condition in the cluster.

5.7 Summary

In this chapter, replication algorithms are proposed for Cloud data centers and compared with existing replication algorithm, LALW. To compare and evaluate the algorithms, cloud data center infrastructures are designed and simulated by using java programming language. To test actual situations, Yahoo web log data set is used to apply as data access pattern, which is the critical input for the proposed algorithm. The performance results of evaluation are produced in terms of created replicas, storage cost, and disk utilization. According to the evaluation results, the proposed system (ECS) can adapt the access pattern efficiently and reduce storage cost and load balancing much better than *LALW*.

CHAPTER 6
CONCLUSION AND FUTURE WORKS

Cloud storage provides a storage services that is hosted remotely on servers and users can access this through Internet. Data is replicated and stored in multiple data nodes to provide for data availability. This thesis proposes a dynamic replication management scheme for effective cloud storage. In this thesis, unpopular replica can be maintained due to the calculation of popularity of certain time. This factor would be effectiveness on utilization of disk bandwidth, CPU and disk utilization of a node while replicating the data through proposed replica placement algorithm. In reality, the proposed replica algorithm will be performed to be the better load balancing and effective storage utilization compared with existing replication LALW algorithm. The data access frequencies obtained from Yahoo audit log file data source. In this chapter, the main contents of thesis are summarized and future work is suggested.

6.1 Thesis Summary

Currently, well-known cloud storage systems such as GFS, HDFS, and Cassandra, etc. are using static replication scheme, which is simple and straightforward technique for replication management. However, the static replication is not efficient for all data and the degree of replication can affect system performance. In chapter 3, different strategies and models of data replication and data locality are presented. Among replication strategies, static method is more common in today cloud storage system because of simple and straightforward technique. However, it results more storage cost and less availability in very large storage systems as some data files may not need as many as static replication factor due to lack of usage. At the same time, some have to be replicated more than static replication factor to recover highly concurrent access. As a result, dynamic replication becomes an important strategy to cope the weakness of static method. Therefore, most parts of this chapter 3 presents different approaches of dynamic replication which are intended to play a vital role in today cloud storage systems. In chapter 3, several research areas are studied to improve the performance of data locality and evaluated their research outcomes in various environments such as dedicated and shared environment.

A replication management strategy for effective cloud storage (ECS) is proposed in chapter 4. The system contains two portions; replica allocation and replica placement. In the first portion, replica allocation, popularity is taken into account by analyzing the changes in data access pattern. Second, for replica placement, replicas are placed and performed on dedicated assigned nodes in order to enhance data locality. The proposed placement algorithm is able to avoid the overloaded problem of nodes by considering the load of nodes; that is, disk utilization, CPU utilization and adjustable disk bandwidth.

With attention to this, the distributed cloud storage system is simulated with popularity-based replication management strategy in chapter 5. To evaluate the system more realistic, Yahoo Weblog data set is used to extract data access pattern and data popularity.

Data replication is a technique commonly used to improve data availability, throughput and response time for user while it plays an important role for storage system to reduce storage cost. Especially, for cloud environment, replication is the key to improve the performance so that services can be provided to users as an agreement of SLAs. In the proposed system, efficient replication management strategies are proposed for cloud storage by implementing and analyzing different ways. The experimental results show that the proposed strategies are able to apply in different environments.

6.2 Advantages and Disadvantages of Proposed Scheme (ECS)

The advantages of this system are the following. First, the proposed system can adapt the degree of replication based on data popularity. Second, it saves storage cost for unpopular files than existing replication strategy such as LALW. Third, it achieves more load balancing than *LALW* algorithm. Fourth, it considers the heterogeneous conditions of nodes in the cluster. Fifth, it does not place the replicas randomly and it obeys the placement policy of Hadoop. Sixth, it considers the load factor of nodes before placement of replicas into nodes in the cluster to avoid the overloaded condition of the cluster in cloud storage.

The proposed scheme (ECS) has some problems concerning with time complexity. As ECS performs the calculation of the rate of change of file popularity from the access frequencies of all data, then, the determination of the increment and decrement of the number of replicas for all data and finally, placement of these replicas into nodes to achieve load balancing, ECS takes more processing time than the existing replication algorithms.

6.3 Further Extension

In storage cluster implementation case, the proposed system is implemented as cloud storage system by using open-source CloudSim simulator. One aspect to extend the system is that the replication strategy could be analyzed in different storage frameworks. Then the second aspect is to extend the storage system by connecting cloud computing infrastructure.

As the main part of thesis, replication algorithms are proposed and compared with other existing algorithms. As a future work, the replication algorithms could be upgraded by implementing in various distributed file systems such as Lustre file system, Google file system and Gluster file system etc.

AUTHOR'S PUBLICATIONS

[P1] May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye, "Dynamic Replication Management Scheme for Hadoop Distributed File System", 15th International Conference on Computer Applications **(ICCA 2017)**, Yangon, MYANMAR, February 2017. **Page [55-61]**

[P2] May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye, "Dynamic Replication Management Scheme for Cloud Storage", The 1st International Conference on Advanced Information Technologies **(ICAIT 2017)**, Yangon, MYANMAR, ISBN 978-99971-0-381-9, November 2017. **Page [1-7]**

[P3] May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye, "Dynamic Replication Management Scheme for Distributed File System", First International Conference on Big Data Analysis and Deep Learning Applications **(ICBDL 2018)**, Miyazaki, JAPAN, May 2018.

[P4] May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye, "Dynamic Replication Management Scheme for Distributed File System", **Advances in Intelligent Systems and Computing (Book Series Volume: 744), Springer. (Scimago index –Q3)**

[P5] May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye, "Replication Based on Data Locality for Hadoop Distributed File System", 2019 The 9th International Workshop on Computer Science and Engineering **(WCSE 2019)**, HONG KONG, ISBN 978-981-14-1684-2, June 2019. **(Scimago index) Page [663-667]**

[P6] May Phyo Thu, Khine Moe Nwe, Kyar Nyo Aye, "Data Popularity - Aware Replication Strategy for Cloud Storage", **Iconic Research And Engineering Journals (IRE Journals) -International Peer Reviewed Journal,** e-ISSN: 2456-8880**,** Volume 3, Issue 2, August 2019**. Page [494-500]**

BIBLIOGRAPHY

[1] C. L. Abad, N. Roberts, Y. Lu, and R. H. Campbell, “A storage-centric analysis of MapReduce workloads: File popularity, temporal locality and arrival patterns,” 2012 IEEE Int. Symp. Workload Charact., pp. 100–109, Nov. 2012.

[2] C. L. Abad, Y. Lu, and R. H. Campbell, “DARE: Adaptive Data Replication for Efficient Cluster Scheduling,” 2011 IEEE Int. Conf. Clust. Comput., pp. 159–168, Sep. 2011.

[3] P. Across and H. Hardware, “The Hadoop Distributed File System: Architecture and Design,” pp. 1–14, 2007.

[4] I. Adel Ibrahim, W. Dai, and M. Bassiouni, “Intelligent Data Placement Mechanism for Replicas Distribution in Cloud Storage Systems,” 2016 IEEE International Conference on Smart Cloud, 2016.

[5] G. Alonso, D. Barbara, and H. Garcia-Molina, “Data caching issues in an information retrieval system,” ACM Trans. Database Syst., vol. 15, no. 3, pp. 359–384, Sep. 1990.

[6] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, “Scarlett : Coping with Skewed Content Popularity in MapReduce Clusters,” in Proceedings of the sixth conference on Computer systems-EuroSys ’11 (2011), 2011, pp. 287–300.

[7] T. Anderson, M. Hill, Y. Breitbart, and H. F. Korth, “Replication, Consistency, and Practicality : Are These Mutually Exclusive ?,” ACM Sigmod’ 98, 1998.

[8] H. Aung, and N. Nyein Oo, “EDAS: Efficient Data Access Scheme of Data Replication for Hadoop Distributed File System (HDFS),” ICFCT'2015, pp. 177-183, 2015.

[9] C.Baun, M.Kunze, J.Nimis, and S.Tai, “Cloud Computing: Web-Based Dynamic IT Services”, Springer-Verlag Berlin Heidelberg, 2011.

[10] O. Beaumont, T. Lambert, L. Marchal, and B. Thomas, “Matching-Based Allocation Strategies for Improving Data Locality of Map Tasks in MapReduce,” [Research Report] RR-8968, Inria - Research Centre Grenoble – Rhône-Alpes; Inria Bordeaux Sud-Ouest, 2016.

[11] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, F. Zini), " Evaluation of an economy-based file replication strategy for a data grid," In: Proceedings of the 3rd IEEE/ACM international symposium on cluster computing and the grid (CCGRID'03), pp 661–668, 2003.

[12] I. Bin Abdullah, "Incremental PageRank for Twitter Data Using Hadoop," University of Edinburgh, 2010.

[13] R. Calheiros, R. Ranjan, A. Beloglazov, C. Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software-Practice and Experience, vol.41, no.1, 2011.

[14] M. J. Carey and M. Livny, "Conflict Detection Replicated Data Tradeoffs for Replicated Data," ACM Trans. Database Syst., vol. 16, no. 4, pp. 703–746, 1991.

[15] P. H. Carns, W. B. L. Iii, and R. B. Ross, "PVFS : A Parallel File System for Linux Clusters," in In Proceedings of the 4th Annual Linux Showcase and Conference, 2000, no. October, pp. 317–327.

[16] I. Casas, J. Taheri, R. Ranjanb, L. Wangc, and A. Y. Zomaya, "A Balanced Scheduler with Data Reuse and Replication for Scientific Workflows in Cloud Computing Systems," Future Generation Computer Systems, 2016.

[17] Z. Challal, and T. Bouabana-Tebibel, "A priori replica placement strategy in data grid," In: International conference on machine and web intelligence, pp 402–406, 2010.

[18] R. Chang, H. Chang, and Y. Wang, "A dynamic weighted data replication strategy in data grids," 2008 IEEE/ACS Int. Conf. Comput. Syst. Appl., pp. 414–421, Mar. 2008.

[19] L. Chen, and D.B. Hoang, " Adaptive data replicas management based on active data-centric framework in cloud environment," In High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), pp. 101-108, 2013.

[20] Z.D. Cheng, Z.Z. Luan, Y. Meng, Y.J. Xu and D.P. Qian. "ERMS: An Elastic Replication Management System for HDFS", 2012 IEEE International Conference on Cluster Computing Workshops, pp. 32-40,

2012.

[21] L. Cherkasova, M. Gupta, and I. Systems, “Analysis of Enterprise Media Server Workloads : Access Patterns , Locality , Dynamics , and Rate of Change,” IEEE/ACM Trans. Netw., no. May, pp. 12–14, 2004.

[22] B. Ciciani, D. M. Dias, and P. S. Yu, “Analysis of replication in distributed database systems,” IEEE Trans. Knowl. Data Eng., vol. 2, no. 2, pp. 247–261, Jun. 1990.

[23] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, “Copysets:reducing the frequency of data loss in cloud storage,” Proceedings of the 2013 USENIX Annual Technical Conference, pp.37–48, 2013.

[24] C. Debians, P.T. Togores, and F. Karakusoglu, “Hdfs replication simulator,” https://github/peteratt/HDFS-Replication-Simulator, 2012.

[25] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, and J. Good, “Pegasus: A framework for mapping complex scientific workflows onto distributed systems,” Scientific Programming, Volume 13, Issue 3, pp.219-237, 2005.

[26] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li, “A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files,” 2010 IEEE Int. Conf. Serv. Comput., pp. 65–72, Jul. 2010.

[27] D. Fesehaye and N. G. Ave, “A Scalable Distributed File System for Cloud Computing,” 2010.

[28] J X. Gao, Y. Ma, and M. Pierce, “VBS-Lustre : A Distributed Block Storage System for Cloud.”

[29] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” ACM SIGOPS Oper. Syst. Rev., vol. 37, no. 5, p. 29, Dec. 2003.

[30] N. Grozev and R. Buyya, “Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments,” The Computer Journal, vol.58, no.1, 2015.

[31] Z. Guo, G. Fox, and M. Zhou, “Investigation of Data Locality in MapReduce,” In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 419-

426, 2012.

[32] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-dusseau, and R. H. Arpaci-dusseau, “Analysis of HDFS Under HBase : A Facebook Messages Case,” FAST 2014, 2014.

[33] C. He, Y. Lu, and D. Swanson, “Matchmaking: A New MapReduce Scheduling Technique,” 2011 IEEE Third International Conference on Cloud Computing Technology and Science, pp. 40–47, 2011.

[34] A. Hunger and J. Myint, “Comparative Analysis of Adaptive File Replication Algorithms for Cloud Data Storage”, 2014 International Conference on Future Internet of Things and Cloud, 2014.

[35] S. Ibrahim, H. Jin, and L. Lu, “Maestro: Replica-Aware Map Scheduling for MapReduce,” 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 435–442, 2012.

[36] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, “Quincy: Fair Scheduling for Distributed Computing Clusters,” Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 261–276, 2009.

[37] K. Jackson, “OpenStack Cloud Computing Cookbook”, September 25, 2012.

[38] L. Jiang, B. Li and M. Song, “The Optimization of HDFS Based on Small Files”, In Proceedings of IC-BNMT20 10, the 3rd IEEE International Conference on Broadband Network& Multimedia Technology, pp. 912-915, 2010.

[39] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, “Bar: An Efficient Data Locality Driven Task Scheduling Algorithm for Cloud Computing,” In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2011), pp. 295–304, 2011.

[40] K. Kaur, “Eucalyptus Cloud Computing Architecture”, 2017.

[41] Rini T. Kaushik and M. Bhandarkar, “GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster,” In Proceedings of the 2010 international conference on Power aware computing and systems, 2010.

[42] K.Keahey, and T.Freeman, “Contextualization Providing One-Click Virtual

Clusters", In Proceedings of the 4th IEEE International Conference on eScience, pp.301-308, 2008.

[43] L.M. Khanli, A. Isazadeh, T.N. Shishavanc, "PHFS: A Dynamic Replication Method, To Decrease Access Latency in Multi-Tier Data Grid", Future Generation Computer Systems, 2010.

[44] J. H. Kim, K. Kim and G. Fox, "Analysis for Optimal Degree of Replication", 2012.

[45] G. Kousiouris, G. Vafiadis, and T. Varvarigou, "Enabling proactive data management in virtualized hadoop clusters based on predicted data activity patterns," Proceedings of the Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2013.

[46] C.Krintz, "Appscale: An open-source research framework for execution of Google AppEngine applications and investigation of scalable cloud computing fabrics", In Proceedings of Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, pp.1-14, 2010.

[47] C. Krintz, "The AppScale Cloud Platform – NCBI", 2013.

[48] Y. Labs, "S3 - Yahoo! Hadoop grid logs, version 1.0 (Hosted in AWS)," 2013.[Online].Available:http://webscope.sandbox.yahoo.com/catalog.php?datatype=s.

[49] J. Lee, L. JongBeom, Y. Heonchang, J. Daeyong, C. KwangSik, and G. JoonMin, "Adaptive Data Replication Scheme Based on Access Count Prediction in Hadoop," The World Congress in Computer Science, Computer Engineering, and Applied Computing, 2013.

[50] E. Levy and A. Silberschatz, "Distributed File Systems: Concepts and Examples," ACM Comput. Surv., vol. 22, no. 4, 1990.

[51] W. Li, Y. Yang, J.Chen, and D.Yuan, "A cost-effective mechanism for cloud data reliability management based on proactive replica checking," Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012.

[52] Y. Lin, and H. Shen, "EAFR: An Energy-Efficient Adaptive File Replication System in Data-Intensive Clusters," Journal of latex class files, vol. 13, no. 9, 2014.

[53] F. Luo, J. Yi, and H. Yu, "Elastic Replication on the Metadata in Object-based Storage Systems," 2013 International Conference on Cloud Computing and Big Data, 2013.

[54] G. Mackey, S. Sehrish, and J. Wang, "Improving metadata management for small files in HDFS," 2009 IEEE Int. Conf. Clust. Comput. Work., pp. 1–4, 2009.

[55] M. Malak, "Parallel vs. Distributed file systems: Time for RAID on Hadoop?", 2014.

[56] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware Scheduling of MapReduce Jobs for Big Data Applications," IEEE Transactions on Parallel and Distributed Systems, Volume: 26, Issue: 10, 2015.

[57] T. Mather, S. Kumaraswamy, and S. Latif, Cloud Security and Privacy : An Enterprise Perspective on Risks and Compliance. 2009.

[58] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," 2011.

[59] S. Microsystems, "LUSTRE TM FILE SYSTEM," 2007.

[60] R. Mukkamala, "Measuring the effects of data distribution models on performance evaluation of distributed database systems," IEEE Trans. Knowl. Data Eng., vol. 1, no. 4, pp. 494–507, 1989.

[61] E.I. Neaga and A.V. Gheorghe, "A System-of-Systems Standardized Architectural Approach Driven by Cloud Computing Paradigm ", The 4th Annual International Conference on Next Generation Infrastructures, Virginia Beach, Virginia, November 16-18, 2011.

[62] M. Nicola, "Performance Modeling of Distributed and Replicated Databases," IEEE Trans. Knowl. Data Eng., pp. 645–672, 2000.

[63] N.E.Pius, L.Qin, F.Yang and Z.H.Ming, "Optimizing Hadoop Block Placement Policy and Cluster Blocks Distribution", In Proceedings of Work Academy of Science, Engineering and Technology, pp. 1117-1123, 2012.

[64] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving data availability

through dynamic model- driven replication in large peer-to-peer communities,” In: Proceedings of the 2nd IEEE/ACM international symposium on cluster computing and the grid (CCGRID’02), pp 376–381, 2002.

[65] K. Ranganathan, and I. Foster, “Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications,” In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, 2002.

[66] A. Rani, and R.K. Yadav, “A fuzzy based approach for effective data replication in peer-to-peer networks,” International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 7, pp 2921-2926, 2013.

[67] S. Seo, I. Jang, K. Woo, I. Kim, J. Kim, and S. Maeng, "HPMR: Prefetching and Pre-shuffling in Shared MapReduce Computation Environment", IEEE International Conference on Cluster Computing (CLUSTER 2009), pp.1-8, 2009.

[68] M. Shorfuzzaman, “Access-Efficient QoS-Aware Data Replication to Maximize User Satisfaction in Cloud Computing Environments,” 2014 15th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 13- 20, 2014.

[69] E. Sit, A. Haeberlen, F. Dabek, B. Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek, and J. Kubiatowicz, “Proactive replication for data durability,” in In Proceedings of the 5th Int’l Workshop on Peer-to-Peer Systems (IPTPS, 2006).

[70] M. Sun, H. Zhuang, X. Zhou, K. Lu, and C. Li, “HPSO: Prefetching Based Scheduling to Improve Data Locality for MapReduce Clusters,” International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), in: Lecture Notes in Computer Science, vol. 8631, Springer, pp. 82–95, 2014.

[71] F. Systems, “GFS : Evolution on Fast-forward,” pp. 1–11, 2009.

[72] J. Tan, S. Meng, X. Meng, and L. Zhang, “Improving Reduce Task Data Locality for Sequential MapReduce Jobs,” 2013 Proceedings IEEE INFOCOM, pp. 1627–1635, 2013.

[73] J. Tan, X. Meng, L. Zhang, “Coupling Task Progress for MapReduce Resource-Aware Scheduling,” 2013 Proceedings IEEE INFOCOM, Turin, Italy, pp. 1618–1626, 2013.

[74] Andrew S. Tanenbaum. Modern Operating Systems. Prentice-Hall, 1992.

[75] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, “Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality,” IEEE/ACM Transactions on Networking, vol. 24, no. 1, pp. 190–203, 2013.

[76] K. Wang, X. Zhou, T. Li, D. Zhao, M. Lang, and I. Raicu, “Optimizing Load Balancing and Data-Locality with Data-Aware Scheduling,” 2014 IEEE International Conference on Big Data (Big Data), pp. 119–128, 2014.

[77] J. Wang, H. Wub, and R.Wand, “A new reliability model in replication based big data storage systems,” Parallel and Distributed Computing, 2017.

[78] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, “CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster,” 2010 IEEE Int. Conf. Clust. Comput., pp. 188–196, Sep. 2010.

[79] R. Xiong, J. Luo, and F. Dong, “SLDP: a Novel Data Placement Strategy for Large-Scale Heterogeneous Hadoop Cluster,” 2014 Second International Conference on Advanced Cloud and Big Data, 2014.

[80] J. Pin Yang, “Efficient Load Balancing Using Active Replica Management in a Storage System,” Mathematical Problems in Engineering Volume 2016, Article ID 4751829, 2016.

[81] C. Yi Lin, and Y. Chen Lin, “A Load-Balancing Algorithm for Hadoop Distributed File System,” 2015 18th International Conference on Network-Based Information Systems, 2015.

[82] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling,” In Proceedings of the 5th European conference on Computer systems, pp.265-278, 2010.

[83] X. Zhang, Y. Feng, S. Feng, J. Fan, and Z. Ming, "An Effective Data Locality Aware Task Scheduling Method for MapReduce Framework in Heterogeneous Environments,” International Conference on Cloud and Service Computing (CSC), 2011.

[84] W. Zhao, L. Meng, J. Sun, Y. Ding, H. Zhao, and L. Wang, "An Improved Data Placement Strategy in a Heterogeneous Hadoop Cluster," The Open Cybernetics & Systemics Journal, 2014, Volume 8, 957-963, 2014.

[85] "Apache CloudStack: Open Source Cloud Computing", https://cloudstack.apache.org/

[86] "CloudSim", https://github.com/Cloudslab/cloudsim/releases.

[87] "CloudSimEx", https://github.com/Cloudslab/CloudSimEx.

[88] "Maintenance Cost Estimation", http://st.inf.tu-dresden.de/files/teaching/ws12/ring/03-Planning-Software-Evolution.pdf

[89] "Nimbus: cloud computing for science – Quintagroup", https://quintagroup.com/cms/python/nimbus.

[90] "The Promise & Challenges of Cloud Storage", 2013. http://www.imexresearch.com.

LIST OF ACRONYMS

2D	Two-Dimensional
ABW	Adjustable Bandwidth
AWS	Amazon Web Services
BW	Bandwidth
C	Cluster
CPU	Central Processing Unit
DC	Datacenter
DFS	Distributed File System
DN	DataNode
EBS	Elastic Block Store
GFS	Google File System
HDFS	Hadoop Distributed File System
IaaS	Infrastructure-as-a-service
LALW	Latest Access Largest Weight
LF	Load Factor
LFU	Least Frequently Used
LRU	Least Recently Used
MT	Map Task
PaaS	Platform-as-a-service
RAM	Random Access Memory
RP	Replica
S3	Simple Storage Service
SaaS	Software-as-a-service
SLA	Service Level Agreement

U	Disk Utilization
VM	Virtual Machine
YARN	Yet Another Resource Negotiator